МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МОБІЛЬНИЙ ЗАСТОСУНОК ПЕРСОНАЛЬНОГО ПЛАНУВАЛЬНИКА СПРАВ»

Рівень «бакалавр»
Галузь знань 12 «Інформаційні технології»
спеціальність 121 «Інженерія програмного забезпечення»
Виконала здобувачка 4 курсу
Антіпіна Надія Сергіївна

Керівник к.т.н., доцент
Трофименко Олена Григорівна

Рецензент: к.т.н., доцент
Манаков Сергій Юрійович

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»

Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційні технології
Галузь знань: 12 Інформаційні технології
Спеціальність: 121 Інженерія програмного забезпечення
Назва освітньої програми: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент Н.І. Логінова
_____ «___» _____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачці Антіпіній Надії Сергіївні

1. Тема роботи: «Мобільний застосунок персонального планувальника справ»
   Керівник роботи: к.т.н, доцент Трофименко Олена Григорівна
   затверджені наказом НУ «ОЮА» від «02» квітня 2024 року № 809-06
2. Дата видачі завдання «15» вересня 2023 року
3. Строк подання здобувачем роботи «20» травня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Формування функціональних вимог	20.10.2023	
2	Аналіз та вибір засобів розробки	10.11.2023	
3	Розробка архітектури застосунку	30.11.2023	
4	Проєктування бази даних	25.12.2023	
5	Розробка серверної частини	12.02.2024	
6	Розробка клієнтської частини	19.03.2024	
7	Тестування функціональності сайту	20.04.2024	
8	Оформлення звітної частини	19.05.2024	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Антіпіна Н.С. Мобільний застосунок персонального планувальника справ. – Кваліфікаційна робота бакалавра за спеціальністю 121 «Інженерія програмного забезпечення».

Кваліфікаційна робота викладена на 64 сторінках основного тексту і 88 сторінках загального тексту, містить 3 розділи, 35 рисунків, 11 таблиць, 2 додатки, 23 використаних джерел.

Метою роботи є розроблення застосунку персонального планувальника справ, що надаватиме можливості формувати перелік щоденних завдань, структурувати план заходів у зручний для користувача спосіб та фіксувати прогрес їх виконання.

Об’єктом дослідження є специфіка розробки мобільних застосунків засобами ASP.NET.

Предмет дослідження – засоби розробки мобільного застосунку з планування справ.

В першому розділі досліджено специфіку предметної області розробки мобільних застосунків з ігровим елементом, вибір засобів розробки та маркетингових інструментів. В другому розділі здійснено проєктування застосунку. У третьому розділі описана реалізація функціоналу й інтерфейсу застосунку за допомогою засобів React.js та ASP.NET, перетворення вебсторінок на гібридний мобільний застосунок засобами фреймворка Capacitor.js та тестування його функціональних можливостей.

Ключові слова: мобільний застосунок, планувальник справ, тамагочі, ASP.NET, React.js, Capacitor.js.

ABSTRACT

Antipina N.S. Personal planner mobile application. – Bachelor's qualifying work on specialty 121 "Software engineering".

The work consists of 64 pages of main text and 88 pages of total text, comprising 3 chapters, 35 figures, 11 tables, 2 appendices, and 23 used sources.

The goal of the work is a development of personal planner mobile application that would be able to form a list of daily tasks, structure it in a convenient for users way, track tasks' completion progress.

The object of the study is the specifics of mobile applications' development using ASP.NET.

The subject of research is technology tools for developing personal planner mobile application.

In the first section, the specifics of the subject area of mobile application development with a gaming element, the selection of development tools and marketing instruments were examined. In the second section, the application design was carried out. The third section described the implementation of the functionality and interface of the application using React.js and ASP.NET, the transformation of web pages into a hybrid mobile application using the Capacitor.js framework, and the testing of its functional capabilities.

Keywords: mobile application, planner, tamagotchi, ASP.NET, React.js, Capacitor.js.

