ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку з елементами геймифікації для спілкування та знайомства з використанням JS, Vue, TS»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Михайло КРИВОРУЧКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-44
Михайло КРИВОРУЧКО

Керівник: Владислав ЯСКЕВИЧ
к.т.н.

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Криворучку Михайлу Олеговичу

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку з елементами геймифікації для спілкування та знайомства з використанням JS,Vue,TS»
керівник кваліфікаційної роботи к.т.н., доцент кафедри ІПЗ Владислав ЯСКЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.
2. Строк подання кваліфікаційної роботи «28» травня 2024 р.
3. Вихідні дані до кваліфікаційної роботи: теоретичні матеріали про соціальну взаємодію, причини виникнення та наслідкову самотність.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих рішень для поліпшення соціальної взаємодії та зменшення самотності.

2. Визначення вимог та проектування застосунку для сприяння соціальної взаємодії та зменшення самотності.
3. Програмна реалізація та опис функціонування застосунку, який буде сприяти соціальній взаємодії та зменшення самотності.
4. Аналіз відповідності функціональним та нефункціональним вимогам
5. Перелік графічного матеріалу: *презентація*
    1. Аналіз аналогів.
    2. Вимоги до web-застосунку.
    3. Програмні засоби реалізації.
    4. Діаграма прецедентів.
    5. Архітектура web-застосунку
    6. Діаграма класів.
    7. Карта web-застосунку
    8. Екранні форми.
    9. Демонстрація роботи web-застосунку
    10. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02 -06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03 -13.03.2024	
3	Визначення концепції застосунку, що найкраще поліпшує соціальну взаємодію та зменшує самотність	14.03 -20.03.2024	
4	Проектування застосунку для підтримки соціальної взаємодії та зменшення самотності з елементами геймифікації	21.03 -29.03.2024	
5	Програмна реалізація застосунку	30.03 -19.04.2024	
6	Перевірка функціональних та нефункціональних вимог	20.04 - 29.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	30.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Михайло КРИВОРУЧКО
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 1 табл., 35 рис., 11 джерел.

Мета роботи – підтримка процесів соціальної взаємодії, знайомства та зменшення самотності з допомогою елементів геймифікації.

Об'єкт дослідження – соціальна взаємодія, знайомство та самотність.

Предмет дослідження – застосунок для соціальної взаємодії з елементами геймифікації.

Короткий зміст роботи: в роботі проведений аналіз предметної галузі, визначені проблеми які виникають у суспільстві в контексті соціальної взаємодії та самотності,. Визначені чинники, що впливають на самотність, зокрема емоційна самотність, соціальна самотність. Проведено аналіз існуючих застосунків для розвитку соціальної взаємодії та зменшення самотності.

Розроблено концепцію web-застосунку для карткової гри з елементами геймифікації спрямованого на покращення комунікаційних навичок та зменшення самотності, емоційної самотності зокрема. Програмно реалізовані ключові необхідні можливості, зокрема: авторизація користувачів, можливість карткової гри за темами, можливість створювати власні теми з питаннями, можливість ділитися створеними темами між користувачами, досягнення для заохочення використати весь функціонал застосунку, накопичувальні бали для більшого залучення користувачів. В роботі використано Vue для реалізації прогресивного web-застосунку, TypeScript для статичної типізації для кращої структуризації коду, Nuxt для серверної частини додатку та структуризації коду, Supabase для інтеграції з базою даних.

Сферою використання застосунку є підтримка соціальної взаєм ... боротьби із самотністю.

КЛЮЧОВІ СЛОВА: КАРТКОВА ТЕМА, АККАУНТ, АВТОРИЗАЦІЯ.

ЗМІСТ

ВСТУП

Актуальність теми. У суспільстві дедалі зростає соціальна ізоляція та самотность, що має чимало негативних ефектів на психічне здоров'я та загальне самопочуття[1]. Тема стала актуальнішою після пандемії Covid-19, яка поширила відчуття самотності серед молоді [2]. Зокрема в Україні після повномасштабного вторгнення Російської Федерації у 2022 році через вимушений переїзд самотність теж стала більшою проблемою [3].

Деякі дослідження говорять, що самотність є ключовим показником, що впливає на якість життя. Тривале відчуття самотності негативно впливає на ваше харчування, викликає проблеми з серцем та мотивацією щодо повсякденних занять [3][4].

Самотність поділяється на два типи:

- соціальна самотність - нестача людей з якими можна поговорити та поділитися власним досвідом. Людина не відчуває себе частиною певної спільноти;
- емоційна самотність - нестача саме близьких стосунків та емоційного зв'язку з людьми. Може виникати навіть коли є багато людей у колі спілкування, але розмови є поверхневими;
- екзистенційна самотність - також іноді виділяться цей тип самотності, коли люди не бачать загального сенсу у житті [5];

Цифрові рішення з елементами геймифікації можуть сприяти залученості студентів у розвиток соціальних навичок, створювати соціальні можливості, заохочувати до постійного спілкування. Це усе сприяє значному зменшенню самотності через відчуття приналежності до спільноти [6][7].

Об'єкт дослідження – соціальна взаємодія, знайомство та самотність.

Предмет дослідження – застосунок для соціальної взаємодії з елементами геймифікації.

Мета роботи – підтримка процесів соціальної взаємодії, знайомства та

зменшення самотності з допомогою елементів геймифікації.

Методи дослідження – порівняльний метод дослідження був використаний для аналізу та оцінки різних підходів та інструментів, що застосовуються у розробці та впровадженні web-застосунків з елементами гейміфікації. Цей метод дозволив провести детальне зіставлення функціональних можливостей, переваг та недоліків кількох популярних додатків, таких як Happify, Noneliness та Playersgetready. Завдяки цьому вдалося визначити найефективніші практики та підходи, які можна інтегрувати у власний проект. Це метод допоміг визначити найбільш дієві стратегії та інструменти для реалізації web-застосунку, спрямованого на покращення соціальної взаємодії та зменшення самотності серед користувачів. Аналіз досвіду інших додатків дозволив уникнути поширених помилок і впровадити перевірені рішення, що підвищують ефективність та привабливість застосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Соціальна ізоляція та самотність

Соціальна ізоляція та самотність є серйозними проблемами сучасного суспільства, що впливають на мільйони людей по всьому світу. Соціальна ізоляція виникає, коли людина має обмежений контакт з іншими, тоді як самотність — це суб'єктивне відчуття відсутності емоційного зв'язку з іншими людьми. Обидва явища можуть мати значний негативний вплив на психічне та фізичне здоров'я, включаючи підвищений ризик депресії, тривоги, серцево-судинних захворювань та навіть передчасної смертності [1].

Пандемія COVID-19 значно посилила ці проблеми, змушуючи людей дотримуватися соціальної дистанції та проводити більше часу вдома, що зменшило можливості для соціальної взаємодії. Це підкреслило важливість пошуку нових способів підтримки соціальних зв'язків та інтеграції технологічних рішень для боротьби з ізоляцією та самотністю. В Україні проблема набула більшого загострення після повномасштабного вторгнення Російської Федерації у 2022 році, змушуючи людей переїзджати [3].

1.1.1 Цифрові рішення

Цифрові рішення, зокрема web-застосунки з елементами геймифікації, стали перспективним підходом до вирішення проблеми соціальної ізоляції та самотності. Геймифікація включає використання ігрових механік, таких як нагороди, бали, рівні та досягнення, для підвищення залученості користувачів та покращення їхнього досвіду.

Дослідження показують, що геймифікація може значно підвищити ефективність цифрових рішень, сприяючи залученню користувачів та їхньому емоційному благополуччю. Інтерактивні платформи, які використовують геймифікацію, можуть стати потужним інструментом для боротьби з соціальною ізоляцією та самотністю, допомагаючи людям підтримувати зв'язки

та знаходити нові шляхи соціальної взаємодії.

Існує широкий спектр застосунків, що допомагають у певних аспектах проблеми соціальної взаємодії та самотності, а саме:

- застосунки зосереджені на збільшення соціальних можливостей для зустрічі у різних контекстах. Це допомагає людям із соціальною самотністю;
- застосунки, які розвивають навички комунікації з наявним колом спілкування. Це можуть бути набори питань або курси з розвитку певних аспектів свого характеру. Це дозволить розвивати ближчі та інтимніші стосунки, що допомагає з емоційною самотністю.

1.2 Аналіз існуючих цифрових рішень

1.2.1 Noneliness

Застосунок Noneliness створив Роджеріо Аугусто Бордіні, ігровий дизайнер і аспірант кафедри взаємодії людини з комп'ютером в Університеті Оффенбурга, який з 2017 року досліджує дизайн цифрового втручання для вирішення проблеми самотності у молоді [8].

Основна мета додатка Noneliness — зменшити рівень самотності серед студентів університету шляхом створення соціальних можливостей за допомогою геймифікованої системи квестів.

Головний екран застосунку представлено на рисунку 1.1, видає зведену інформацію про події, квести, ваші бали подяки та рівень, які набуваються через активність у додатку. Щоб приєднатися до події або квесту, потрібно лише приєднатися до чату, натиснувши відповідну кнопку.

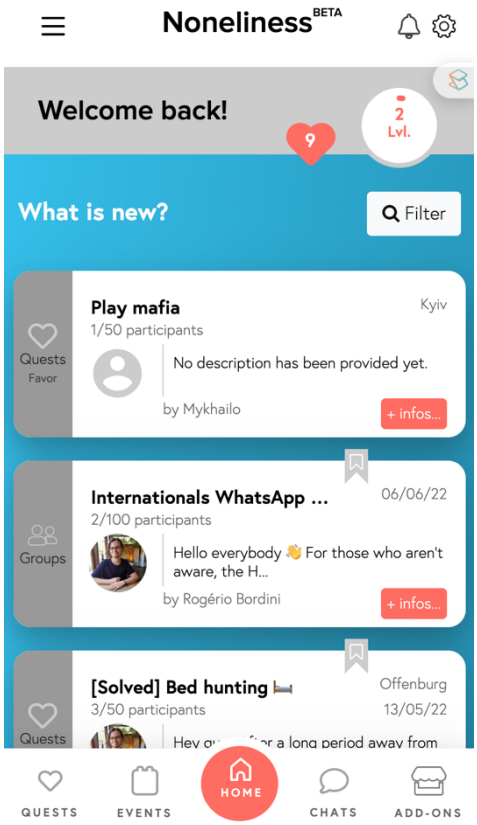


Рис. 1.1 Скріншот головного екрану web-застосунку Noneliness

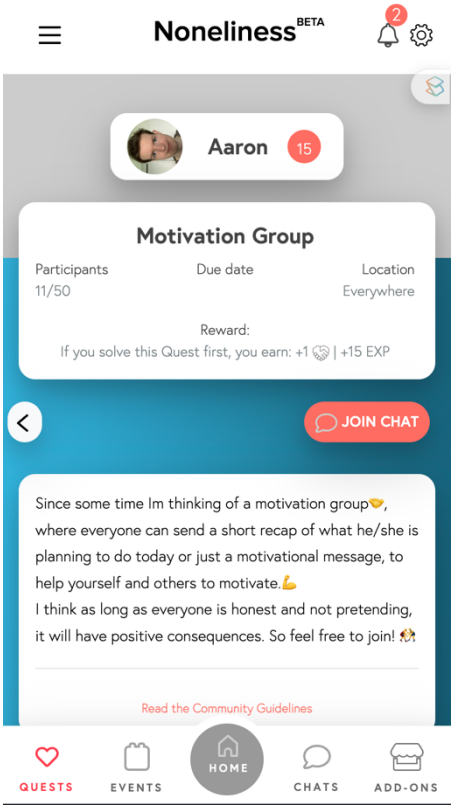


Рис. 1.2 Скріншот сторінки квесту Noneliness

«Квести» розроблено, щоб заохотити членів спільноти допомагати один одному, сприяти почуттю соціального зв'язку та винагороджувати корисну поведінку очками подяки та очками досвіду. Квести можуть мати час виконання та місце виконання, на рисунку 1.2.

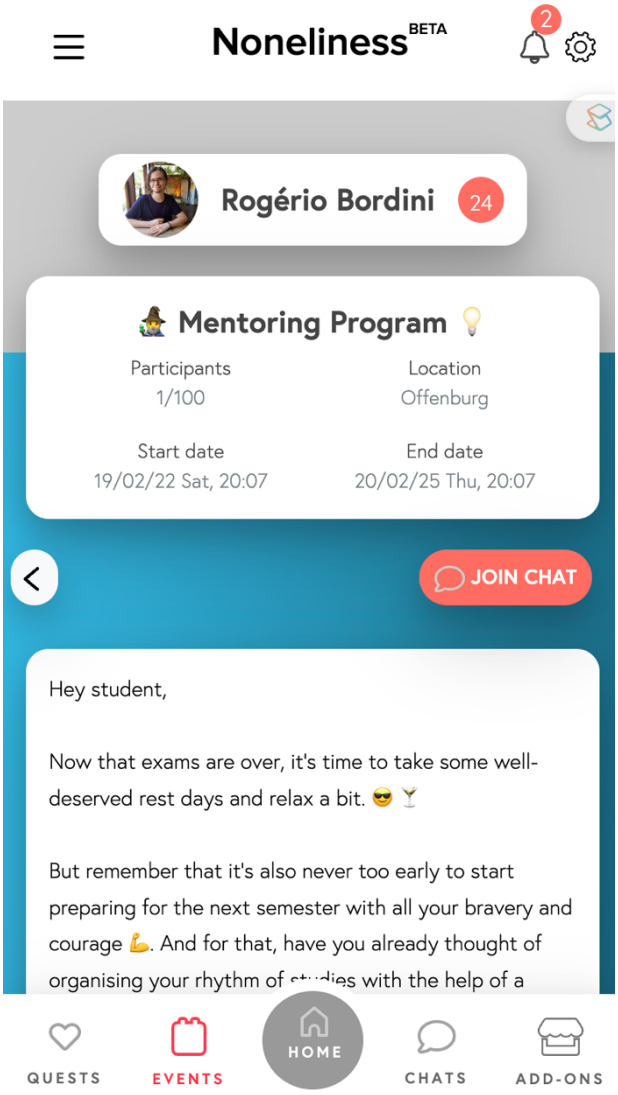


Рис. 1.3 Скріншот сторінки квесту Noneliness

«Події» ще один інструмент, щоб заохотити користувачам приєднуватися або створювати події в певний час і в певних місцях, як-от художні виставки чи зустрічі, допомагаючи людям збиратися та ділитися досвідом, представлено на рисунку 1.3. Це також може дозволяє співробітникам робити внесок у

соціальну та інституційну діяльність, сприяючи почуттю спільності та участі в університеті.

Прив’язаність до контексту університету є суттєвим недоліком застосунку і значно обмежує аудиторію.

Інтерфейс додатку є досить застарілим, що може бути проблемою для вимогливих користувачів.

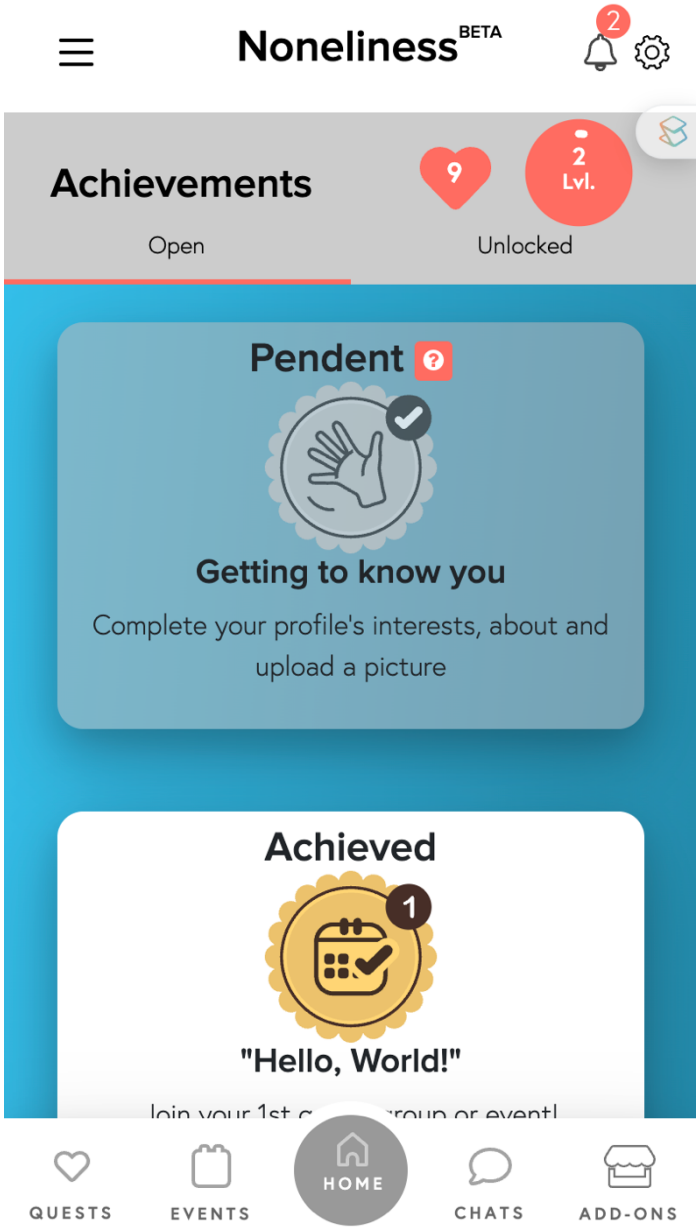


Рис. 1.4 Скріншот сторінки досягнень Noneliness

Перевагою Noneliness є чимало елементів гемифікації для заохочення взаємодії із застосунком, а саме:

- Досягнення, що предаствлені на рисунку 1.4, використовуються, щоб допомогти користувачам краще зрозуміти функції програми та заохотити взаємодію між ними шляхом виконання певних завдань або досягнення етапів. Ці досягнення є своєрідними орієнтирами, що мотивують користувачів до активної участі в різних аспектах програми. Наприклад, користувачі можуть отримувати досягнення за виконання першого завдання, участь у певній кількості соціальних взаємодій або досягнення визначеного рівня активності протягом певного періоду. Ця система допомагає створити відчуття прогресу та досягнень, що сприяє підвищенню мотивації користувачів і стимулює їх до подальшої взаємодії з програмою.
- Бали подяки: користувачі заробляють або дають бали, щоб висловити свою вдячність за отриману підтримку, які можна використовувати для придбання віртуальних предметів, заохочуючи позитивне та сприятливе середовище спільноти. Ця система балів подяки стимулює користувачів до взаємної підтримки та допомоги, формуючи культуру вдячності та взаєморозуміння. Бали подяки можна накопичувати, а потім обмінювати на різні віртуальні предмети, що робить процес більш захоплюючим. Така механіка сприяє активнішій взаємодії користувачів, оскільки вони відчувають, що їхні зусилля та добрі вчинки цінуються.
- Бали досвіду: у міру того, як користувачі більше взаємодіють із програмою, вони отримують бали досвіду за кожне надіслане повідомлення та бали подяки, які пожертвувано, підвищуючи рівень і демонструючи свою залученість і внесок у спільноту. Система балів досвіду нагороджує користувачів за активну участь у програмі, створюючи відчуття розвитку та особистого зростання. Бали досвіду можуть використовуватися для підвищення рівня користувача, що відкриває доступ до нових функцій або можливостей у програмі. Це не тільки підтримує мотивацію користувачів, але й допомагає їм бачити свій

прогрес, що додатково сприяє їхньому залученню та постійній активності.

1.2.2 Happify

Happify це застосунок, розроблений для того, щоб допомогти людям покращити своє психічне здоров'я, навчаючи їх навичкам боротьби з почуттям самотності та сприяння благополуччю, використовуючи дії та ігри, засновані на психологічних дослідженнях, для виховання позитивних звичок і думок [9].

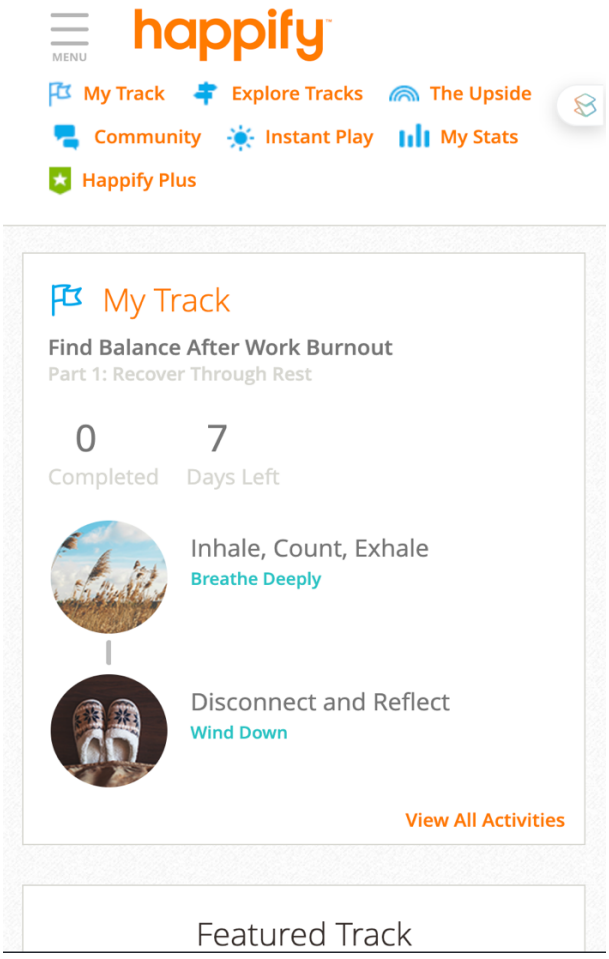


Рис. 1.5 Скріншот головної сторінки Happify

Додаток Happify використовує концепцію «доріжок», представлено на рисунку 1.5, як метод, спрямований на досягнення конкретних цілей і вирішення психологічних проблем користувачів. Кожна «доріжка» — це

комплекс заходів, ігор, вправ і завдань, розроблених з урахуванням сучасних психологічних методик. Ці треки зосереджені на розвитку конкретних навичок або вирішенні психологічних аспектів, таких як стрес, тривога, самотність тощо.

Кожна доріжка в Happify складається з кількох рівнів, кожен з яких представляє певну послідовність завдань, спрямованих на поступове досягнення поставлених цілей. Наприклад, для зменшення стресу користувачі можуть пройти трек, який включає медитації, дихальні вправи та когнітивно-поведінкові техніки. Для поліпшення емоційного стану доріжка може включати вправи на вдячність та активності, що сприяють розвитку емпатії.

Така комплексна система є неабиякою перевагою та недоліком одночасно. Вона дозволяє користувачам систематизувати свої психологічні цілі, розбиваючи їх на послідовні кроки, і забезпечує підтримку через інтерактивні ігри та завдання, пропонуючи користувачам цілеспрямований підхід для вирішення їхніх проблем, як-от самотність. Крім того, користувачі можуть обирати ті доріжки, які найбільше відповідають їхнім поточним потребам, що робить процес особистісного розвитку більш структурованим і зрозумілим.

Проте складність цієї системи вимагає самодисципліни від користувачів, оскільки кожен рівень потребує виконання певних завдань у визначений час. Відсутність індивідуального підходу може бути перешкодою для деяких користувачів, адже не всі психологічні методики підходять кожній людині. Деякі користувачі можуть відчувати, що завдання занадто загальні або не відповідають їхнім конкретним потребам, що може призвести до зниження мотивації та інтересу до використання додатку.

Загалом, концепція «доріжок» у Happify пропонує структурований і науково обґрунтований підхід до покращення психологічного благополуччя, але вимагає від користувачів самодисципліни та готовності до систематичної роботи над собою.

Рис. 1.6 Базові навички Happify

Застосунок виділяє шість базових навичок, які необхідні людини, представлено на рисунку 1.6:

- смакування (Savor) - це здатність зосереджуватися на позитивних сторонах життя та цінувати їх. Ця навичка може допомогти забути про самотність, заохочуючи вдячне та уважне ставлення до щоденних подій. Користувачі вчаться насолоджуватися моментами радості, фіксуватися на приємних подіях і відчуттях, що допомагає створювати позитивний емоційний фон і знижує рівень стресу;
- вдячність (Thank) - це навичка вираження вдячності, яка допомагає вам розпізнавати та цінувати те, що ви маєте. Вона сприяє задоволенню та зменшує відчуття відчуження, висвітлюючи зв’язки та позитивні аспекти життя. Практика вдячності допомагає формувати позитивне мислення,

зосереджуючись на добрих моментах і людях, що оточують вас, що може суттєво покращити загальний емоційний стан;

- дарування (Give) - це акт доброти та щедрості до інших, який може зміцнити соціальні зв'язки, зробити людей більш зв'язаними та менш самотніми, сприяючи почуттю спільності. Допомагаючи іншим, користувачі відчувають себе корисними та потрібними, що підвищує їх самооцінку та зміцнює соціальні взаємозв'язки;
- емпатія (Empathize) - це розуміння та розділення почуттів інших, що допомагає будувати глибокі, значущі стосунки та зменшує відчуття самотності завдяки співчуттю та зв'язку з іншими. Розвиваючи емпатію, люди стають більш чутливими до потреб і емоцій оточуючих, що сприяє кращому взаєморозумінню та підтримці в стосунках;
- оживлення (Revive) - передбачає використання різноманітних стратегій, таких як уважність і вдячність, які допомагають зменшити почуття самотності та заохочують до активної участі в практиках, які покращують психічне здоров'я. Ця навичка включає методи релаксації, відновлення сил та підтримку емоційного балансу, що є важливими для збереження психологічного благополуччя;

Щоб заробити бали у застосунку та отримувати доступ до нових рівнів, потрібно брати участь у різноманітних заходах, як-от читання коротких статей, перегляд відео про щастя, гра в міні-ігри на телефоні та застосування навичок у реальному житті, які зароблять вам бали, щоб покращити певну статистику. Така геймифікація є хорошою перевагою для більшої залученості аудиторії, представлено на рисунку 1.7.

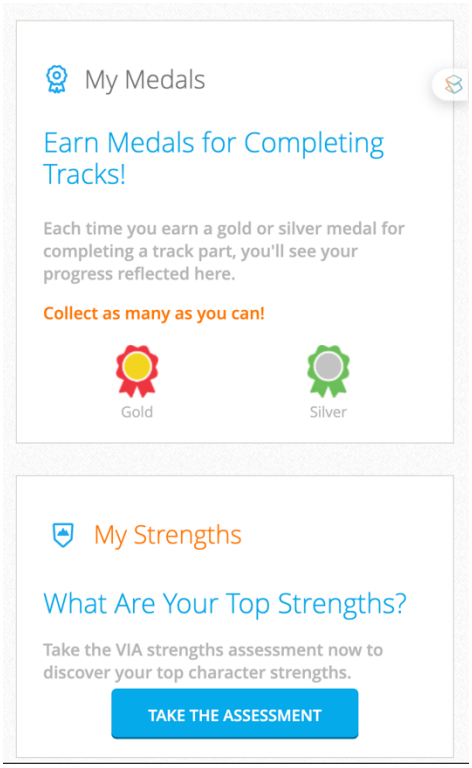


Рис. 1.7 Елементи геймифікації Happify

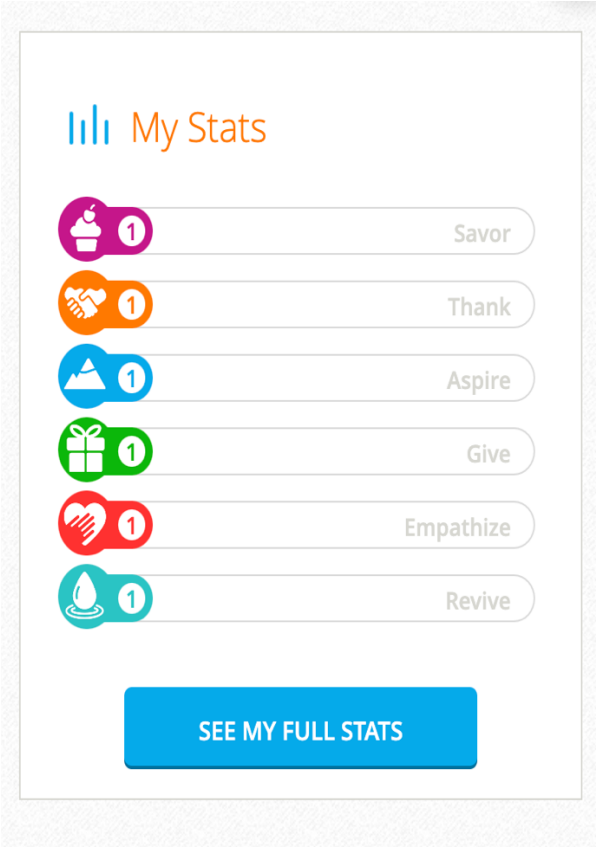


Рис. 1.8 Статистика Happify

Happify активно використовує статистику для відстеження прогресу та взаємодії користувачів з додатком, представлено на рисунку 1.8. Користувачам надаються індивідуальні статистичні дані, що відображають їх активність, досягнення та емоційний стан протягом часу. Додаток збирає дані про кількість виконаних завдань, завершених доріжок та здобутих досягнень, що допомагає користувачам бачити свій прогрес і мотивує їх продовжувати користуватися додатком. Це також дозволяє користувачам аналізувати, як різні вправи та активності впливають на їхній емоційний стан і добробут.

Загальна статистика, зібрана від усіх користувачів, дозволяє команді Happify аналізувати ефективність різних методик і вправ, постійно вдосконалюючи контент і адаптуючи його до потреб користувачів. Це забезпечує безперервне поліпшення додатка і підвищення його ефективності в досягненні головної мети — покращення емоційного здоров'я і добробуту користувачів. Завдяки детальній статистиці та персоналізованому підходу, Happify створює ефективну програму для покращення емоційного стану кожного користувача.

Додаток доступний на платформах Web, Android та iOS, що є перевагою для залучення якомога більшої аудиторії.

1.2.3 Playersgetready

Playersgetready - це застосунок, що позиціонує себе як інструмент для усвідомленого пізнання себе та людей навколо. Основна мета цього додатку — сприяти глибшому розумінню себе та оточуючих через інтерактивні запитання та активності. Застосунок надає користувачам можливість поглиблювати взаємодію з іншими через серії питань, які стимулюють саморефлексію та відкриті розмови на різні теми [9].

Рис. 1.9 Скріншот головного екрану web-застосунку Playersgetready

Застосунок пропонує користувачам різноманітні набори питань, представлено на рисунку 1.9, що охоплюють такі аспекти життя, як любов, дружба, родина та нові знайомства. Кожен набір питань розроблений таким чином, щоб допомогти користувачам краще розуміти свої власні почуття, а також почуття та думки інших. Це сприяє створенню більш глибоких і значущих зв’язків між людьми [10].

Рис. 1.10 Скріншот гри Playersgetready

Застосунок у формі гри містить набір питань, які можна переглянути за допомогою свайпу або натиснувши на стрілку, щоб перейти до наступного запитання, представлено на рисунку 1.10. Крім того, є можливість переглянути всі запитання, натиснувши кнопку «Картотека». Кожне запитання для одного гравця, і хоча функціональність програми проста, вона значно полегшує спілкування.

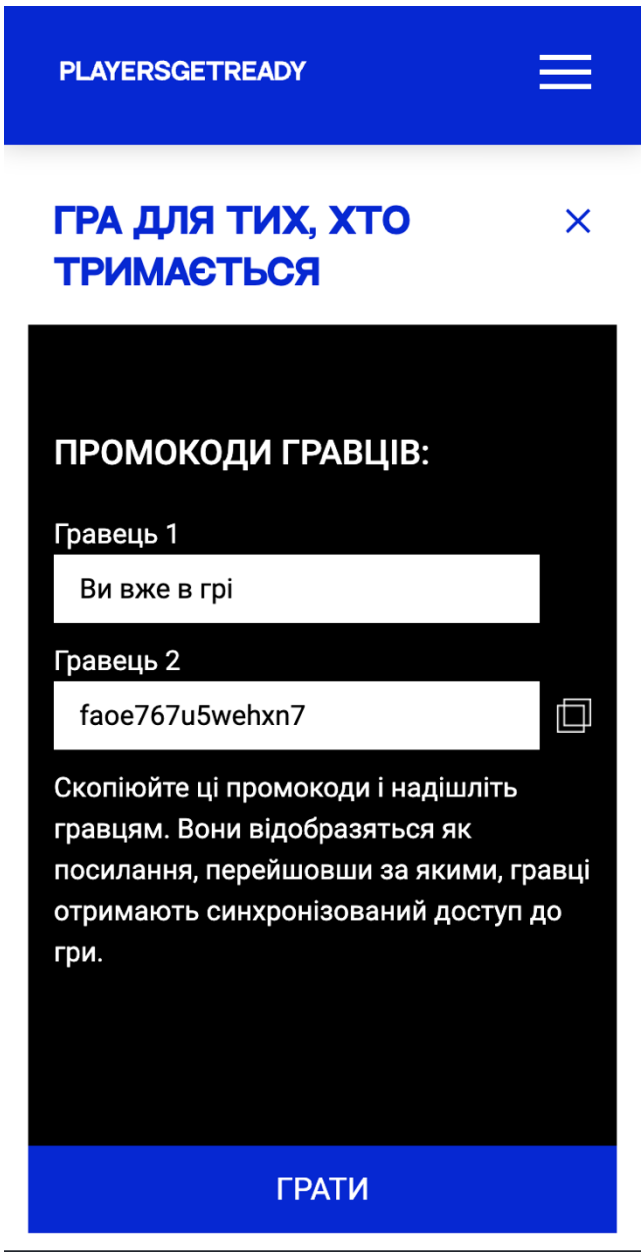


Рис. 1.11 Скріншот налаштування багатокористувацької гри Playersgetready

Також доступна багатокористувацька гра, що стає доступною завдяки можливості отримати та надати доступ до гри іншим гравцям, що грають на відстані. Для цього потрібно створити промокоди, які дозволять гравцям приєднатися до гри. Необхідно вибрати бажану кількість учасників на відстані та отримати промокод, представлено на рисунку 1.11.

Серед переваг можна виділити широку аудиторію, адже додатком можна користуватися у різних життєвих ситуаціях. Також можливість

багатокористувацької гри є великою перевагою.

До недоліків можна віднести відсутність елементів геймифікації, що обмежує ігровий досвід користувачів. Це може знижувати мотивацію для тривалого використання застосунку, оскільки відсутні додаткові стимули, такі як досягнення або бали, які могли б заохочувати користувачів до активнішої участі.

1.2.4 Порівняння наявних цифрових рішень

Таблиця 1.1

Зведені результати аналізу характеристик застосунків для соціальної взаємодії та зменшення самотності

Показник	**Nonelines**	**Happify**	**Playersgetread**
Елемент геймифікації “досягнення”	+	-	-
Елемент геймифікації “бали”	+	+	-
Матеріали для розвитку комунікації	-	+	+
Можливість створювати власний контент	+	-	-
Можливість ділитися власним контентом	+	-	-
Взаємодія між користувачами	+	-	+

Проаналізувавши існуючі рішення для соціальної взаємодії та зменшення самотності (таблиця 1.1), ми визначили основні переваги та недоліки. На основі цих даних, можна визначити як може виглядати оптимальний застосунок для

вирішення цієї проблеми.

Хоча Noneliness має досить застарілий інтерфейс та обмежену аудиторію, це додаток чудово залучає користувачів за допомогою багатьох елементів геміфікації. Для нашого застосунку ми можемо взяти систему “балів”, які прив’язуються до профілю користувача.

Happify - потужний інструмент, однак теж має застарілий інтерфейс та вимагає чималої дисципліни від користувачів. З цього додатку можна виділити систему досягнень, які можуть допомагати залучати користувачів.

Playersgetready допомагає користувачам краще розуміти себе та сприяє активному спілкуванню в різних життєвих ситуаціях. Цей застосунок має широку аудиторію та може бути основою для універсального застосунку, який розробляється. Серед ключових мінусів є відсутність елементів геймифікації, відсутність персоналізації профілю та застарілий інтерфейс.

Як висновок, можна визначити функції, які необхідні для застосунку для поліпшення комунікації та вирішення самотності:

- можливість карткової гри за темами;
- авторизація користувачів;
- можливість створювати власні теми з питаннями;
- можливість ділитися створеними темами між користувачами;
- можливість здобування досягнень для заохочення використати весь функціонал застосунку;
- можливість накопичування балів для більшого залучення користувачів;
- кросбраузерність за стандартом Baseline;
- відповідність стандартам Apple UX/UI;
- розробка під мобільні пристрої;

На основі цього, було розроблено діаграму прецедентів, представлено на рисунку 1.12.

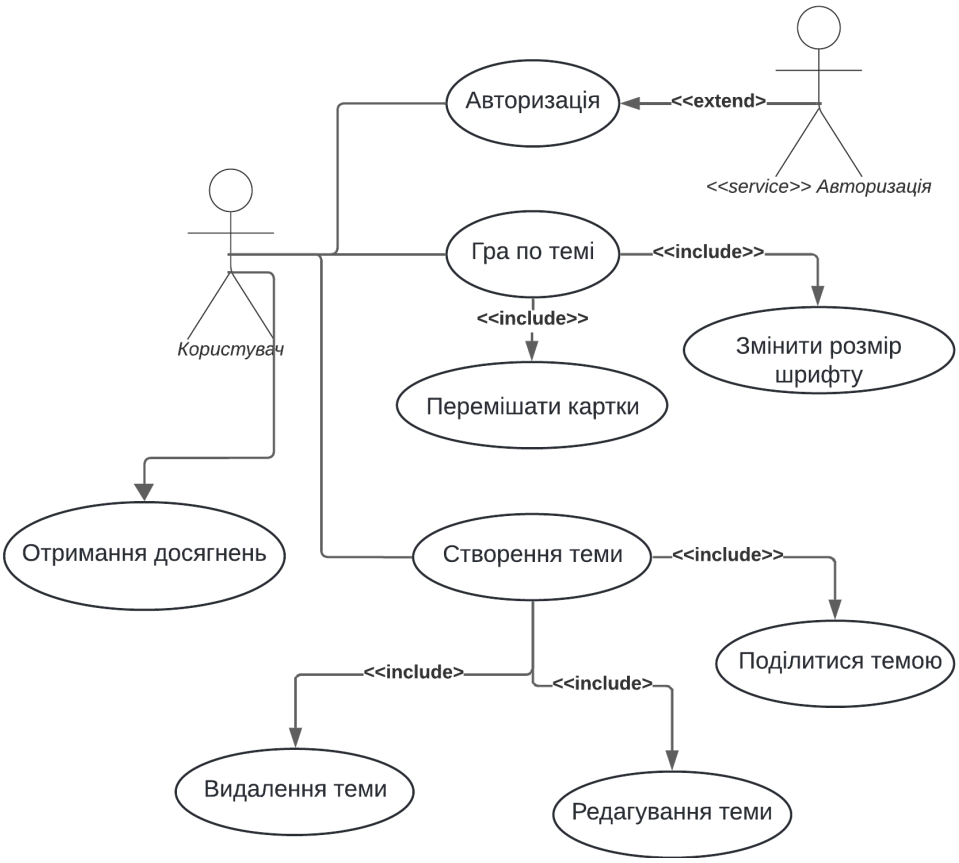


Рис. 1.12 Діаграма прецедентів для розроблюваного застосунку

2. РОЗРОБКА ВИМОГ ДО WEB-ЗАСТОСУНКУ

2.1. Технічне завдання

Основною концепцією застосунку є створення та використання наборів тем з питаннями, що покращують якість спілкування. Користувачі можуть грати вдвох або більше осіб, відповідаючи на питання, які стосуються різних тем. Це допомагає поглиблювати спілкування та зменшує емоційну самотність, оскільки люди більше дізнаються один про одного та знаходять спільні інтереси.