ЗМІСТ

ВСТУП 6
1 АНАЛІЗ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ 8
1.1 Аналіз особливостей розробки 8
1.2 Аналіз наявних аналогів 10
1.3 Розробка функціональних вимог 13
1.4 Вибір засобів розробки застосунку 15
1.4.1 Вибір засобів розробки бази даних 17
1.4.2 Вибір засобів розробки backend частини 18
1.4.3 Вибір засобів розробки frontend частини 20
1.5 Маркетингові інструменти в створенні та просуванні програмного продукту 20
1.6 Висновки до розділу 1 23
2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ 25
2.1 Архітектура програмного застосунку 25
2.2 Проєктування бази даних 27
2.3 Моделювання програмної системи 31
2.4 Висновки до розділу 2 33
3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 35
3.1 Розробка бази даних 35
3.2 Розробка серверної частини з використанням ASP.NET 40
3.3 Результат роботи програми 48
3.4 Тестування програмного продукту 57
3.5 Висновки до розділу 3 60
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 63
ДОДАТКИ 65
Додаток А. Фрагменти програмного коду 65
Додаток Б. Копії документів про апробацію результатів роботи 79

ВСТУП

Сучасний ритм життя є стрімким та потребує від людини навичок ефективного управління своїм часом, оптимізації робочого навантаження, концентрації уваги на важливих щоденних аспектах. Оскільки тримати весь перелік справ, планів, зустрічей тощо в голові цілий день – це складна та виснажлива задача, різноманітні інструменти для фіксування порядку денного стали популярними щоденними помічниками. З появою комп'ютерних технологій їхні можливості суттєво розширилися: записи у щоденнику, позначки у календарі, надрукований розклад та багато інших пунктів тепер можна зібрати в одному мобільному застосунку.

Поганий тайм-менеджмент, складність концентрації на поточних завданнях та велика кількість справ, що легко забути є постійними проблемами багатьох користувачів. Водночас популярними залишаються застосунки для догляду за віртуальними вихованцями, стан яких підтримується шляхом регулярного проходження квестів та приділення їм уваги кожного дня. Якщо поєднати процеси планування та піклування за улюбленою рослиною чи твариною в один застосунок, з'являється можливість підтримувати зацікавленість користувачів у виконанні щоденних завдань та стимулювати формування корисних довготривалих звичок. Оскільки мобільні пристрої стали невід'ємною частиною нашого повсякденного життя, саме їх доцільно використовувати для успішного планування справ протягом дня та контролю за їх виконанням. Тому тема розроблення мобільного застосунку персонального планувальника справ як помічника в подоланні щоденних труднощів та пошуку сил у різні періоди життя є вельми актуальною.

Метою роботи є розроблення застосунку персонального планувальника справ, що матиме можливості формувати перелік щоденних завдань, структурувати план зручним для користувача шляхом, фіксувати прогрес виконання.

Об'єктом дослідження є специфіка розробки мобільних застосунків засобами ASP.NET.

Предмет дослідження – засоби розробки мобільного застосунку з планування справ.

Відповідно до поставленої мети **завданнями** роботи є:

– аналіз предметної області з оглядом наявних аналогів;

– формування функціональних вимог та вибір засобів розробки;

– проєктування бази даних та моделювання програмної системи застосунку;

– розробка клієнтської, серверної частини та бази даних;

– тестування та оцінка якості застосунку;

– підсумування виконаної роботи та огляд результатів.

Методи досліджень для розв’язанні зазначених завдань роботи:

– метод *аналізу* було використано у процесі дослідження мобільних аналогів в контексті створення планів на день та можливостей для кастомізації, а також вибору сучасного засобів розробки;

– методи *аналогій* використовувався при аналізі наявних застосунків із подібним функціоналом задля виявлення популярних тенденцій;

– метод *абстрагування* та *порівняння* використовувалися для аналізу та узагальнення можливостей, загроз, слабких та сильних сторін серед аналогів програмних застосунків для планування справ;

– методи *спостереження*, *пояснень*, *аналізу* було використано під час проєктування архітектури, моделюванні предметної області та розробки мобільного застосунку, як засоби, що допомагають ґрунтовно охарактеризувати та дослідити той чи інший аспект програмної системи.

Результати роботи були апробовані у чотирьох наукових публікаціях авторки [1-4] (додаток Б).