Важливою частиною застосунку є елементи геймифікації, що сприяють залученню користувачів та підвищенню їхньої активності. Система досягнень на накопичувальних балів мотивуватиме регулярно заходити у застосунок і сприятиме регулярному спілкуванню.

Для ширшого спектру можливостей та більшої кількості варіантів використання, застосунок повинен мати можливість зберігати власні теми та ділитися ними. Це дозволить користувачам створювати власні унікальні теми, що відображають їхні вподобання, та ділитися ними з друзями. Така функціональність збільшує кількість можливих взаємодій між користувачами.

Застосунок повинен також мати зручний та інтуїтивний інтерфейс, щоб орієнтуватися серед доступних функцій. Відповідність стандартам Apple UI/UX.

2.2 Функціональні вимоги

2.2.1 Можливість карткової гри за темами

Додаток повинен містити теми, які являють собою списки питань, створені для покращення комунікації та соціальної взаємодії. Кожна тема

повинна мати іконку для швидшого візуального сприйняття, а також опис, що допомагає користувачам зрозуміти, для кого ця тема призначена. Відкриваючи тему, користувачі спершу бачать іконку, опис теми, кількість питань у ній та кнопку “Розпочати гру”.

На сторінці гри питання повинні відображатися по одному у форматі карток, між якими можна перемикатися стрілкою або проведенням пальцем по картці. Натиснувши на номер питання, повинна бути можливість вибрати конкретне питання за номером. Це дозволяє користувачам більш гнучко керувати процесом гри, адаптуючи його під свої потреби та уподобання.

2.2.2 Авторизація користувачів

Необхідно реалізувати авторизацію користувачів за допомогою Google, що забезпечить швидкий та безпечний вхід до застосунку. Ця функція повинна дозволяти користувачам приєднувати інформацію до свого облікового запису у базі даних, що включає їхні налаштування, досягнення та прогрес у грі. Це забезпечить персоналізований досвід використання

2.2.3 Можливість створювати власні теми з питаннями

Користувачі повинні мати можливість створювати власні теми з питаннями, що буде реалізовано через додавання відповідної таблиці у базу даних. Ця функція дозволить користувачам створювати унікальні теми, що відображають їхні інтереси та потреби, а також ділитися цими темами з іншими користувачами. Це сприятиме розширенню контенту застосунку та залученню більшої кількості користувачів.

2.2.4 Можливість ділитися створеними темами між користувачами

Додаток повинен підтримувати зв’язок багато-до-багатьох, щоб користувачі могли ділитися створеними темами з іншими користувачами. Це буде реалізовано за допомогою можливості поділитися кодом теми. Інший

авторизований користувач повинен мати кнопку “Додати тему за кодом”, після чого тема з’являється у його списку серед системних тем. Така функція дозволяє розширювати можливості взаємодії між користувачами та сприяє більш активному використанню застосунку.

2.2.5 Можливість здобування досягнень для заохочення використати весь функціонал застосунку

Необхідно додати розділ з досягненнями, де буде чіткий опис, як їх здобути. Після здобування досягнення, це має записуватися у базу даних, а користувачеві повинно показуватися діалогове вікно із сповіщенням. Така система досягнень заохочує користувачів використовувати весь функціонал застосунку, підтримуючи їхню мотивацію та інтерес до подальшого використання.

2.2.6 Можливість накопичування балів для більшого залучення користувачів

Накопичувальні бали повинні нараховуватися після здобування досягнень, причому кількість балів визначається для кожного досягнення окремо. Крім того, бали також повинні нараховуватися після кожного п’ятого питання у темі, яку користувач проходить. Це створює додатковий стимул для користувачів брати участь у грі та підвищує рівень їхньої залученості.

2.3 Нефункціональні вимоги

2.3.1 Кросбраузерність за стандартом Baseline

Web-застосунок повинен однаково відображатися на найпопулярніших браузерах відповідно до стандарту Baseline [11]. Це забезпечить коректну роботу та однаковий користувацький досвід незалежно від вибору браузера користувачем, підвищуючи доступність та зручність застосунку.

2.3.2 Відповідність стандартам Apple UX/UI

Застосунок повинен реалізовувати інтерфейс, оптимізований для мобільних пристроїв, і відповідати стандартам Apple UI/UX [12]. Кнопки повинні бути розміром не менше 44 на 44 пікселів для зручності натискання. Важливі кнопки повинні розташовуватися у нижній частині екрану для легкої доступності. Також необхідно забезпечити можливість регулювання розміру тексту для питань, що підвищить зручність користування для людей з вадами зору.

2.3.3 Розробка під мобільні пристрої

Інтерфейс застосунку повинен бути пристосований для використання на мобільних пристроях, забезпечуючи зручний доступ та використання всіх функцій на різних екранах.

2.4 Сховище даних застосунку

На основі функціональних та нефункціональних вимог розроблюваного застосунку, необхідно визначити яку базу даних нам необхідно використовувати для збереження даних. Виділяють основні два типи баз даних: SQL; NoSQL;

2.4.1 SQL база даних

SQL база даних реалізовує реляційну модель даних та керується за допомогою мови структурних запитів SQL для обробки даних. Дані організовуються у вигляді таблиць, де рядки представляють записи (користувач 1, користувач 2), а стопці - зазначені поля (ім’я, прізвище). Це забезпечує високий рівень структуризації та цілісність даних.

Основні характеристики SQL баз даних включають:

– Структурованість - дані зберігаються у таблицях з чітко визначеними

схемами, що забезпечує зрозумілість і зручність у роботі з даними;

- Цілісність - кожен запис у таблиці має унікальне поле або комбінацію полів, що гарантує унікальність кожного запису;
- Зв’язки між таблицями встановлюються за допомогою зовнішніх ключів, що дозволяє зберігати та підтримувати відношення між даними в різних таблицях, наприклад, зв’язок між користувачами і їхніми темами;

2.4.2 NoSQL база даних

NoSQL бази даних пропонують різні моделі зберігання та обробки даних, які допомагають при обробці великих об’ємів інформації. Вони також краще масштабуються, можуть забезпечувати швидший доступ до даних та мають більшу гнучкість за рахунок відсутності схем.

Основні характеристики NoSQL баз даних включають:

- Гнучкість - відсутність жорстких схем дозволяє швидко змінювати структуру даних, що може бути корисно в динамічних середовищах.
- Масштабованість - база даних NoSQL фактично краще масштабується горизонтально, що означає можливість додавати нові сервери для обробки збільшених обсягів даних.
- Різні моделі зберігання - існують різні типи баз даних NoSQL, включаючи орієнтовані на документи, бази даних із графіками, бази даних зі стрілками та ключ-значення, кожна з яких підходить для певних типів завдань.

2.4.3 База даних розроблюваного застосунку

Потреби нашого додатку може цілком задовільнити SQL база даних з кількома таблицями для збереження тем, користувачів та зв’язку багато-до-багатьох між ними. Детальніше про вибір конкретної бази даних та її структури у розділі 3.3.

2.5 Прототип інтерфейсу користувача

На основі вимог та структури застосунку було розроблено перші протопити, які будуть взяти за основу для подальшого проектування, представлено на рисунку 2.1.

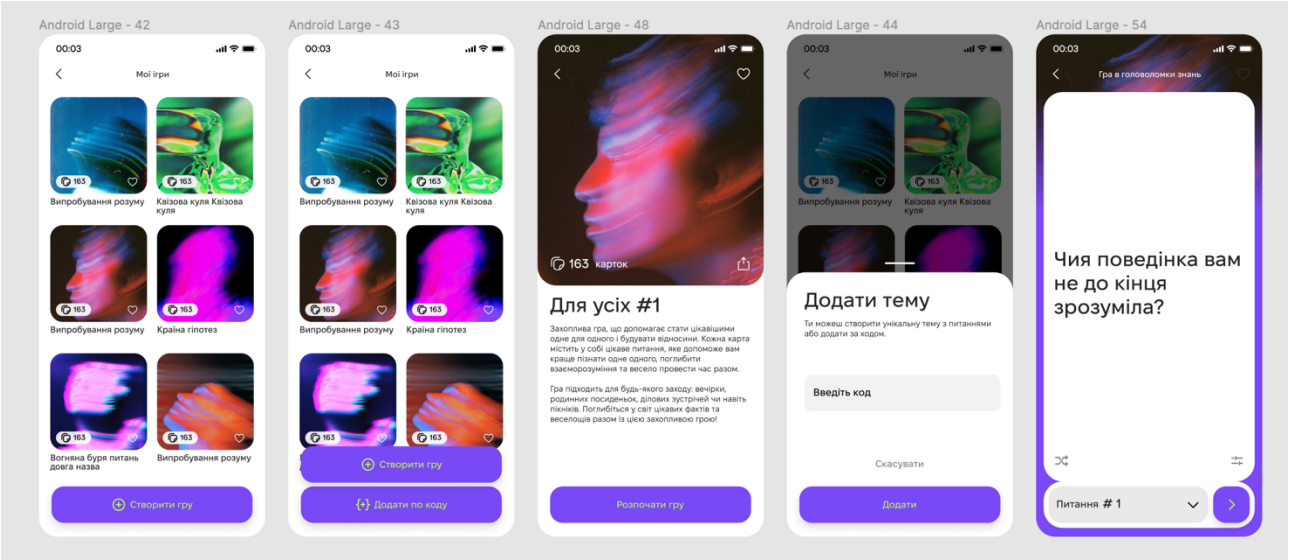


Рис. 2.1 Дизайн прототипу для розроблюваного застосунку

3. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Засоби програмної реалізації

3.1.1 Загальні засоби реалізації web-застосунків

Розробка web-застосунків включає в використання технологій для реалізації, які охоплюють розробку клієнтської та серверної частини застосунку.

Клієнтська частина (фронтенд) - включає в себе реалізацію інтерфейсу застосунку. Основними технологіями для цього є HTML для розмітки структури застосунку, CSS для оформлення та забезпечення адаптивного дизайну, JavaScript для створення інтерактивних елементів, логотипи представлені на рисунку 3.1. Зокрема існує чимало більш високорівневих технологій, які допомагають розробляти web-застосунки ефективніше та значно структурованіше. Про них буде детальніше згадано у наступних розділах.

Рис. 3.1 Логотипи HTML, CSS, JavaScript - базові технології для розробки web-застосунків

Серверна частина (бекенд) - відповідає за обробку запитів від фронтенду, взаємодію з базами даних та виконання складних обчислень. Основними завданнями бекенду є обробка запитів та відповідей, взаємодія з базами даних і виконання бізнес-логіки. Сервер отримує запити від клієнтів, обробляє їх і повертає відповідні відповіді, забезпечуючи функціональність і взаємодію між різними частинами застосунку. Взаємодія з базами даних дозволяє зберігати, оновлювати, видаляти та отримувати дані, необхідні для роботи застосунку. Виконання бізнес-логіки включає валідацію даних, обробку транзакцій та виконання складних алгоритмів.

Розробка web-застосунків вимагає злагодженої роботи клієнтської та серверної частин, які взаємодіють між собою через HTTP-запити та відповіді, забезпечуючи інтерактивність та функціональність кінцевого продукту.

3.1.2 Середовище розробки Visual Studio Code

Visual Studio Code - безкоштовний редактор коду з відкритим вихідним кодом, який розробляється компанією Microsoft. Завдяки систему розширень, редактор може підтримувати широкий спектр мов програмування та синтаксисів, серед яких JavaScript, Python, Java, C++, C#, PHP, HTML, CSS. Можливість легко додати підтримку цих мов, робить цей інструмент універсальним інструментів серед розробників, представлено на рисунку 3.2.

Редактор також підтримує інтеграцію з системою контролю версій Git, що дозволяє розробникам легко керувати версіями прямо з нього. Підтримуються усі необхідні функції: коміти, пуші, пули, перегляд змін, тощо, представлено на рисунку 3.3.

Таким чинок цей редактор коду є оптимальним рішення для розробки застосунку, у якому є клієнтська та серверна частина. Оскільки це вимагає використання широкого спектру технологій, які повинні мати синтаксичну підсвітку.

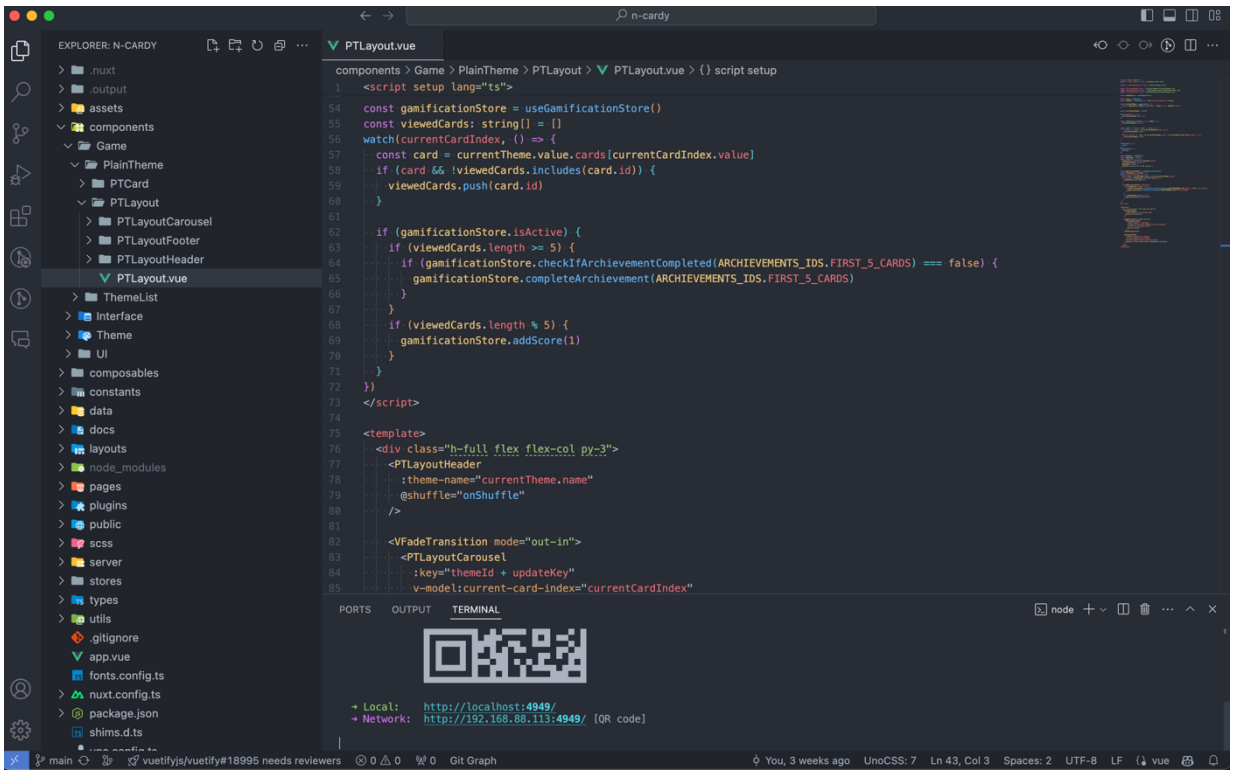


Рис. 3.2 Інтерфейс Visual Studio Code

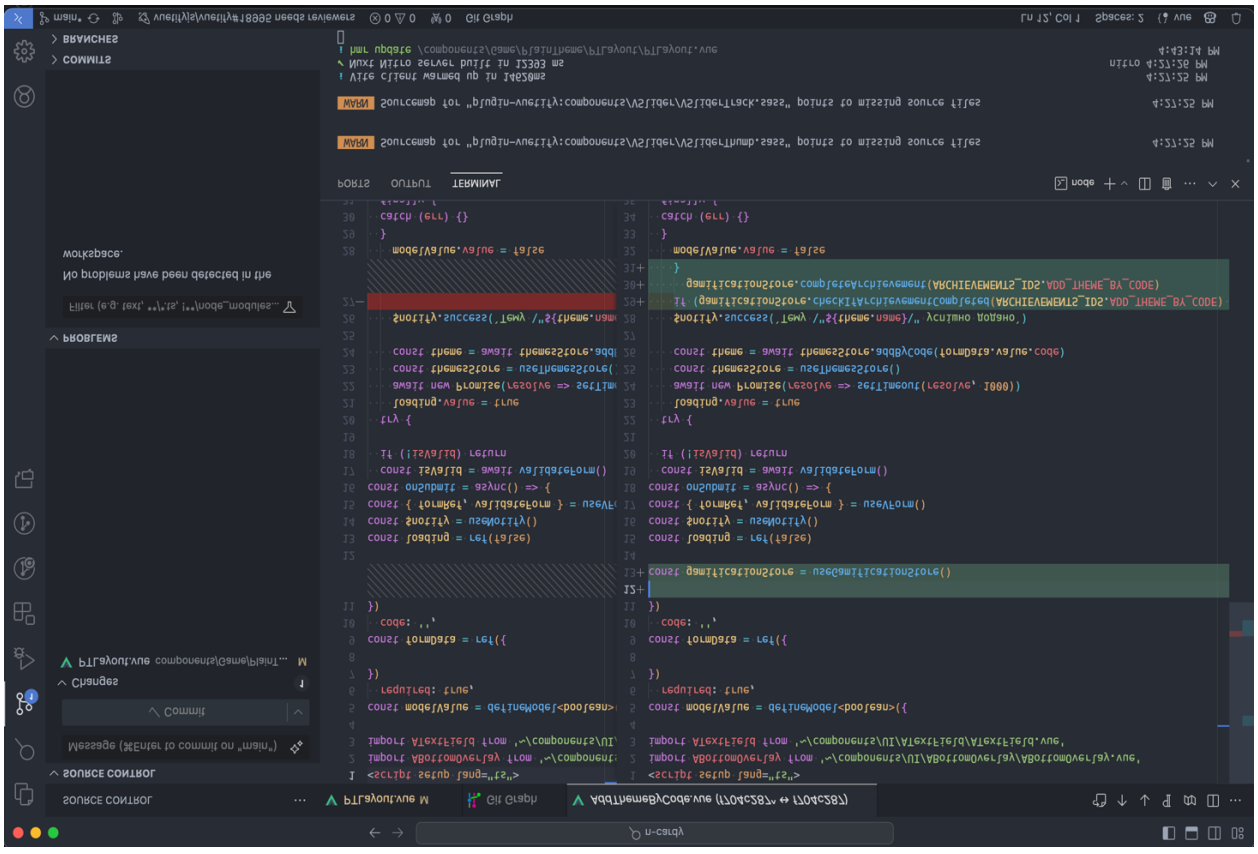


Рис. 3.3 – Приклад використання GIT у Visual Studio Code

3.1.3 Фреймворк Vue

Для рішення комплексних задач варто обрати фреймворк з компонентним підходом до розробки. Тож для використовуватиметься фреймворк Vue, що є одним з трьох провідних фреймворків, які використовуються для прогресивних web-застосунків. Він чудово поєднує прототипу використання, чудові інструменти для розробки та покриває увесь необхідний функціонал з чудовою ефективністю. Він реалізовує патерн програмування "Model-View-ViewModel" (MVVM) для зручного розділення моделі “Model” (даних та бізнес логіки) та виду “View” (візуального відображення інтерфейсу користувача), а “ViewModel” є посередником, що забезпечує зв’язок між видом та моделлю, керуючи даними, станом та бізнес логікою. Це забезпечує читабельність коду та гнучкість у розробці

3.1.4 Pinia

Pinia - це найсучасніший інструмент керування станом у Vue.js, який забезпечує зручний та ефективний метод керування станом програми. Діючи як офіційний наступник Vuex, Pinia забезпечує вдосконалений API і кращу сумісність із новими функціями Vue 3. Головною перевагою Pinia є її простота та гнучкість: вона дозволяє розробникам легко створювати та керувати глобальним станом програми без складні налаштування. Завдяки зрозумілій та інтуїтивно зрозумілій структурі коду Pinia стає зручнішою у використанні для розробників різного рівня.