1 АНАЛІЗ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Аналіз особливостей розробки

Категоризація та розподіл пунктів плану за пріоритетами допомагає оптимізувати використання свого часу та побудувати структуру на день, щоб не тільки тримати під контролем свої обов'язки і звички, але й знаходити час на відпочинок та відновлення енергії. Особливо корисними такі інструменти будуть для тих, хто швидко перенавантажується від великого та нечіткого обсягу справ, щоб позбавитися надмірного стресу та зберегти ефективність протягом дня. Для подолання складності на початку роботи з подібними застосунками використовуються готові шаблони з переліком стандартних звичок, з якими простіше зробити крок до будування здорової рутини та виконати своє перше завдання.

Дослідження показує: формування корисних звичок займає від декількох тижнів до декількох місяців, що може стати суттєвою перешкодою для людей, які страждають від пригніченого стану та проблем з мотивацією, особливо поширених за сучасних реалій [5]. Для підтримання зацікавленості користувачів доцільним може бути включення в застосунок ігрового елемента, а саме концепції, популярної та знайомої як дітям, так і дорослим гри «тамагочі». Такий віртуальний вихованець з різними стадіями розвитку зробить дотримання плану на день приємнішою справою для багатьох, а можливість побачити результати за свою продуктивність збільшить шанс утримати увагу користувачів та стимулювати бажання продовжувати. А в часи скрутного морального стану, коли навіть найлегші справи можуть здаватися надто складними, такий застосунок покаже користувачам, що піклування про себе (випита склянка води чи прибрана кімната) – це теж важливі щоденні досягнення. Завдання різних категорій та пріоритетів матимуть однаковий вплив на прогрес розвитку вихованця, адже на шляху до формування здорових звичок та бажаної щоденної рутини цінуються і маленькі, і великі перемоги.

Застосунок персонального планувальника справ має свою специфіку, яку потрібно враховувати при його розробці. Одними з найголовніших складових при створенні подібного програмного продукту є:

– *інтерфейс користувача:* розроблення інтуїтивно зрозумілого та простого у використанні інструментарію для забезпечення максимального комфорту користувачів;

– *механізм вирощування об'єкта:* потрібна продумана система співвідношення відсотка виконаних завдань до прогресу та рівня здоров'я об'єкта, наявність декількох стадій його розвитку, щоб користувач міг відчувати наслідки та задоволення від щоденного виконання своїх цілей, знаходити мотивацію продовжувати користування застосунком. Запобігання ситуацій, коли користувач не дотримується свого плану, через зниження показників здоров'я об'єкта, повернення на попередню стадію розвитку тощо;

– *планування*: можливість додавання, редагування, видалення завдань та відображення їх на екрані, чек-бокси для позначення виконаних пунктів та підрахування, скільки ще залишилося активних завдань;

– *оповіщення*: можливість вказати дату та час для кожної справи, щоб користувач отримував нагадування та не забував про важливі завдання протягом дня;

– *класифікація*: вказання пріоритету завдань та розподілення їх на категорії, за якими користувач зможе здійснити їхнє сортування та фільтрацію, що допоможе ефективніше концентруватися на найважливіших пунктах;

– *список шаблони*: можливість використання системних шаблонів та збереження власних завдань у базу даних, щоб мінімізувати час, що витрачається на планування й повторно використовувати створені раніше пункти;

– *магазин*: додавання товарів з кастомізації об'єкта та придбання корисних предметів для його зростання, щоб підтримати інтерес користувача в подальшому виконанні справ під час розвитку об'єкта та після того, як він досягнув своєї фінальної стадії;

– *система підняття рівня*: виконання завдань має впливати на прогрес рівня користувача, а кожен здобутий рівень – нагороджуватися ігровою валютою, яку можна витратити на покупки в магазині;

– *авторизація:* створення акаунтів користувачів для збереження їхнього прогресу, завдань та придбаних предметів в інвентарі, реєстрація з можливістю називати об’єкт вирощування для більшої зацікавленості; редагування свого профілю;

– *захист даних:* забезпечення захисту персональної інформації користувачів та бази даних;

– *тестування:* здійснення тестування та оцінки якості застосунку перед його випуском підтвердить працездатність продукту та відповідність вимогам.