3.1.5 Мета-фреймворк Nuxt

Був використаний мета-фреймворк Nuxt, що є надбудовою для Vue для структуризації коду, представлено на рисунку 3.3, та можливості додати серверну частину застосунку, яка буде тісно пов’язана з інтерфейсом. Це оптимальне рішення для додатків з невеликою кількістю серверної логіки. Nuxt після збірки видає код для запуску NodeJS сервера на базі Nitro, який

автоматично може визначати середовище запуску на найпопулярніших рішень для деплою, таких як aws amplify, azure, cloudflare pages, netlify, stormkit, vercel, zeabur.

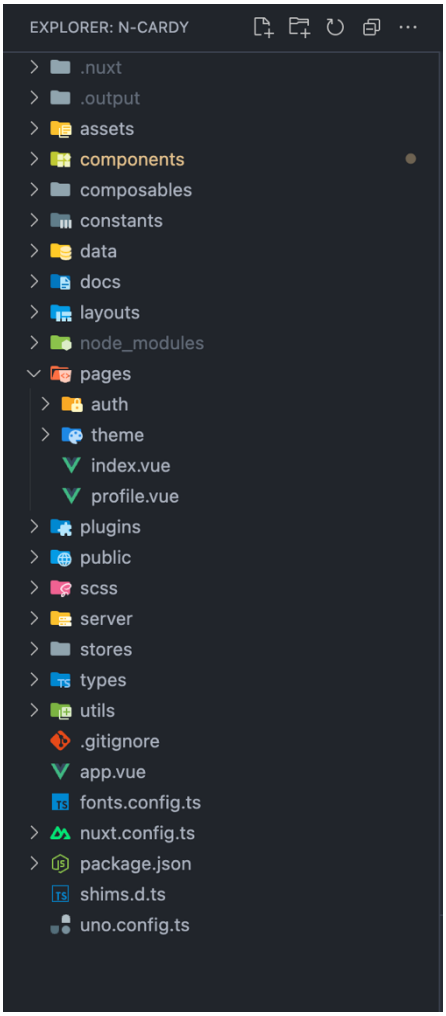


Рис. 3.3 Структура проекту з використанням Nuxt

3.1.6 TypeScript

TypeScript — це мова програмування, яка розширює можливості JavaScript, вводячи статичний тип. З його допомогою розробники можуть додавати типи до змінних, функцій та інших елементів коду, що покращує зрозумілість і придатність кодової бази. Статична типізація дозволяє виявляти помилки на етапі компіляції, а не під час виконання програми, що знижує ймовірність помилок у готовому продукті. Це особливо важливо для великих

проектів зі складною структурою.

TypeScript також сприяє кращій організації коду, дозволяючи визначати інтерфейси, класи та модулі. Це допомагає зберегти код чистим і полегшує його масштабування. Завдяки TypeScript створюються більш надійні та зрозумілі програми, що особливо корисно в командній роботі, коли над проектом працюють кілька розробників. TypeScript використовується тільки на етапі розробки, а потім компілюється в звичайний JavaScript, який підтримується всіма сучасними браузерами, представлено на рисунку 3.4. Це означає, що кінцевий продукт сумісний з будь-яким середовищем, де використовується JavaScript, забезпечуючи широку доступність і функціональність.

Рис. 3.4 Приклад як TypeScript код компілюється у JavaScript

3.1.7 Supabase та PostgreSQL

Supabase — це програмне забезпечення, яке спрощує будь-яку взаємодію з базою даних PostgreSQL, надаючи зручний інтерфейс і розширені можливості для розробників. Supabase дозволяє легко працювати з базами даних, виконувати запити, керувати даними та інтегрувати їх у веб-додатки. Крім того, Supabase надає широкий спектр додаткових функцій, які можуть бути корисними для нашого проекту, включаючи інструменти для авторизації

користувачів і SDK для використання сервісу з середовища JavaScript, зокрема через пакет npm «@supabase/supabase-js» .

Однією з важливих функцій Supabase є Supabase Auth. Цей інструмент надає можливість підключення популярних постачальників авторизації, таких як Google, Facebook, Discord та інші. Це дозволяє користувачам швидко та безпечно входити в систему без необхідності створювати нові логіни та паролі. На рисунку 3.5 представлено список можливих постачальників, які підтримуються Supabase Auth, що робить процес інтеграції авторизації максимально зручним і безпечним для кінцевих користувачів.

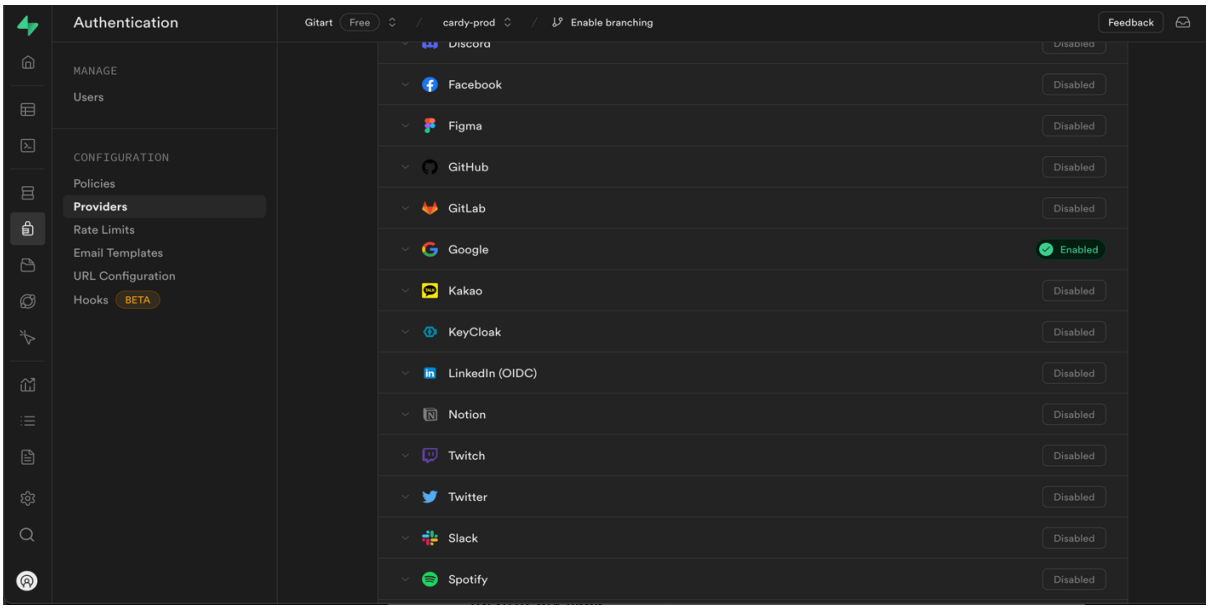


Рис. 3.5 Провайдери для авторизації Supabase Auth

Крім того, Supabase надає веб-інтерфейс, який дозволяє переглядати та керувати записами у вашій базі даних, представлено на рисунку 3.5. За допомогою цього веб-сайту ви можете переглядати всі записи в таблицях, керувати користувачами, вносити зміни в дані та структуру бази даних, а також виконувати інші адміністративні завдання. На рисунку 3.6 показано сторінку перегляду та редагування таблиці бази даних на веб-сайті Supabase, демонструючи зручність і функціональність цього інструменту для розробників і адміністраторів баз даних.

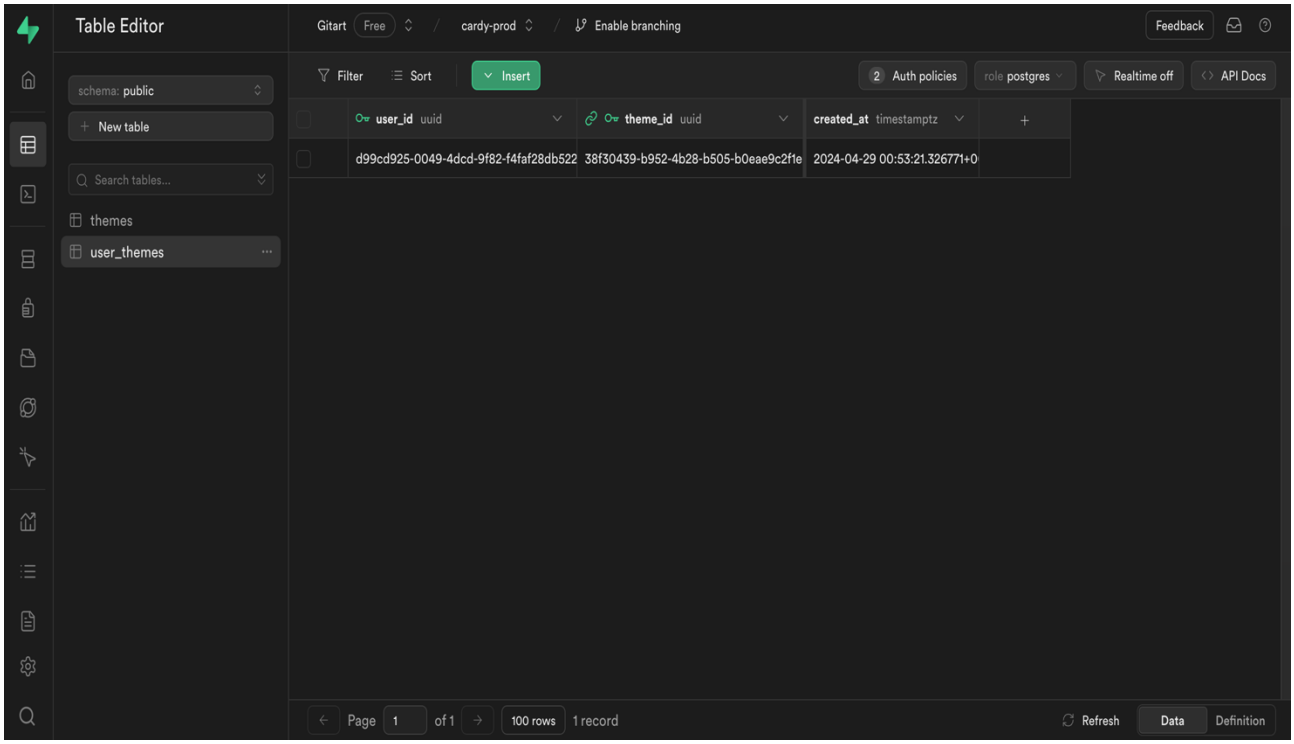


Рис. 3.6 Сторінка перегляду та редагування таблиці у базі даних на сайті Supabase

3.2 Архітектура

На основі обраних технологій була розроблена архітектура системи, яку можна побачити на рисунку 3.7. Ця архітектура включає як клієнтську, так і серверну частини, і детально описує, як вони взаємодіють один з одним, забезпечуючи повний цикл роботи web-застосунків. Завдяки використанню сучасних технологій архітектура забезпечує зручну та інтуїтивно зрозумілу взаємодію з користувачем, обробку запитів та відображення отриманих даних. Серверна частина обробляє запити від клієнта, виконуючи бізнес-логіку та взаємодіючи з базою даних та іншими зовнішніми службами, що забезпечує стабільність системи. Взаємодія між клієнтською та серверною частинами здійснюється через API, що забезпечує безперервний і надійний обмін даними, створюючи умови для ефективного функціонування веб-додатку в цілому.

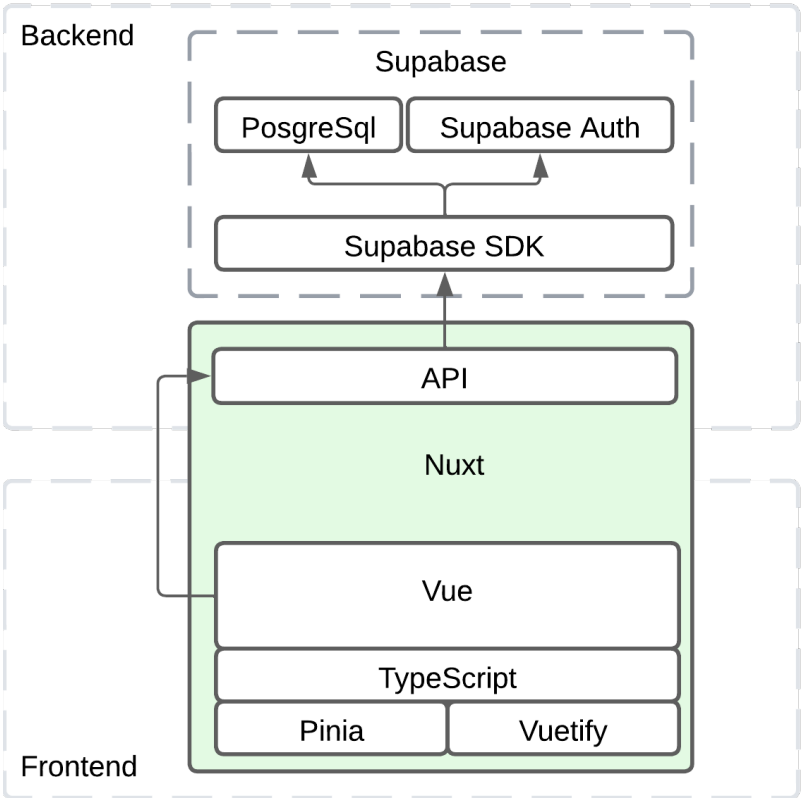


Рис. 3.7 Архітектура розроблюваного застосунку

3.2.1 Клієнтська частина

В основі клієнтської частини лежить фреймворк Vue, який використовується для створення інтерактивного та динамічного користувацького інтерфейсу. Vue забезпечує компонентний підхід до розробки, що дозволяє легко масштабувати та підтримувати код. Кожен компонент відповідає за окрему частину інтерфейсу, що сприяє кращій структуризації та повторному використанню коду.

3.2.2 Серверна частина

Серверна частина побудована за допомогою Supabase, яка забезпечує масштабовану базу даних та бекенд-сервіси. Supabase використовує PostgreSQL як основну базу даних, що забезпечує надійність та ефективність зберігання даних.

Основні функції Supabase включають:

– Авторизація та аутентифікація: Supabase надає інструменти для

безпечного управління користувачами, включаючи підтримку соціальних логінів (наприклад, Google, Facebook).

- Реалізація API: Supabase дозволяє легко створювати та керувати API для взаємодії з базою даних, забезпечуючи швидкий доступ до даних та їх обробку.

Інтеграційною ланкою є Nuxt, мета-фреймворк для Vue, який забезпечує додатковий функціонал для клієнтської частини та дозволяє реалізувати API для серверної частини. Nuxt полегшує розробку та розгортання застосунку.

3.2.3 Взаємодія компонентів

У цій архітектурі клієнтська та серверна частини тісно взаємодіють між собою:

- Користувацький інтерфейс: Реалізується за допомогою Vue та Vuetify, забезпечуючи інтерактивність та привабливий дизайн.
- Менеджмент стану: Використовується Pinia для централізованого управління даними, що гарантує їх узгодженість між різними компонентами.
- Запити до сервера: Здійснюються за допомогою API, створених на основі Supabase. Nuxt служить мостом між клієнтською та серверною частинами, забезпечуючи безшовну інтеграцію та ефективну взаємодію.

Ця архітектура забезпечує надійний, масштабований та легко підтримуваний web-застосунок, який відповідає всім сучасним вимогам до розробки програмного забезпечення.

3.3 Реалізація

3.3.1 Структура бази даних

Потреби нашого додатку може цілком задовільнити SQL база даних з кількома таблицями для збереження тем, користувачів та зв’язку багато-до-

багатьох між ними. На основі цього було розроблено діаграму класів для проектування схеми бази даних, представлено на рисунку 3.8, з наступними сутностями:

- user - відповідає за зберігання інформації про користувачів застосунку;
- themes - сутність зберігає інформацію про різні тематики, створені користувачами;
- user_themes - сутність забезпечує зв’язок між користувачами та тематиками. Вона дозволяє відстежувати, які користувачі мають доступ до яких тематик;

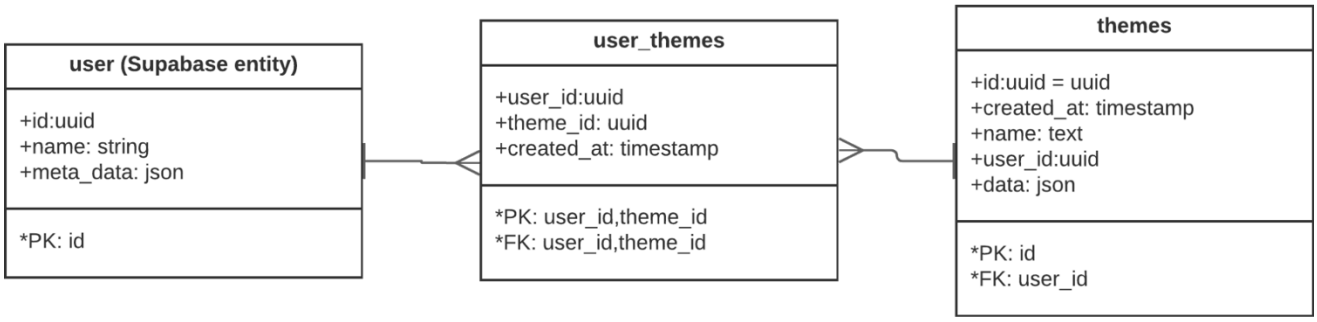
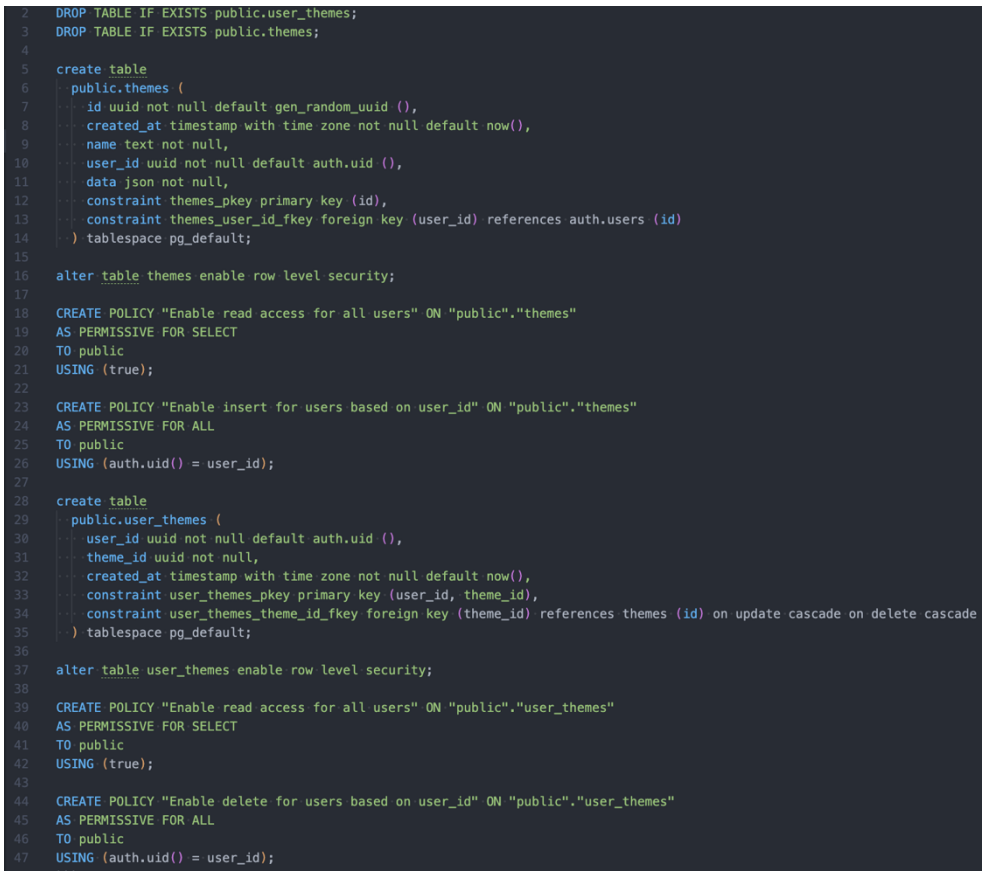


Рис. 3.8 Діаграма класів для розроблюваного застосунку

Завдяки сервісам Supabase, база користувачів є системною для цього сервісу. Тож було розроблено SQL код для створення схеми даних для *themes* та *user_themes* представлено на рисунку 3.9.



```sql
DROP TABLE IF EXISTS public.user_themes;
DROP TABLE IF EXISTS public.themes;

create table
  public.themes (
    id uuid not null default gen_random_uuid (),
    created_at timestamp with time zone not null default now(),
    name text not null,
    user_id uuid not null default auth.uid (),
    data json not null,
    constraint themes_pkey primary key (id),
    constraint themes_user_id_fkey foreign key (user_id) references auth.users (id)
  ) tablespace pg_default;

alter table themes enable row level security;

CREATE POLICY "Enable read access for all users" ON "public"."themes"
AS PERMISSIVE FOR SELECT
TO public
USING (true);

CREATE POLICY "Enable insert for users based on user_id" ON "public"."themes"
AS PERMISSIVE FOR ALL
TO public
USING (auth.uid() = user_id);

create table
  public.user_themes (
    user_id uuid not null default auth.uid (),
    theme_id uuid not null,
    created_at timestamp with time zone not null default now(),
    constraint user_themes_pkey primary key (user_id, theme_id),
    constraint user_themes_theme_id_fkey foreign key (theme_id) references themes (id) on update cascade on delete cascade
  ) tablespace pg_default;

alter table user_themes enable row level security;

CREATE POLICY "Enable read access for all users" ON "public"."user_themes"
AS PERMISSIVE FOR SELECT
TO public
USING (true);

CREATE POLICY "Enable delete for users based on user_id" ON "public"."user_themes"
AS PERMISSIVE FOR ALL
TO public
USING (auth.uid() = user_id);
```

Рис. 3.9 SQL код для схеми бази даних

3.3.2 Менеджмент даних у застосунку

За допомогою Nuxt було реалізовано 6 API ендпоінтів для отримання та редагування даних у базі даних, представлено на рисунку 3.10:

- */initialize*

  Ініціалізує користувача в програмі, отримуючи інформацію про користувача та пов’язані з ним теми з бази даних. Це ендпоінт використовується для завантаження початкових даних, необхідних для запуску програми після автентифікації користувача;

- */theme/add-by-code*

  Призначений для додавання теми до профілю користувача на основі введеного коду теми. Він перевіряє, чи існує тема з заданим кодом, і додає її до списку тем користувача;

- */theme/create*

  Призначений для створення нової теми в системі на основі даних,

наданих користувачами. Це дозволяє авторизованим користувачам створювати нові потоки, які можна використовувати для гри та взаємодії з іншими користувачами;

- */theme/edit*

  Призначений для редагування існуючої теми в системі. Цей ендпоінт дозволяє авторизованим користувачам змінювати дані теми, яку вони створили, зокрема її назву та вміст;

- */theme/get-edit-data*

  Призначений для отримання даних існуючої теми, яку користувач бажає редагувати. Цей ендпоінт дозволяє авторизованим користувачам завантажити інформацію про тему за її ідентифікатором;

- */theme/ remove*

  Призначений для видалення існуючої теми з системи. Цей ендпоінт дозволяє авторизованим користувачам видалити тему за її ідентифікатором, а також пов’язані з нею записи в таблиці user_themes;

З клієнтської частини застосунку було створено Pinia “стори”, представлено на рисунку 3.10, які відповідають за певну логіку застосунку, а саме:

- app.store.ts

  Відповідає за базову ініціалізацію застосунку після його запуску у браузері. Перевіряє користувача, завантажує теми;

- auth.store.ts

  Відповідає за авторизацію та вихід користувача із системи;

- error.store.ts

  Відповідає за обробку помилок, вивід їх користувачу у вигляді повідомлення;

- gamification.store.ts

  Допомагає взаємодіяти з такими елементами геймифікації як досягненнями та балами;

- theme-create.store.ts

  Відповідає за створення теми;

- theme-edit.store.ts

  Відповідає за зміну теми;

- theme.store.ts

  Відповідає за загальний функціонал щодо тем, який може використовуватися у інших сторах;

- themes.store.ts

  Зберігає теми користувача та системні;

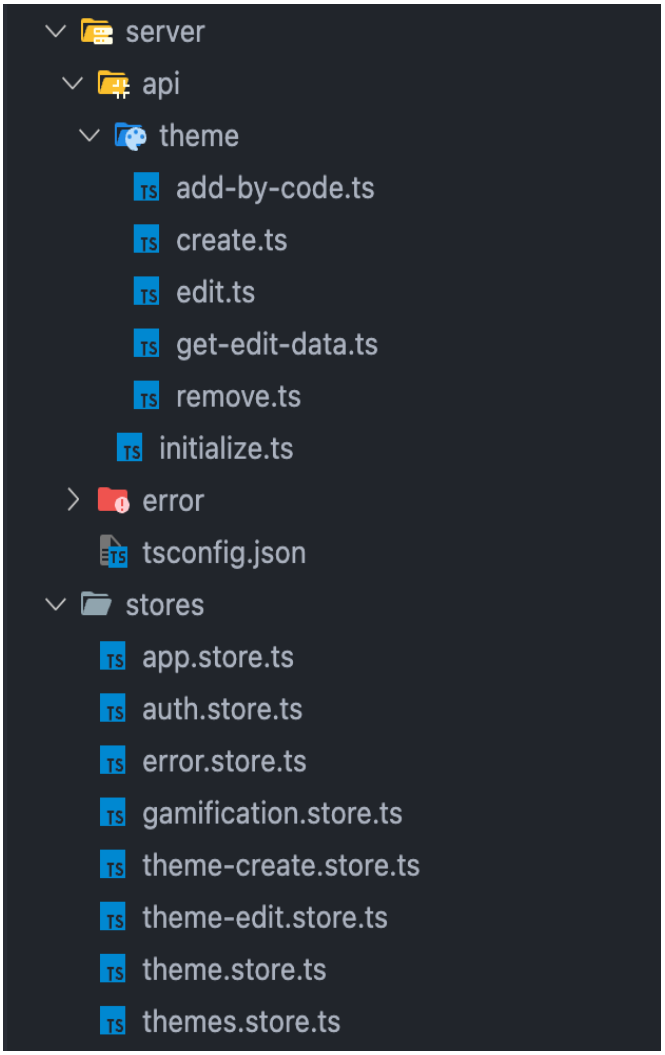


Рис. 3.10 файли у кодовій базі, що відповідають за менеджмент даних (ендпоінти та Pinia стори)

3.3.3 Карта web-застосунку

На основі вимог було розроблено карту web-застосунку, рисунок 3.11.

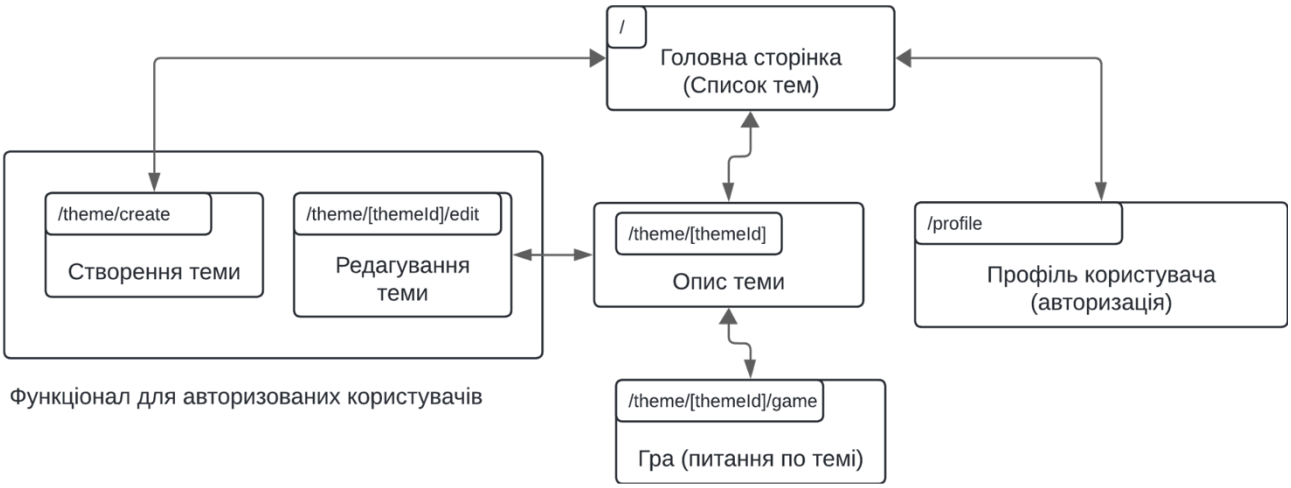


Рис. 3.11 Діаграма карти для розроблюваного застосунку

3.3.4 Розробка інтерфейсу користувача

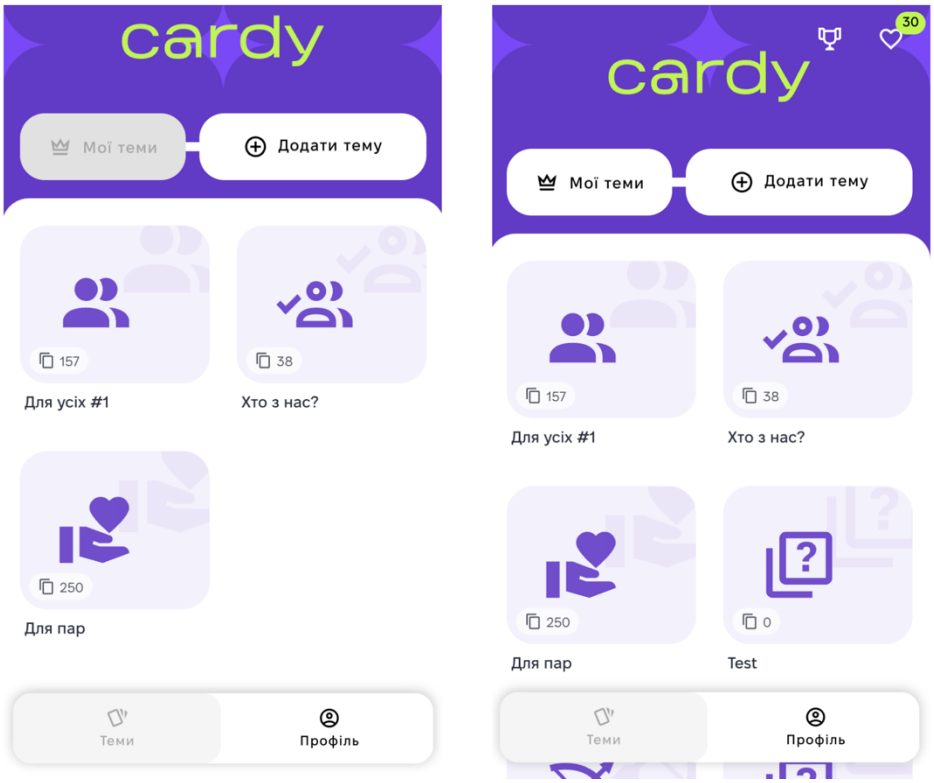


Рис. 3.12 Приклади екранних форм головної сторінки до і після авторизації

Застосунок починається з головної сторінки з набором готовий тем у які можна грати одразу, рисунок 3.12. Після авторизації на головній сторінці у правому верхньому кутку з’являються іконки для перегляду досягнень та балів.

Натиснувши на будь-яку з тем, ми потрапляємо на сторінку з деталями: ім’я, опис, іконка та кількість карток у темі, рисунок 3.13. Далі натиснувши на кнопку “Розпочати гру” ми потрапляємо на сторінку гри, де показано перше питання. Перегортати питання можна за допомогою стрілки, свайпу по карточці с питанням. Також можна обрати питання натиснувши на номер питання і ми побачимо список усіх.

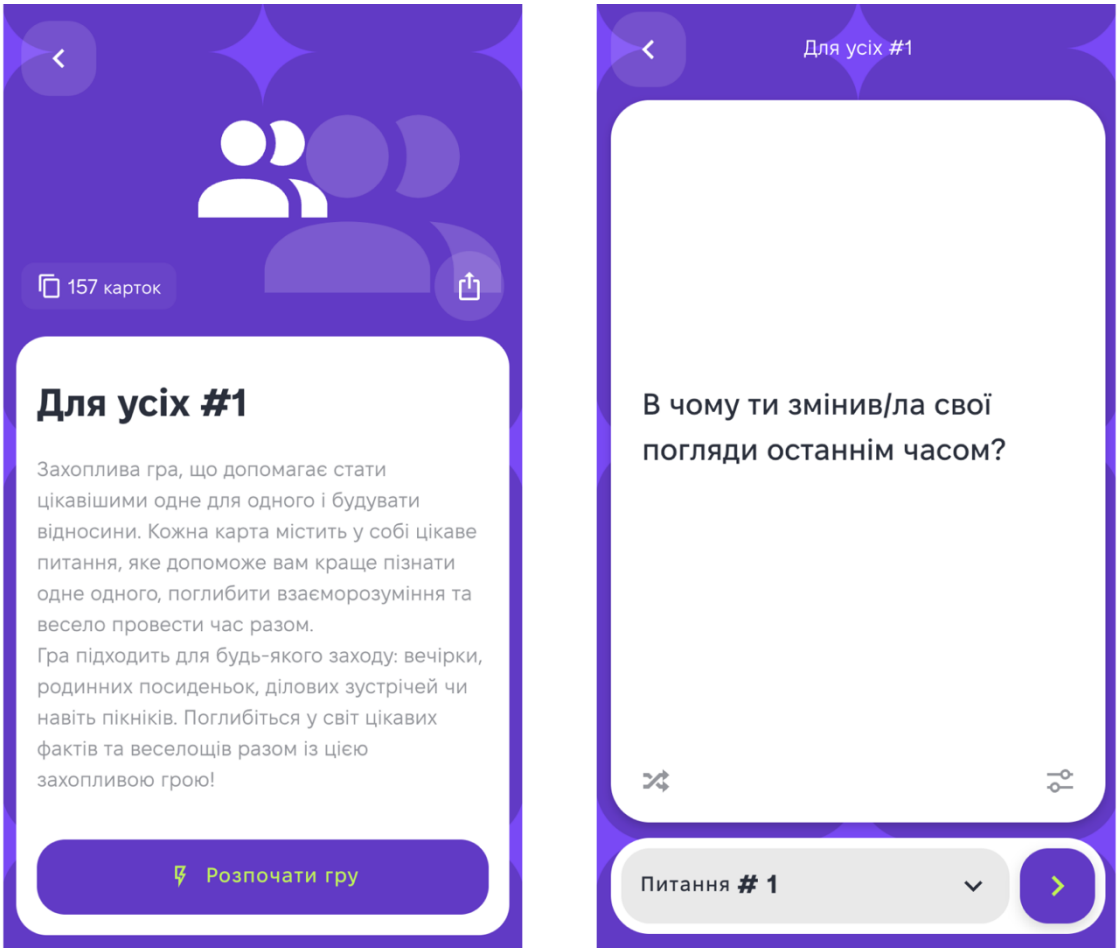


Рис. 3.13 Приклади екранних форм гри та деталей гри

Сторінка профілю має можливість увійти за допомогою “Google”. Після чого ми бачимо на сторінці електронну пошту та фото, що підтягується з облікового запису Google, представлено на рисунку 3.14.

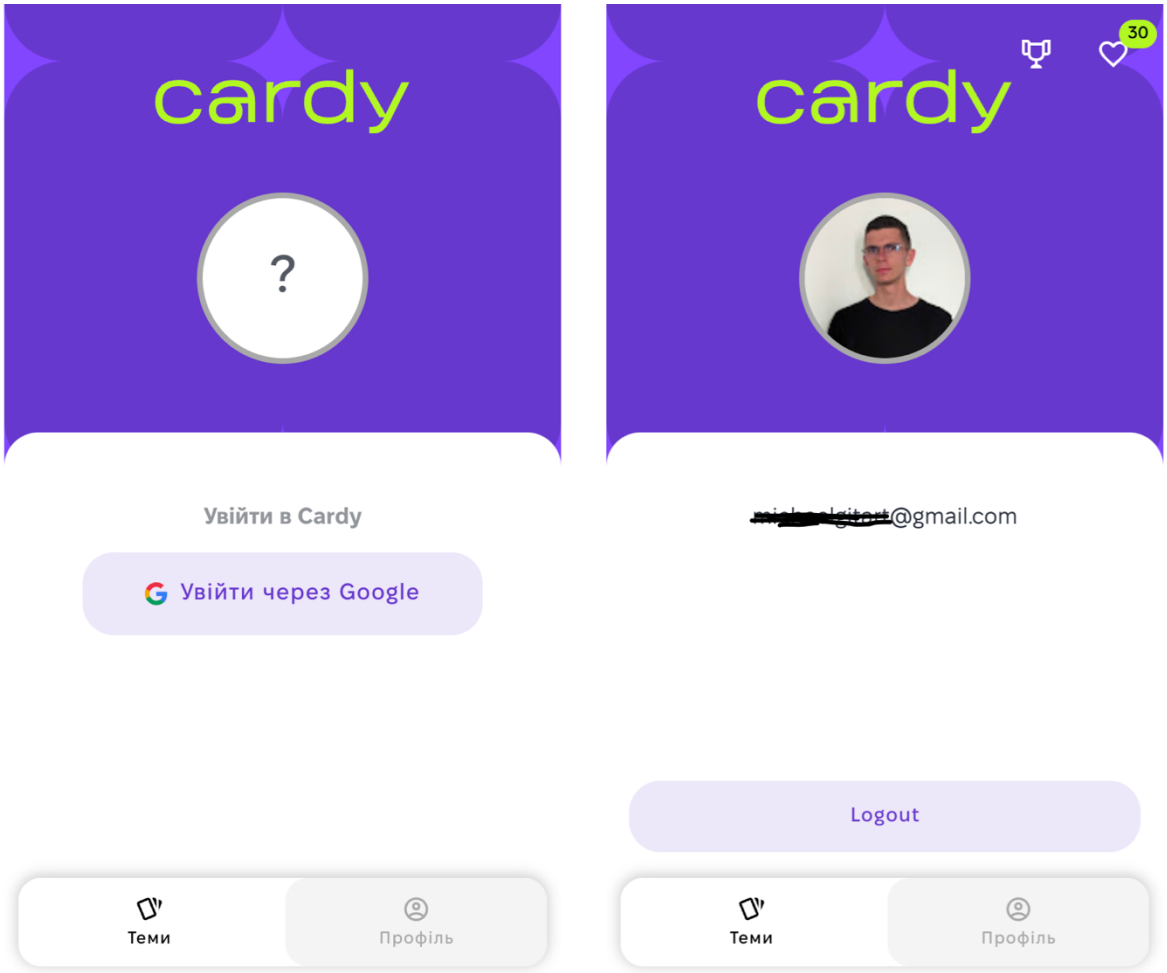


Рис. 3.14 Сторінка профілю до та після входу

Після авторизації, користувача з’являється можливість створити тему. Для цього необхідно натиснути “Додати тему” та вибрати “Створити тему”, представлено на рисунку 3.15. Після цього відкривається сторінка створення теми, представлено на рисунку 3.16.