1.2 Аналіз наявних аналогів

Аналіз наявних застосунків із подібним функціоналом допомагає виявити популярні тенденції, відмітити переваги та недоліки серед аналогів, збільшити конкурентоспроможність продукту та зрозуміти доцільність його розробки. Потужним інструментом для такого аналізу є SWOT таблиці, що відображають сильні та слабкі сторони, можливості та загрози кожного продукту. Вибраними для аналізу є застосунки Finch, Edem, Habitica, Plant Nanny.

Finch (https://finchcare.com/) – застосунок, в якому користувачі можуть вирощувати пташку шляхом виконання щоденних завдань.

Таблиця 1.1 – SWOT-аналіз застосунку Finch

Сильні сторони	**Слабкі сторони**
– можливість створювати власні завдання; – велика кількість згенерованих шаблонів завдань на різні випадки; – велика кількість матеріалів з самодопомоги; – наявність магазину з прикрасами для об’єкта вирощування та елементами інтер’єру; – система збільшення рівня та нагород; – можливість колекціонування стікерів та локацій; – статистика активності; – можливість задати дату та час для нагадування щодо завдання; – можливість додавання друзів.	– перенасиченість функціоналом; – інтуїтивно незрозуміла навігація між вікнами застосунку; – незрозумілий механізм сортування завдань; – немає вибору об’єкта вирощування (окрім кольору); – обмежений рівень енергії на виконання завдань протягом дня в безкоштовній версії; – створені користувачем завдання видаляються після виконання; – немає опції сортування/фільтрації; – не приймаються заходи, якщо користувач не виконує завдання.
Можливості	**Загрози**
– зростання чисельності аудиторії	– немає прив’язки акаунтів до пошти або номеру телефону, можлива втрата даних.

Finch має великий набір інструментів для створення плану на день та можливостей для кастомізації, проте основною слабкою стороною є перенасиченість функціоналом та суттєва кількість відволікаючих елементів, що сильно ускладнює фокусування уваги користувачів на виконанні їхніх завдань протягом дня.

Edem – це застосунок, в якому процес вирощення квітки залежить від процента виконаних користувачем завдань.

Таблиця 1.2 – SWOT-аналіз застосунку Edem

Сильні сторони	**Слабкі сторони**
– є системні рекомендації щодо цілей на день та пояснення, чому важливо їх дотримуватися; – вибір музики в застосунку; – можливість створювати власні завдання; – опція нагадування про завдання; – можливість запустити таймер на виконання завдання; – система зменшення прогресу об’єкта вирощування, якщо користувач не виконує завдання.	– немає можливості для кастомізації об’єкта вирощування в безкоштовній версії; – надто повільний прогрес вирощування; – немає сортування/фільтрації списку завдань; – лише для платформи iOS; – немає нагород за виконані завдання
Можливості	**Загрози**
– зростання аудиторії	– немає системи авторизації.

Edem має простий та інтуїтивно зрозумілий інтерфейс, рослина «в'яне», якщо за день завдання не виконуються, проте користувач довго не бачить результату своїх дій після старанного виконання завдань, процент прогресу зростає надто повільно, що може демотивувати.

Habitica (https://habitica.com/) – це застосунок, в якому користувач створює свій аватар та збирає валюту на його амуніцію через виконання щоденних завдань та дотримання звичок. Habitica має великий асортимент товарів в магазині та інструменти для створення завдань і звичок різних категорій та складності, проте суттєвими слабкими сторонами є надто суворі наслідки за втрату здоров’я та несправний механізм пониження його шкали. Також застосунок потребує, щоб користувачі власноруч зазначали невиконані звички, за які вони втратять показник здоров’я, що в деяких випадках може демотивувати.