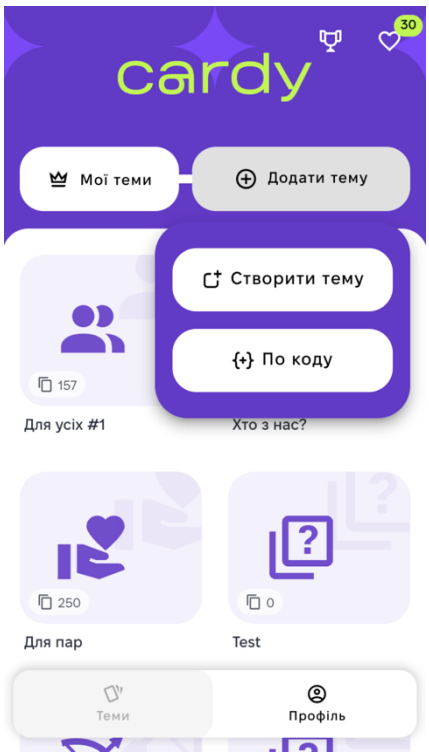


Рис. 3.15 Відкрите меню для можливості створити або додати тему по коду.

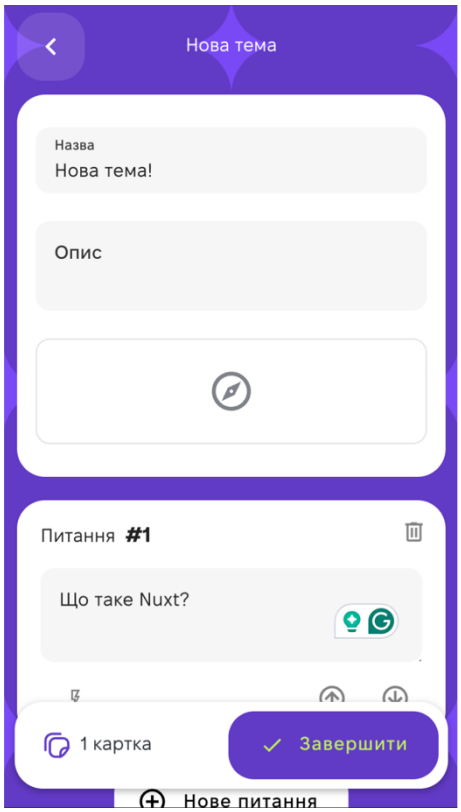


Рис. 3.16 Сторінка створення форми

Сторінка створення теми містить верхню форму для назви, опису та іконки. Далі іде набір питань та можливість завершити та зберегти тему, представлено на рисунку 3.16.

На сторінці деталей теми, якщо авторизований користувач є власником, з’являється додаткова кнопка з трьома крапками, яка відкриває доступ до таких кнопок як редагування та видалення, представлено рисунку 3.17.

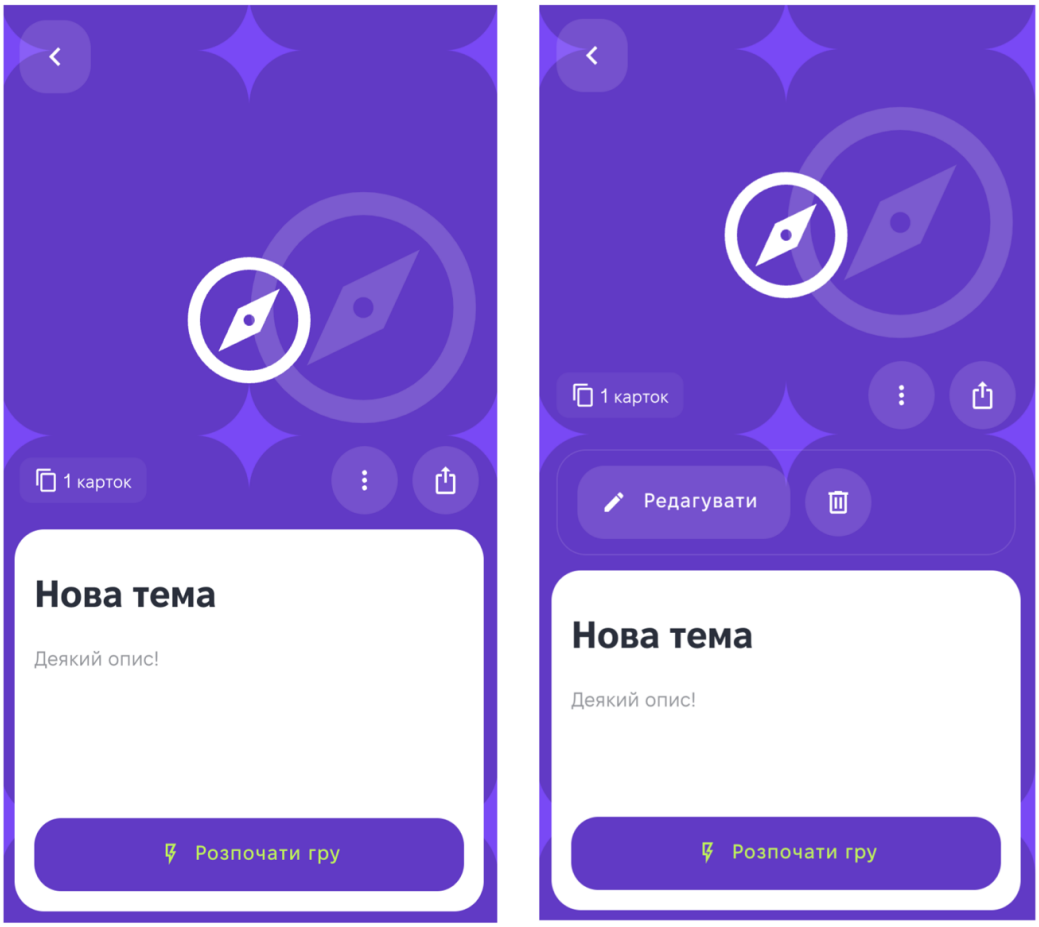


Рис. 3.17 Сторінка деталей теми для власника цієї теми

Щоб поділитися створеною темою, необхідно надіслати код теми для іншого користувача. Щоб це зробити необхідно відкривши відповідне меню як показано на рисунку 3.18. Використовуючи цей код інший користувач додає тему як показано на рисунку 3.19.

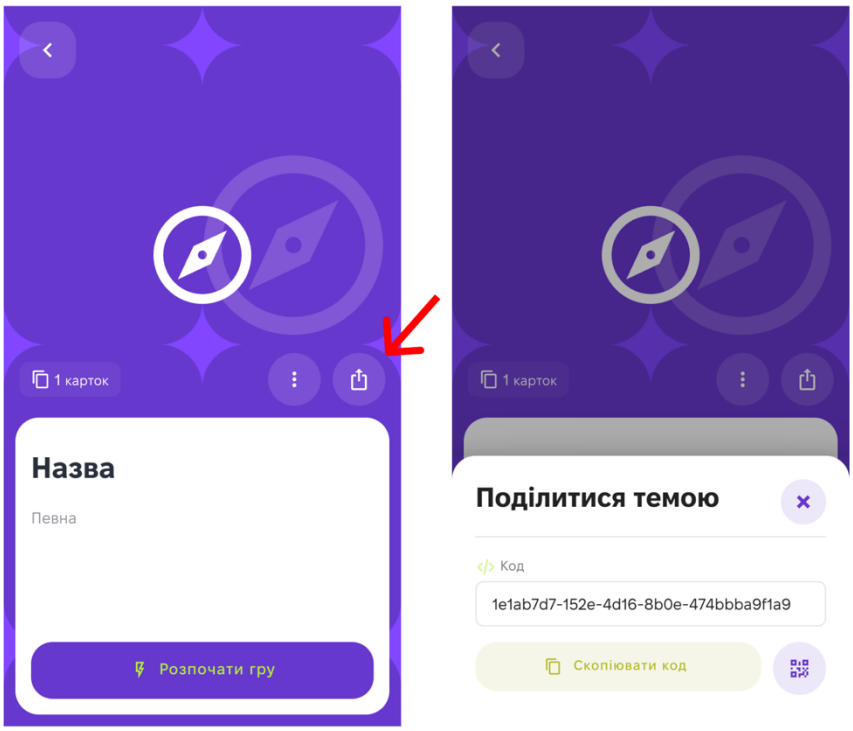


Рис. 3.18 Демонстрація як поділися власною темою

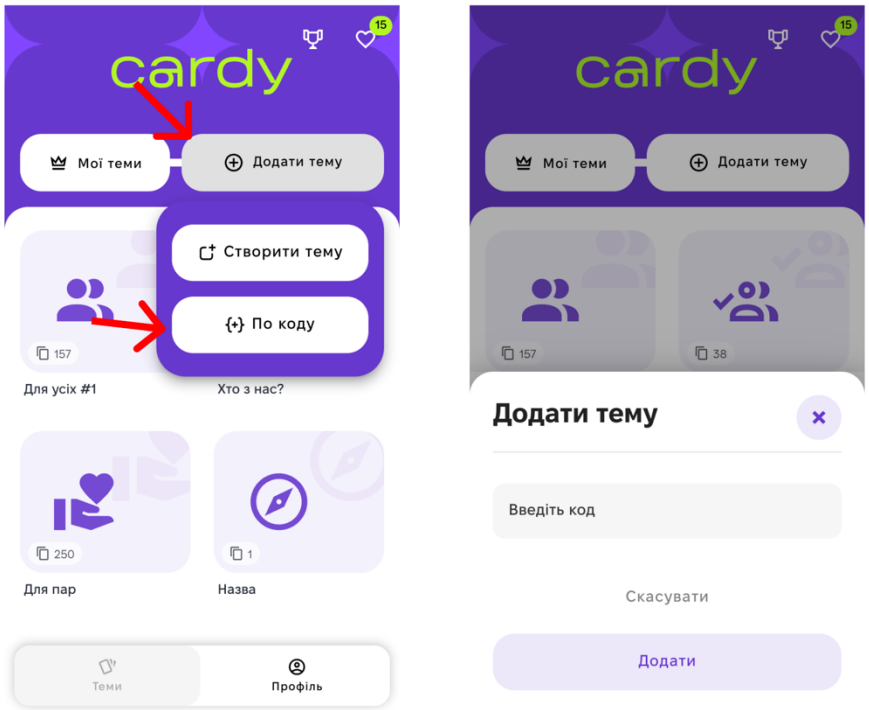


Рис. 3.19 Демонстрація як додати нову тему за кодом

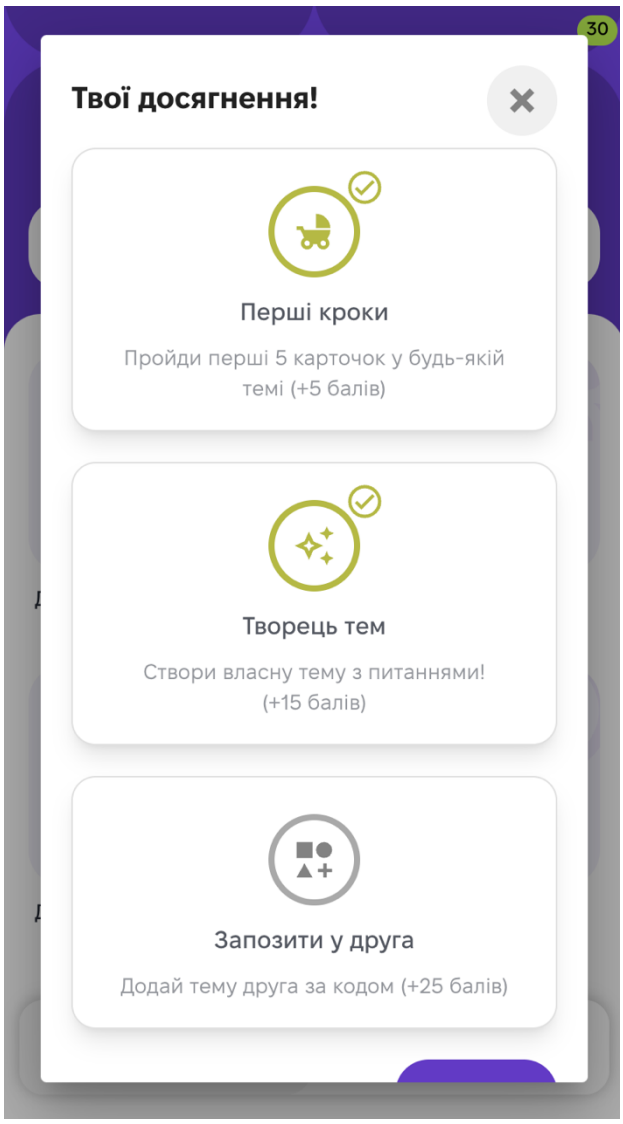


Рис. 3.20 Діалогове вікно досягнень

Після авторизації на головній та сторінці профілю у верхньому правому кутку з’явилися кнопки.

– Кнопка з іконкою “кубок”. Натиснувши на неї, відкривається діалогове вікно із списком досягнень які доступні та виконанні. Користувачу доступно три досягнення:
  - Перші кроки - здобувається після проходження перших 5 карток у будь-якій темі.
    Винагорода: +5 балів
  - Творець тем - здобувається після створення першої власної теми з питаннями.

Винагорода: +15 балів

- Запозити у друга - здобувається після додавання теми іншого користувача за кодом.

  Винагорода: +25 балів

За кожне досягнення нараховуються бали. На рисунку 3.20 видно, що користувач здобув перше та друге досягнення. За це йому додалося 20 балів.

– Кнопка з іконкою “сердечко”. Число біля кнопки відображає кількість балів накопичених користувачем. Після натискання відображається пояснення що таке бали, представлено на рисунку 3.21.

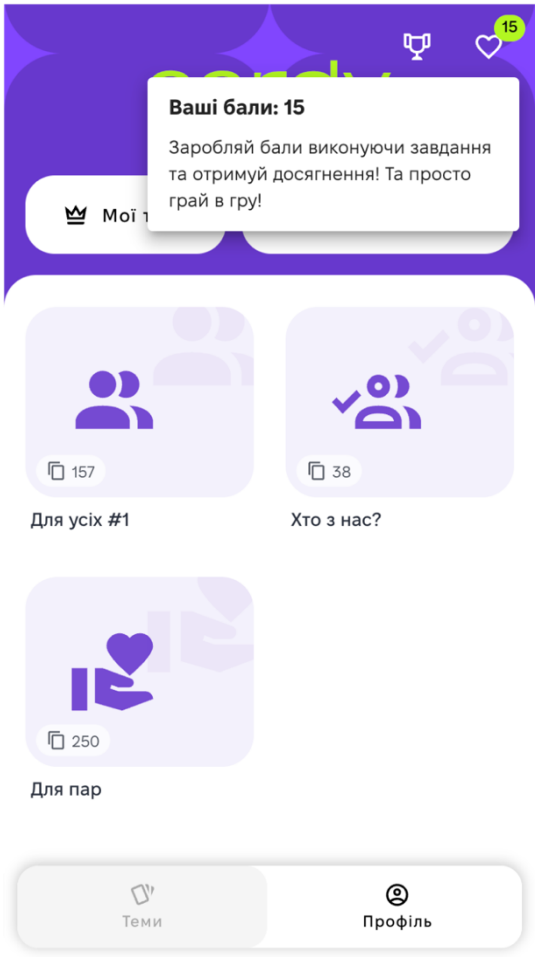


Рис. 3.21 Діалогове вікно балів

Система досягнень і нарахування балів у застосунку створює додаткову мотивацію для користувачів, заохочуючи їх активно взаємодіяти з контентом, проходити карткові ігри, створювати нові теми та ділитися ними з іншими. Відчуття прогресу та винагороди, отримані за досягнення, підвищують залученість користувачів, стимулюючи їх повертатися до застосунку та активно брати участь у його житті, що в кінцевому підсумку сприяє формуванню лояльності та тривалого інтересу до застосунку.

3.4 Аналіз відповідності вимогам

Можливість карткової гри за темами: у застосунку було реалізована можливість карткової гри за темами, представлено на рисунку 3.13.

Авторизація користувачів: авторизація користувачів реалізована за допомогою використання Supabase Auth, представлено на рисунку 3.14. У застосунку є можливість авторизуватися за допомогою облікового запису Google.

Можливість створювати власні теми з питаннями: можливість створення власних тем реалізована та доступна авторизованим користувачам, представлено на рисунку 3.16.

Можливість ділитися створеними темами між користувачами: можливість ділитися темами реалізована та продемонстрована на рисунках 3.18 та 3.19.

Можливість здобування досягнень для заохочення використати весь функціонал застосунку: у застосунку реалізовані досягнення. Їх список та інструкції для здобуття показано на рисунку 3.20.

Можливість накопичування балів для більшого залучення користувачів: накопичувальні бали реалізовані та їх можна набирати здобуваючи досягнення або під час гри по темах.

Кросбраузерність за стандартом Baseline: застосунок реалізовано використовуючи функціонал доступних у браузерах відповідно до стандарту

Baseline.

Відповідність стандартам Apple UX/UI: дотримані такі вимоги відповідно до Apple UX/UI як розмір кнопок для мобільних пристрої та доступність ключових елементів керування у нижній частині екрану [12].

Розробка під мобільні пристрої: усі сторінки застосунку реалізовані для використання на мобільних пристроях.

ВИСНОВКИ

Було розроблено web-застосунок, який інтегрує елементи гейміфікації для покращення соціальної взаємодії та зменшення відчуття самотності серед користувачів. Застосунок включає систему досягнень та накопичення балів, що стимулює активну участь користувачів у різних тематичних карткових іграх та спілкуванні. Користувачі мають можливість створювати власні теми, ділитися ними з іншими та отримувати винагороди за досягнення, що підвищує їхню залученість і мотивацію. Завдяки використанню сучасних технологій та ефективних гейміфікаційних стратегій, було створено інструмент, який не тільки покращує комунікаційні навички, але й сприяє формуванню нових соціальних зв’язків, підвищуючи загальне емоційне благополуччя користувачів.

Під час виконання роботи було досягнуто наступних результатів:

1. Проведено аналіз предметної області. Досліджено проблему соціальної взаємодії та самотності серед молоді. Виявлено ключові аспекти та причини самотності, зокрема емоційну та соціальну самотність;
2. Аналіз існуючих рішень показав, що існують додатки для зменшення самотності та підвищення соціальної взаємодії, такі як Noneliness, Happify, Playersgetready. Визначено переваги та недоліки цих додатків, на основі яких сформовано вимоги до розроблюваного застосунку;
3. Проектування застосунку включало розробку технічного завдання та визначення функціональних і нефункціональних вимог до системи. Спроектовано архітектуру застосунку, яка включає використання бази даних Supabase та технологій Nuxt.js для серверної частини;
4. Програмна реалізація охопила створення ключових можливостей застосунку, таких як авторизація користувачів, можливість створювати та редагувати теми, а також інтерактивну карткову гру за темами. Впроваджено елементи геймифікації для підвищення залученості

користувачів, такі як досягнення та накопичувальні бали;

5. Робота пройшла апробацію:

   Криворучко М.О., Зінченко О.В. ВИКОРИСТАННЯ NUXT ТА SUPABASE ПРИ РОЗРОБЦІ ВЕБЗАСТОСУНКУ. IV Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. Київ: ДУІКТ, 2024. С. 159-161.

ПЕРЕЛІК ПОСИЛАНЬ

1. Coping with Loneliness: Methods Young People Often Prefer1 [Електронний ресурс] // Richtmann Publishing. – 2014. – Режим доступу до ресурсу: https://www.richtmann.org/journal/index.php/jesr/article/view/4069.
2. Review: Interventions addressing loneliness amongst university students: a systematic review. [Електронний ресурс] // Child and Adolescent Mental Health. – 2022. – Режим доступу до ресурсу: https://acamh.onlinelibrary.wiley.com/doi/10.1111/camh.12614.
3. Loneliness in times of war [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.tandfonline.com/doi/full/10.1080/24720038.2023.2209610.
4. A Systematic Review of Loneliness Interventions Among Non-elderly Adults [Електронний ресурс] // School of Social Welfare, Stony Brook University, Stony Brook, USA. – 2019. – Режим доступу до ресурсу: https://link.springer.com/article/10.1007/s10615-019-00724-0.
5. How do lonely older people talk about loneliness? preliminary analysis of the bbc loneliness experiment [Електронний ресурс] // Oxford University Press on behalf of The Gerontological Society of America. – 2022. – Режим доступу до ресурсу: https://academic.oup.com/innovateage/article/6/Supplement_1/155/6936752?login=false.
6. Gamification for Health and Wellbeing : A Systematic Review of the Literature [Електронний ресурс] // Elsevier B.V.. – 2016. – Режим доступу до ресурсу: https://linkinghub.elsevier.com/retrieve/pii/S2214782916300380.
7. Noneliness: A Gamified Mobile App to Reduce Loneliness Among University Students [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://dl.acm.org/doi/10.1145/3450337.3483480.

8. Noneliness. Affective Lab. Режим доступу: https://gratitude.affective-lab.org/about?lang=en/
9. Happify. Режим доступу: https://www.happify.com/.
10. PLAYERSGETREADY. Режим доступу: https://www.playersgetready.com/pro-nas/.
11. MDN Web Docs [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/Baseline/Compatibility.
12. UI/UX Fundamental Design for Mobile Application Prototype to Support Web Accessibility and Usability Acceptance [Електронний ресурс] // ACM. ACM. – 2023. – Режим доступу до ресурсу: https://dl.acm.org/doi/10.1145/3587828.3587845.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-застосунку з елементами геймифікації для спілкування та знайомства з використанням JS, Vue, TS

Виконав студент 4 курсу
групи ПД-44
Криворучко Михайло Олегович
Керівник роботи
к.т.н, доцент кафедри ІПЗ Яскевич Владислав Олександрович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи - підтримка процесів соціальної взаємодії, знайомства та зменшення самотності з допомогою елементів геймифікації.

Об'єкт дослідження - соціальна взаємодія, знайомство та самотність.

Предмет дослідження - застосунок для соціальної взаємодії з елементами геймифікації.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати проблематику щодо соціальної взаємодії та проблеми самотності.
2. Зібрати інформацію про існуючі застосунки, виділити їх основні переваги та недоліки. На основі цього визначити вимоги до web-застосунку.
3. Дослідити оптимальні інструменти розробки та дизайн web-застосунку елементами гейміфікації.
4. Розробити архітектурну модель web-застосунку.
5. Розробити web-застосунку на основі вибраних інструментів, архітектури та визначених вимог.

3

АНАЛІЗ АНАЛОГІВ

Показник	Noneliness	Happify	Playersgetready	Cardy
Елемент геймифікації "досягнення"	+	-	-	+
Елемент геймифікації "бали"	+	+	-	+
Матеріали для розвитку комунікації	-	+	+	+
Можливість створювати власний контент	+	-	-	+
Можливість ділитися власним контентом	+	-	-	+
Взаємодія між користувачами	+	-	+	+
Відповідність стандартам Apple UX/UI	-	-	-	+
Широка цільова аудиторія	-	-	+	+

4

ВИМОГИ ДО WEB-ЗАСТОСУНКУ

Функціональні вимоги:

1. можливість карткової гри за темами;
2. авторизація користувачів;
3. можливість створювати власні теми з питаннями;
4. можливість ділитися створеними темами між користувачами;
5. можливість здобування досягнень для заохочення використати весь функціонал застосунку;
6. можливість накопичування балів для більшого залучення користувачів;

Нефункціональні вимоги:

1. кросбраузерність за стандартом Baseline;
2. відповідність стандартам Apple UX/UI;
3. розробка під мобільні пристрої;

5

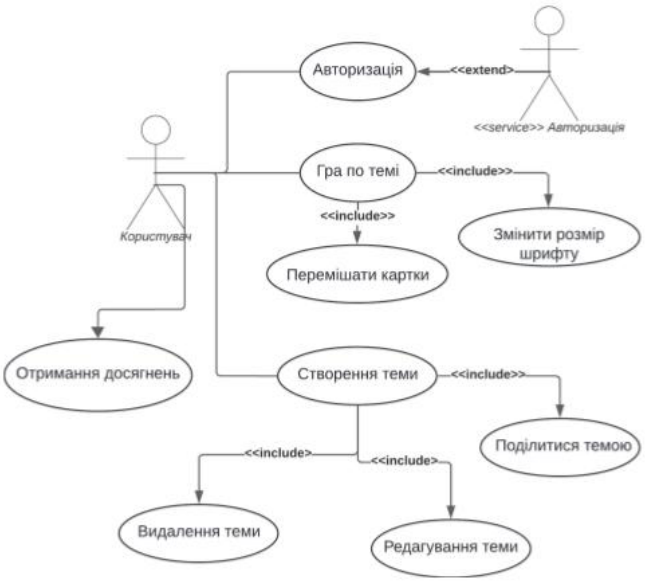
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Pinia

Postgresql

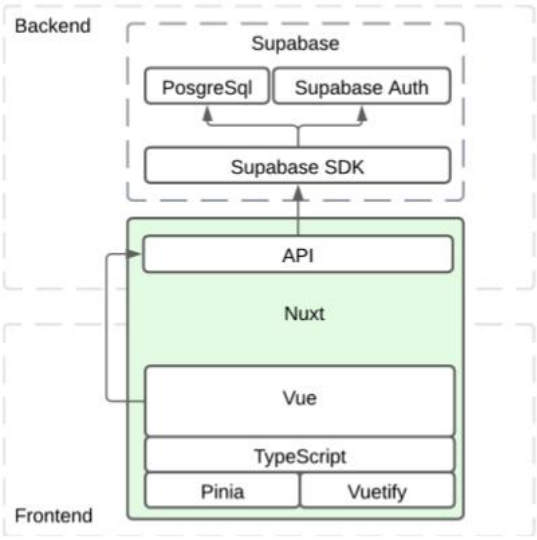
6

Діаграма прецедентів



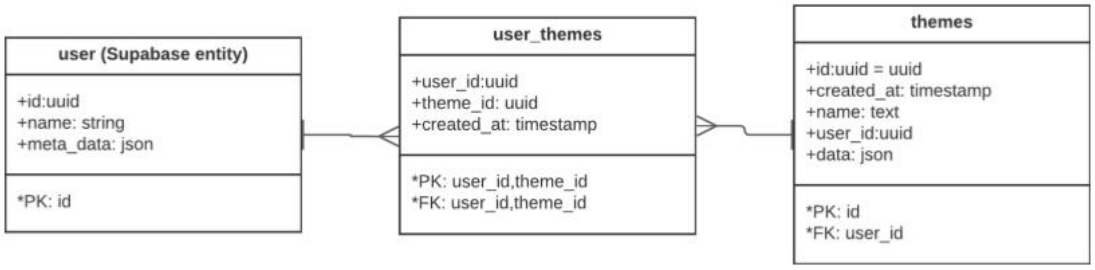
7

АРХІТЕКТУРА WEB-ЗАСТОСУНКУ



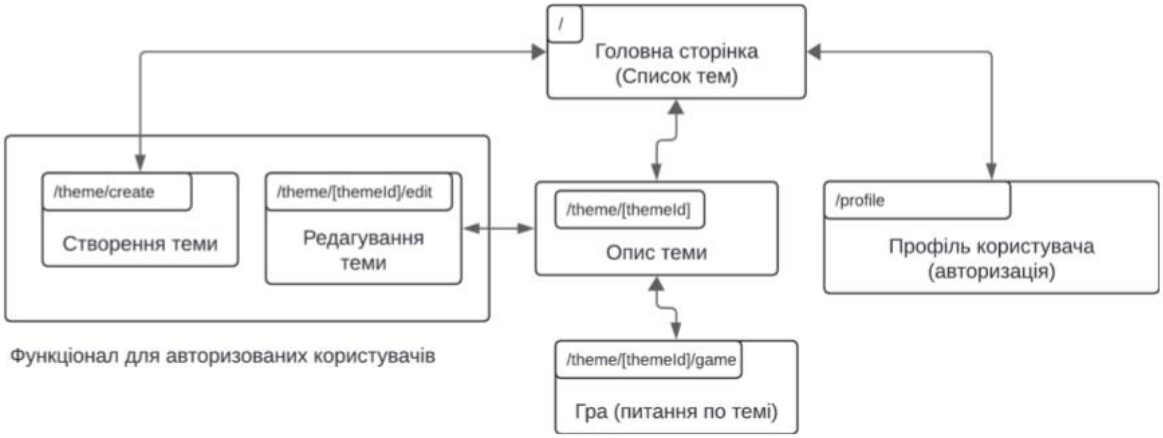
8

ДІАГРАМА КЛАСІВ



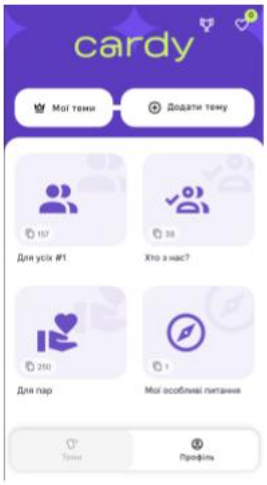
9

КАРТА WEB-ЗАСТОСУНКУ

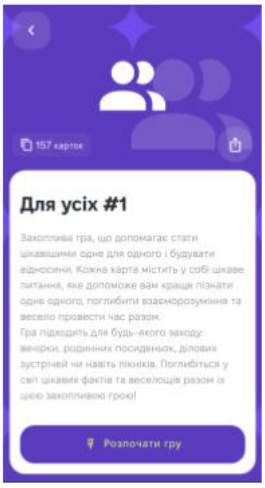


10

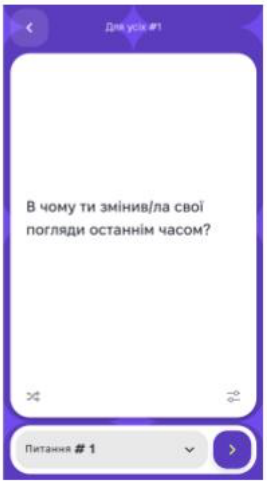
ЕКРАННІ ФОРМИ



Головна сторінка (список тем)



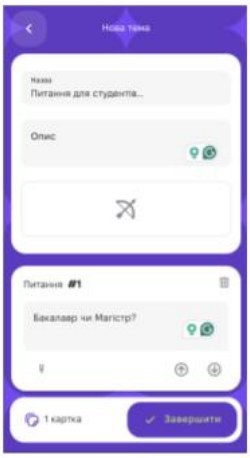
Опис теми



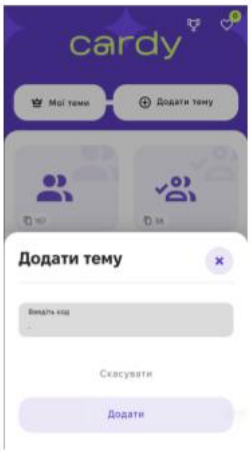
Гра по темі

11

ЕКРАННІ ФОРМИ

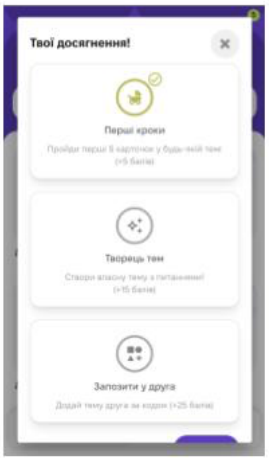


Створення власної теми



Додавання теми за кодом

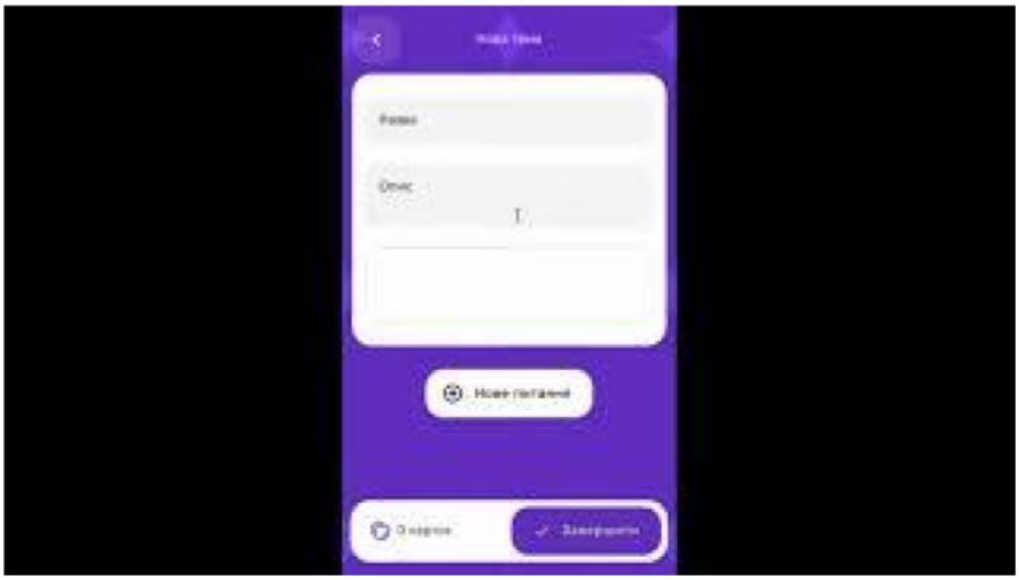
Отримання нового досягнення



Список досягнень

12

ДЕМОНСТРАЦІЯ РОБОТИ WEB-ЗАСТОСУНКУ



13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Криворучко М.О., Зінченко О.В. ВИКОРИСТАННЯ NUXT ТА SUPABASE ПРИ РОЗРОБЦІ ВЕБЗАСТОСУНКУ. IV Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15 травня 2024р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2024 с. 159-161.

14

ВИСНОВКИ

1. Проведено аналіз проблематики соціальної взаємодії та самотності.
2. Зібрано інформацію про застосунки для соціальної взаємодії та зменшення самотності (Noneliness, Happify, Playersgetready), виділено їх переваги та недоліки.
3. На основі аналізу предметної галузі та аналогів визначено вимоги до розроблюваного web-застосунку.
4. Визначено оптимальні інструменти: Vue, Typescript, Supabase, Nuxt та розроблена архітектура web-застосунку.
5. Розроблено web-застосунок відповідно до вимог.

ДОДАТОК В. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

app.store.ts

```
import { defineStore } from 'pinia'

import { useAuthStore } from './auth.store'

export const useAppStore = defineStore('app', () => {
const authStore = useAuthStore()
const initializeLoading = ref(false)

const user = computed(() => authStore.user)
const {
isChecked,
} = storeToRefs(authStore)

const initialize = async() => {
initializeLoading.value = true

const res = await $fetch('/api/initialize', {
headers: useRequestHeaders(['cookie']),
})

const authStore = useAuthStore()
const themeStore = useThemesStore()
authStore.initialize(res.user)
themeStore.initialize(res.data)

initializeLoading.value = false
}

return {
isChecked,
user,
initialize,
}
})
```

auth.store.ts

```
import { defineStore } from 'pinia'

import type { User } from '@supabase/supabase-js'

export const useAuthStore = defineStore('auth', () => {
const user = ref<null | User>(null)
const isChecked = ref(false)

const initialize = async(u: User | null) => {
if (!u) {
isChecked.value = true
return
}

user.value = u
isChecked.value = true
}

const loginWithGoogle = async() => {
const supabaseClient = useSupabaseClient()

await supabaseClient.auth.signInWithOAuth({
provider: 'google',
options: {
redirectTo: `${window.location.origin}/profile`,
},
})
}

const logout = async() => {
const supabaseClient = useSupabaseClient()
await supabaseClient.auth.signOut()
user.value = null

const themesStore = useThemesStore()
themesStore.initialize([])
}

const syncUserMetadata = async(data: any) => {
user.value!.user_metadata = data
}

return {
user,
isChecked,
initialize,
loginWithGoogle,
logout,
syncUserMetadata,
}
})
```

error.store.ts

```
import { errorMessages } from
```

```
'~/constants/errorMessages'

	const debug = false
	export const useErrorStore = defineStore('error', {

	actions: {
	getResErrorMessage(err: any): string {
	const key = (err.statusMessage) as keyof typeof errorMessages
	return errorMessages[key] || errorMessages.UNKNOWN_ERROR
	},

	handleError(err: any, options?: {
	skipNotify?: boolean
	}): any {
	// @ts-ignore
	if (err.handled)
	return err

	if (debug) {
	console.error(err)
	}

	const message = this.getResErrorMessage(err)
	if (!options?.skipNotify) {
	this.$notify.error(message)
	}

	// @ts-ignore
	err.handled = true

	return err
	},
	},
	})
```

gamification.store.ts

```
	import { defineStore } from 'pinia'

	import { mdiBabyCarriage, mdiCreationOutline, mdiShapePlus } from '@mdi/js'

	import { useGDialog } from 'gitart-manage-vue-dialog'

	import AchievementCompletedDialog from '~/components/Interface/Achievement/AchievementCompletedDialog/AchievementCompletedDialog.vue'

	export const ARCHIEVEMENTS_IDS = {
	FIRST_5_CARDS: 'first-5-cards',
	CREATE_OWN_THEME: 'create-own-theme',
	ADD_THEME_BY_CODE: 'add-theme-by-code',
	}

	const archievementsData = [
	{
	id: ARCHIEVEMENTS_IDS.FIRST_5_CARDS,
	name: 'Перші кроки',
	description: 'Пройди перші 5 карточок у будь-якій темі',
	icon: mdiBabyCarriage,
	score: 5,
	},
	{
	id: ARCHIEVEMENTS_IDS.CREATE_OWN_THEME,
	name: 'Творець тем',
	description: 'Створи власну тему з питаннями!',
	icon: mdiCreationOutline,
	score: 15,
	},
	{
	id: ARCHIEVEMENTS_IDS.ADD_THEME_BY_CODE,
	name: 'Запозити у друга',
	description: 'Додай тему друга за кодом',
	icon: mdiShapePlus,
	score: 25,
	},
	]

	interface UserArchievementData {
	archievements: string[]
	score: number
	}

	export const useGamificationStore = defineStore('gamification', () => {
	const authStore = useAuthStore()
	const isActive = computed(() => authStore.isChecked && !!authStore.user)

	const userArchievements = computed<UserArchievementData>(() => {
	if (!authStore.user || !authStore.user.user_metadata.gamification) {
	return {
	archievements: [],
	score: 0,
	}
	}

	const data = authStore.user.user_metadata.gamification as UserArchievementData

	return {
	archievements: data.archievements,
	score: data.score,
```

```
	}
	})

	const updateUserArchivementsData =
async(
	data:
Partial<UserArchievementData>,
	) => {
	const	supabaseClient	=
useSupabaseClient()

	const	res	=	await
supabaseClient.auth.updateUser({
	data: {
	gamification: {
	...userArchievements.value,
	...data,
	},
	},
	})

	authStore.syncUserMetadata(res.dat
a.user?.user_metadata)
	}

	const clearArchievements = async()
=> {
	await updateUserArchivementsData({
	archievements: [],
	score: 0,
	})
	}

	const $dialog = useGDialog()
	const	completeArchievement	=
async(id: string) => {
	const	archievement	=
archievementsData.find(a => a.id === id)

	if (!archievement) {
	return
	}

	const	archievements	=
userArchievements.value.archievements
	const	score	=
userArchievements.value.score	+
archievement.score

	await updateUserArchivementsData({
	archievements:	[...archievements,
id],
	score,
	})

	// @ts-expect-error
	$dialog.addDialog(AchievementCompl
etedDialog, {
	archievement,
	})
	}
	const	checkIfArchievementCompleted
= (id: string) => {
	return
userArchievements.value.archievements.inc
ludes(id)
	}

	const	addScore	=	async(value:
number) => {
	const	score	=
userArchievements.value.score + value

	await updateUserArchivementsData({
	score,
	})
	}

	const archievements = computed(()
=> {
	if (!authStore.user) {
	return []
	}

	return
archievementsData.map((archievement) => {
	return {
	...archievement,
	isCompleted:
userArchievements.value.archievements.inc
ludes(archievement.id),
	}
	})
	})

	return {
	userArchievements,
	archievements,
	clearArchievements,
	completeArchievement,
	addScore,
	isActive,
	checkIfArchievementCompleted,
	}
	})
```