Таблиця 1.3 – SWOT-аналіз застосунку Habitica

Сильні сторони	**Слабкі сторони**
– є система авторизації користувачів, підтримка входу через Google акаунт та AppleID; – багато способів кастомізації об’єкта; – збільшення рівня відкриває нові можливості об’єкта; – можливість окремо додавати звички (з розділенням на хороші і погані) та щоденні завдання; – категоризація завдань за складністю та тегами, опція пошуку та нагадування про завдання; – магазин з корисними для розвитку аватару предметами та аксесуарами; – відображається шкала здоров’я та досвіду.	– немає стадій розвитку об'єкта; – інтерфейс застосунку повільно реагує на дії користувача; – система віднімання здоров’я об’єкта іноді працює некоректно і віднімає більше, ніж потрібно; – користувач повинен визначити звичку як «не виконану» власноруч, щоб втратити здоров’я; – при повній втраті здоров’я користувач втрачає весь інвентар та валюту.
Можливості	**Загрози**
– зростання аудиторії	– технічні проблеми на сервері

Plant Nanny (https://sparkful.app/plant-nanny) – це застосунок, в якому користувач вирощує вибрану з переліку рослину шляхом дотримання своєї денної норми, що визначається автоматично в залежності від ваги. Попри те, що в ньому немає можливості самостійно створювати завдання, редагувати чи видаляти завдання, цей застосунок варто було розглянути через продуману систему вибору та кастомізації вирощуваних об’єктів.

Таблиця 1.4 – SWOT-аналіз застосунку Plant Nanny

Сильні сторони	**Слабкі сторони**
– авторизація через Google, Facebook, AppleID; – є вибір з декількох об’єктів вирощування, перед вибором можна переглянути кожну стадію розвитку; – є система підвищення рівня і нагород; – є магазин з великим вибором аксесуарів; – опція додати на екран телефону в якості віджету; – є календар, де вказано, скільки днів поспіль виконуються системні завдання; – можливість вирощувати декілька об’єктів одночасно; – перехід об’єкта в стадію «зав’яле», якщо користувач не дотримує норму.	– немає опції самостійно створювати завдання; – немає опції нагадувань; – системні завдання націлені лише на дотримання денної норми води; – немає визначеної шкали здоров’я об’єкта; – велика кількість реклами; – в безкоштовній версії немає можливості редагувати денну норму користувача.
Можливості	**Загрози**
– зростання аудиторії	– технічні проблеми на сервері

Аналіз популярних в PlayMarket та AppStore застосунків виявив тенденції та загальні характеристики, наприклад: функції додавання, редагування завдань,

нагадування із зазначенням дати та часу, шкала прогресу, система підвищення рівня, нагород, наслідки для користувача за невиконані завдання, магазин з предметами кастомізації, процес реєстрації. Цікавим функціоналом є категоризація та фільтрація завдань, системні рекомендації, зміна стадій об’єкту вирощування тощо. Можна побачити, що ці застосунки не передбачають збереження створених користувачем завдань у шаблони для можливості їхнього повторного використання, а також більшість з них не має варіантів вибору між різними видами об’єктів вирощування, отже ці пункти можуть стати унікальними для створюваного продукту. Тим самим аналіз аналогів допоміг визначити функціональні вимоги створюваного застосунку.

1.3 Розробка функціональних вимог

Формування функціональних вимог є одним з найважливіших етапів для розробки будь-якої інформаційної системи. Вони відображають повний перелік функціоналу та варіанти взаємодії системи з користувачем, на основі яких відбуватиметься подальша розробка застосунку. Правильно створені вимоги повинні бути викладені чітко, доступно, вичерпно і лаконічно [6]. Функціональні вимоги для застосунку персонального планувальника справ:

– Авторизація користувача:

- можливість увійти до облікового запису з email и паролем
- можливість зареєструватися з указанням нікнейму, пошти та паролю користувача на першій сторінці реєстрації, вибором ім’я та типу об’єкта вирощування на другій сторінці;

– Сторінка списку завдань користувача:

- crud-операції: створення, перегляд, редагування, видалення кожного елементу зі списку завдань;
- при натисканні кнопки «додати» можливість вибрати між створенням нового завдання чи використанням готового шаблону;

- при створенні нового завдання можливість вказати назву та опис, вибрати зі списку пріоритет та категорію, задати час для оповіщення, зберегти як шаблон для подальшого використання;
- розподілення шаблонів на збережені користувачем та системні;
- можливість редагування всіх полів шаблону;
- можливість фільтрування завдань за пріоритетом та категорією;
- можливість позначити завдання як виконані або невиконані;

– Об’єкт вирощування:

- можливість редагування ім’я об’єкта;
- можливість перегляду шкали здоров’я та стадії розвитку об’єкта, зміна стадії розвитку в залежності від прогресу виконання завдань;

– Рівень користувача:

- збільшення прогресу рівню на 10% за кожне виконане завдання;
- отримання внутрішньої валюти в якості нагороди за кожний досягнутий рівень;

– Сторінка магазину:

- можливість перегляду загального переліку товарів;
- можливість перегляду інформації про окремо вибраний товар окремо;
- можливість купувати вибраний товар в заданій кількості та переглядати в інвентарі.

– Сторінка налаштування:

- можливість зміни аватару та нікнейму користувача;
- можливість зміни ім’я об’єкту вирощування;
- можливість вказання та підтвердження нового паролю від облікового запису;
- можливість виходу з облікового запису;
- можливість здійснення зворотнього зв’язку.

Сформульований перелік вимог визначає траєкторію процесу створення застосунку, подальша розробка буде спрямована на втілення функціоналу, що задовольнить вказані потреби користувача.

1.4 Вибір засобів розробки застосунку

Для оптимізації процесів розробки застосунку та угрупуванні різних стадій життєвого циклу програмного продукту корисно використовувати платформи, що надають інструменти DevOps. Популярним на ринку є рішення від компанії Microsoft – платформа Azure, що надає послуги SaaS (програмне забезпечення як послуга) і втілює методики DevOps для ефективного створення та обслуговування програмних продуктів. Згідно з даними Azure Active Directory [7], станом на 2022 рік було зареєстровано понад 700 мільйонів Azure користувачів.

Вибір багатьох компаній зупиняється на Azure DevOps саме через наявність вичерпного набору інструментів для роботи з застосунками на протязі їхнього повного життєвого циклу та можливості інтегрування як з іншими продуктами Microsoft (Visual Studio, SQL Server Management Studio, Microsoft Teams, ASP.NET, Azure сервіси тощо) так і зі сторонніми сервісами (GitHub, Trello, Jenkins тощо) [8]. Працювати з Azure DevOps можна на різних операційних системах, мовах програмування та навіть хмарних сервісах (Amazon Web Services (AWS) та Google Cloud Platform (GCP)). Гнучкі умови користування та надійна система безпеки роблять платформу особливо зручною для продуктових команд з online режимом роботи.

Інструменти у складі Azure DevOps допомагають організовувати робочій процес, планувати, розподіляти завдання між командою, розробляти, тестувати та запускати застосунок. На рис. 1.2 наведено ключові інструменти Azure DevOps, які використовувались у роботі: Azure Boards, Azure Repos, Azure Pipelines, Azure Test Plans і Azure Artifacts.

Azure Boards містить інструменти для планування з використанням підходу Agile та керування робочим процесом, а також зручну візуалізацію переліку завдань команди.

Azure Repos дозволяє працювати з необмеженими безкоштовними приватними Git-репозиторіями, використовувати систему контроля версій спільно з Visual Studio для оновлення та збереження коду, відслідковування змін в проєкті, вирішення конфліктів, проведення code review тощо.