theme-create.store.ts

```
	import	type	{	IThemeDB	}	from
'~/types/card'

	import	{	useThemesStore	}	from
'./themes.store'

	export const useThemeCreateStore =
defineStore('theme-create', {
	state: () => ({

	}),

	actions: {
	async createTheme(data: {
	name: string
	cards: Array<{
	id: string
	isAction: boolean
```

```
        text: string
        }>
        icon: string
        withParameters: boolean
        description: string
        }): Promise<IThemeDB> {
        const themeData: Partial<IThemeDB>
= {
        name: data.name,
        data: {
        description: data.description,
        cards: data.cards,
        icon: data.icon,
        withParameters:
data.withParameters,
        },
        }


        const createdTheme = await
$fetch('/api/theme/create', {
        method: 'POST',
        body: {
        data: themeData,
        },
        })


        useThemesStore().syncThemeAfterCre
ate(createdTheme.data)


        return createdTheme.data
        },
        },
        })
```

theme-edit.store.ts

```
        import type { IThemeDB } from
'@/types/card'

        import { useAuthStore } from
'./auth.store'

        export const useThemeEditStore =
defineStore('theme-edit', {
        state: () => ({


        }),


        actions: {
        async getThemeData(id: string):
Promise<IThemeDB> {
        const res = await
$fetch('/api/theme/get-edit-data', {
        params: {
        id,
        },
        })


        const themeData = res.data!


        const authStore = useAuthStore()
        if (themeData.user_id !==
authStore.user!.id) {
        throw new Error('User is not
defined')
        }


        return themeData
        },


        async editTheme(data: {
        id: string
        name: string
        cards: Array<{
        id: string
        isAction: boolean
        text: string
        }>
        icon: string
        withParameters: boolean
        description: string
        }): Promise<void> {
        const themeData: Partial<IThemeDB>
= {
        name: data.name,
        data: {
        description: data.description,
        cards: data.cards,
        icon: data.icon,
        withParameters:
data.withParameters,
        },
        }


        const updateTheme = await
$fetch('/api/theme/edit', {
        method: 'POST',
        body: {
        id: data.id,
        data: themeData,
        },
        })


        useThemesStore().syncThemeAfterEdi
t(updateTheme.data)
        },
        },
        })
```

theme.store.ts

```
        import type { BaseTheme } from
'@/types/card'

        export const useThemeStore =
defineStore('theme', {
        state: () => ({


        }),


        actions: {
        getThemeById(id: string):
BaseTheme {
        const themesStore =
useThemesStore()
```

```
      const theme =
themesStore.themes.find(theme => theme.id
=== id)
      if (theme) {
      return theme
      }

      throw new Error('Theme not found')
      },

      async removeThemeById(id: string)
{
      await $fetch('/api/theme/remove',
{
      method: 'POST',
      body: {
      id,
      },
      })

      const themesStore =
useThemesStore()
      themesStore.spliceThemeById(id)
      },
      },
      })
```

themes.store.ts

```
      import type { BaseTheme, IThemeDB
} from '@/types/card'

      import { localThemes } from
'@/data/questions'

      import { useAppStore } from
'./app.store'

      export const useThemesStore =
defineStore('themes', {
      state: () => ({
      themes: [
      ...localThemes,
      ] as BaseTheme[],
      }),

      actions: {
      async initialize(themes:
IThemeDB[]) {
      this.setThemes(themes)
      },

      async syncThemeAfterEdit(theme:
IThemeDB) {
      const themes =
this.themes.map<BaseTheme>((t) => {
      if (t.id === theme.id) {
      return {
      ...t,
      name: theme.name,
      description:
theme.data.description,
      cards: theme.data.cards,
      icon: theme.data.icon ||
'mdi-help-box-multiple-outline',
      withParameters:
theme.data.withParameters || false,
      }
      }

      return t
      })

      this.themes = themes
      },

      async syncThemeAfterCreate(theme:
IThemeDB) {
      this.themes = [
      ...this.themes,
      {
      id: theme.id,
      name: theme.name,
      description:
theme.data.description,
      icon: theme.data.icon ||
'mdi-help-box-multiple-outline',
      cards: theme.data.cards,
      isLocal: false,
      isOwner: true,
      withParameters:
theme.data.withParameters || false,
      },
      ]
      },

      async setThemes(dbThemes:
IThemeDB[]) {
      const appStore = useAppStore()

      const themes =
dbThemes.map<BaseTheme>(theme => ({
      id: theme.id,
      name: theme.name,
      description:
theme.data.description,
      icon: theme.data.icon ||
'mdi-help-box-multiple-outline',
      cards: theme.data.cards,
      isLocal: false,
      isOwner: theme.user_id ===
appStore.user!.id,
      withParameters:
theme.data.withParameters || false,
      }))

      this.themes = [
      ...localThemes,
      ...themes,
      ]
      },

      getThemeById(id: string):
BaseTheme {
      return this.themes.find(theme =>
theme.id === id)!
      },
```

```
      spliceThemeById(id: string) {
      const index = this.themes.findIndex(theme => theme.id === id)
      this.themes.splice(index, 1)
      },

      shuffleCards(themeId: string) {
      const theme = this.themes.find(theme => theme.id === themeId)

      if (!theme) {
      throw new Error('Theme not found')
      }

      theme.cards = theme.cards.sort(() => Math.random() - 0.5)
      },

      async addByCode(code: string): Promise<IThemeDB> {
      try {
      const res = await $fetch<any>('/api/theme/add-by-code', {
      query: {
      code,
      },
      })

      this.setThemes([res.data])
      return res.data
      }
      catch (error) {
      const errorStore = useErrorStore()
      throw errorStore.handleError(error)
      }
      },

      reset() {
      this.setThemes([])
      },
      },
      })
```

/server/api/theme/add-by-code.ts

```
      import type { IThemeDB } from '~/types/card'

      import { handleServerError } from '~/server/error/handleServerError'

      import { errorCodes } from '~/server/error/errors'

      import { serverSupabaseServiceRole, serverSupabaseUser } from '#supabase/server'

      export default defineEventHandler(async(event): Promise<{
      data: null | any
      success: boolean
      }> => {
      try {
      const user = await serverSupabaseUser(event)

      if (!user) {
      throw createError({
      statusCode: 401,
      message: 'No user',
      })
      }

      const query = getQuery(event)
      const code = query.code as string

      const client = serverSupabaseServiceRole<any>(event)

      const selectThemeRes = await client.from('themes').select('*').eq('id', code)
      const theme = (selectThemeRes.data?.[0] || null) as IThemeDB | null

      if (selectThemeRes.error || !theme) {
      throw createError({
      statusCode: 404,
      statusMessage: errorCodes.THEME_BY_CODE_NOT_FOUND,
      message: 'Failed to find theme',
      })
      }

      const insertUserThemesRes = await client
      .from('user_themes')
      .insert({
      user_id: user!.id,
      theme_id: theme!.id,
      })
      .select('*')

      if (insertUserThemesRes.error) {
      const statusMessage = insertUserThemesRes.error.code === '23505'
      ? errorCodes.THEME_ALREADY_ADDED
      : errorCodes.FAILED_TO_ADD_THEME_BY_CODE

      throw createError({
      statusCode: 500,
      statusMessage,
      })
      }
```

```
	const selectUpdatedThemeRes = await client
	.from('themes')
	.select('*, user_themes!inner(user_id)')
	.eq('user_themes.user_id', user.id)

	if (selectUpdatedThemeRes.error) {
	throw createError({
	statusCode: 500,
	})
	}

	return {
	data: selectUpdatedThemeRes.data![0],
	success: true,
	}
	}
	catch (error: any) {
	throw handleServerError(error)
	}
	})
```

/server/api/theme/create.ts

```
	import type { IThemeDB } from '~/types/card'

	import { handleServerError } from '~/server/error/handleServerError'

	import { errorCodes } from '~/server/error/errors'

	import { serverSupabaseServiceRole, serverSupabaseUser } from '#supabase/server'

	export default defineEventHandler(async(event): Promise<{
	data: IThemeDB
	success: boolean
	}> => {
	try {
	const user = await serverSupabaseUser(event)

	if (!user) {
	throw createError({
	statusCode: 401,
	message: 'No user',
	})
	}

	const body = await readBody(event)
	const id = body.id as string

	const data = body.data as Partial<IThemeDB>

	const client = serverSupabaseServiceRole<any>(event)

	const createThemeRes = await client.from('themes').insert({
	name: data.name,
	data: data.data,
	user_id: user.id,
	}).eq('id', id).select('*')

	if (createThemeRes.error) {
	throw createError({
	statusCode: 404,
	statusMessage: errorCodes.FAILED_TO_CREATE_THEME,
	message: 'Failed to create theme',
	})
	}

	const theme = createThemeRes.data![0] as IThemeDB

	const themeUserRelationRes = await client.from('user_themes').insert({
	theme_id: theme.id,
	user_id: user.id,
	}).eq('theme_id', theme.id).eq('user_id', user.id).select('*')

	if (themeUserRelationRes.error) {
	throw createError({
	statusCode: 404,
	statusMessage: errorCodes.FAILED_TO_CREATE_THEME,
	message: 'Failed to create theme',
	})
	}

	return {
	data: theme,
	success: true,
	}
	}
	catch (error: any) {
	throw handleServerError(error)
	}
	})
```

/server/api/theme/edit.ts

```
	import type { IThemeDB } from '~/types/card'

	import { handleServerError } from '~/server/error/handleServerError'

	import { errorCodes } from
```

```
'~/server/error/errors'

      import                              {
serverSupabaseServiceRole,
serverSupabaseUser         }         from
'#supabase/server'

      export                        default
defineEventHandler(async(event):
Promise<{
      data: IThemeDB
      success: boolean
      }> => {
      try {
      const      user      =      await
serverSupabaseUser(event)

      if (!user) {
      throw createError({
      statusCode: 401,
      message: 'No user',
      })
      }

      const body = await readBody(event)
      const id = body.id as string
      const   data   =   body.data   as
Partial<IThemeDB>

      const           client           =
serverSupabaseServiceRole<any>(event)

      const   updateThemeRes   =   await
client.from('themes').update({
      name: data.name,
      data: data.data,
      }).eq('id', id).select('*')

      if (updateThemeRes.error) {
      throw createError({
      statusCode: 404,
      statusMessage:
errorCodes.FAILED_TO_EDIT_THEME,
      message: 'Failed to edit theme',
      })
      }

      return {
      data: updateThemeRes.data![0],
      success: true,
      }
      }
      catch (error: any) {
      throw handleServerError(error)
      }
      })
```

/server/api/theme/get-edit-data.ts

```
      import  type  {  IThemeDB  }  from
'~/types/card'

      import { handleServerError } from
'~/server/error/handleServerError'

      import   {   errorCodes   }   from
'~/server/error/errors'

      import                              {
serverSupabaseServiceRole,
serverSupabaseUser         }         from
'#supabase/server'

      export                        default
defineEventHandler(async(event):
Promise<{
      data: IThemeDB
      success: boolean
      }> => {
      try {
      const      user      =      await
serverSupabaseUser(event)

      if (!user) {
      throw createError({
      statusCode: 401,
      message: 'No user',
      })
      }

      const query = getQuery(event)
      const id = query.id as string

      const           client           =
serverSupabaseServiceRole<any>(event)

      const   selectThemeRes   =   await
client.from('themes').select('*').eq('id'
, id)
      const           theme           =
(selectThemeRes.data?.[0]   ||   null)   as
IThemeDB | null

      if     (selectThemeRes.error     ||
!theme) {
      throw createError({
      statusCode: 404,
      statusMessage:
errorCodes.THEME_BY_CODE_NOT_FOUND,
      message: 'Failed to find theme',
      })
      }

      return {
      data: theme,
      success: true,
      }
      }
      catch (error: any) {
      throw handleServerError(error)
      }
      })
```

/server/api/theme/remove.ts

```
import { handleServerError } from '~/server/error/handleServerError'

import { serverSupabaseServiceRole, serverSupabaseUser } from '#supabase/server'

export default defineEventHandler(async(event): Promise<{
success: boolean
}> => {
try {
const user = await serverSupabaseUser(event)

if (!user) {
throw createError({
statusCode: 401,
message: 'No user',
})
}

const body = await readBody(event)
const id = body.id as string

const client = serverSupabaseServiceRole<any>(event)

await client.from('themes').delete().eq('id', id).select('*')
await client.from('user_themes').delete().eq('theme_id', id)

return {
success: true,
}
}
catch (error: any) {
throw handleServerError(error)
}
})
```

/server/api/initialize.ts

```
import type { User } from '@supabase/supabase-js'

import { serverSupabaseServiceRole, serverSupabaseUser } from '#supabase/server'

const debug = false
const debugUser = {
id: '7c2f5540-f439-4a88-bc13-04e7ee0c8f73',
email: 'michaelgitart@gmail.com',
created_at: '2023-11-08T06:15:36.387348Z',
updated_at: '2024-03-21T22:15:40.566619Z',
}

export default defineEventHandler(async(event): Promise<{
user: User | null
data: Array<any>
}> => {
const client = serverSupabaseServiceRole(event)

if (debug) {
const client = serverSupabaseServiceRole(event)

const { data } = await client
.from('themes')
.select('*, user_themes!inner(user_id)')

return {
data: data!,
user: debugUser as any,
}
}

const user = await serverSupabaseUser(event)

if (!user) {
return {
data: [],
user,
}
}

const { data } = await client
.from('themes')
.select('*, user_themes!inner(user_id)')
.eq('user_id', user.id)

if (!data) {
throw new Error('No data')
}

return {
data,
user,
}
})
```

AppLayout.vue

```
<script setup lang="ts">
import { VApp } from
```

```
'vuetify/components'
      import   {   NotifySpawn   }   from
'gitart-vue-notify'
      import   {   GDialogSpawn   }   from
'gitart-manage-vue-dialog'

      import   {   useAppStore   }   from
'~/stores/app.store'

      import        AppToast        from
'./AppToast.vue'

      const appStore = useAppStore()

      const isUserChecked = ref(false)
      const     isFirstRouteLoaded     =
ref(false)

      const nuxtApp = useNuxtApp()
      ;(async() => {
      await appStore.initialize()
      isUserChecked.value = true

      const            stop            =
nuxtApp.hook('page:finish', async() => {
      isFirstRouteLoaded.value = true
      stop()
      })
      })()
      </script>

      <template>
      <VApp class="app-layout">
      <VFadeTransition>
      <template                          v-
if="!isFirstRouteLoaded">
      <div  class="absolute  z-50  left-0
right-0  top-0  bottom-0  bg-tw-white  flex
items-center justify-center">
      <div class="pb-50">
      <div   class="flex   justify-center
mb-8">
      <div class="bg-white rounded-md h-
40 w-32 shadow shadow-lg px-5 py-8">
      <div class="w-full h-4 bg-gray-200
rounded-xl mb-3" />
      <div  class="w-2/3  h-4  bg-gray-200
rounded-xl" />
      </div>
      </div>

      <VProgressLinear
      indeterminate
      color="primary"
      class="w-64 rounded"
      height="8"
      />
      </div>
      </div>
      </template>
      </VFadeTransition>

      <NuxtLayout v-if="isUserChecked">
      <NuxtPage />
      </NuxtLayout>

      <NotifySpawn>
      <template  #item="{message,  type,
close, progress}">
      <AppToast
      class="mb-3"
      :message="message"
      :type="type"
      :progress="progress"
      @close="close()"
      />
      </template>
      </NotifySpawn>

      <GDialogSpawn />
      </VApp>
      </template>

      <style lang="scss">
      .app-layout {
      background:  url('/bg.svg')  repeat
top center fixed, #7949F5;
      background-size: 100% auto;
      background-position: 0 40px;

      max-width: 500px;
      margin-left: auto;
      margin-right: auto;
      }
      </style>
```

App.vue

```
      <script setup lang="ts">
      import        AppLayout        from
'@/components/Interface/AppLayout/AppLayo
ut.vue'
      </script>

      <template>
      <AppLayout />
      </template>
```

main.scss

```
      @use 'vuetify' with (
      $reset: false,
      $utilities: false,
      $color-pack: false,
      );

      @import './fonts.scss';

      body {
      background: #613AC5;
      }

      .v-notify-container {
      @apply
      pb-12
```

```
        left-0 right-0
        ;
        }

        .v-notify-item {
        @apply
        pointer-events-none
        ;

        & > * {
        @apply
        pointer-events-auto
        ;
        }
        }



        .layout-enter-active,
        .layout-leave-active {
        transition: all 0.2s;
        }
        .layout-enter-from,
        .layout-leave-to {
        opacity: 0;
        }

        .page-enter-active,
        .page-leave-active {
        transition: all 0.2s;
        }
        .page-enter-from,
        .page-leave-to {
        opacity: 0;
        }
```

nuxt.config.ts

```
import  vuetify,  {  transformAssetUrls  }
from 'vite-plugin-vuetify'

        export default defineNuxtConfig({
        ssr: false,

        devtools: { enabled: false },
        build: {
        transpile: ['vuetify'],
        },

        components: false,

        plugins:
['~/plugins/vuetify/vuetify'],

        modules: [
        '@unocss/nuxt',
        '@nuxtjs/supabase',
        '@pinia/nuxt',
        (_options, nuxt) => {
        nuxt.hooks.hook('vite:extendConfig
', (config) => {
        // @ts-expect-error
        config.plugins.push(vuetify({
        autoImport: true,
        styles: {
        configFile:
'plugins/vuetify/settings.scss',
        },
        }))
        })
        },
        ],

        supabase: {
        redirect: false,
        },

        imports: {
        presets: [
        ],

        imports: [
        {
        from: 'gitart-vue-notify',
        name: 'useNotify',
        },
        ],
        },

        app: {
        pageTransition:  {  name:  'page',
mode: 'out-in' },
        layoutTransition:      {      name:
'layout', mode: 'out-in' },

        head: {
        title: 'Cardy | Психологічна гра',
        link: [
        {
        rel: 'icon',
        type: 'image/x-icon',
        href: '/favicon.png',
        },
        ],
        },
        },

        css: ['@/scss/main.scss'],

        vite: {
        vue: {
        template: {
        transformAssetUrls,
        },
        },
        },

        })
```