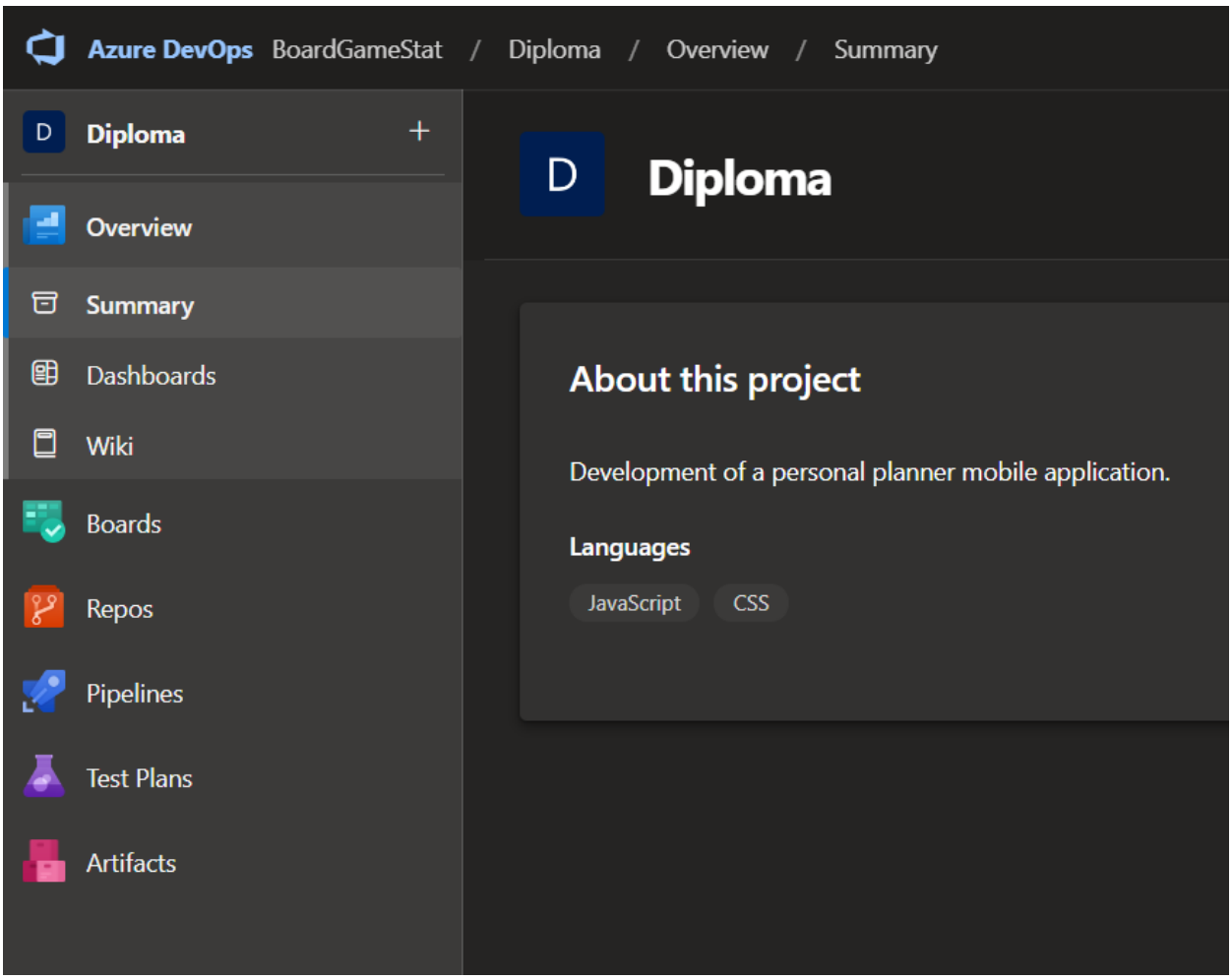

Рисунок 1.1 – Структура інструментів Azure DevOps

Azure Pipelines втілює практику CI/CD (Continuous Integration/Continuous Deployment), що означає неперервні інтеграцію та розгортання застосунку. Цей підхід відповідає за автоматичні побудову, тестування та запуск нової версії програмного коду, чим забезпечується швидкий та безпечний цикл розробки.

Azure Test Plans – комплексне рішення для впровадження різних технік для ручного і автоматичного тестування проєкту, відстеження результатів тестів та ідентифікування багів, формування звітів на основі отриманих даних.

Azure Artifacts надає можливість спільного використання потрібних для застосунку пакетів (Maven, npm, NuGet, Python тощо) із загальних та приватних джерел, а також їхнє безпечне зберігання та інтеграцію з Azure Pipelines [9].

Отже, Azure DevOps оснащений повним функціоналом, необхідним для оптимізації робочого процесу та керування циклом розробки створюваного у цій

роботі програмного продукту: від створення плану до тестування і запуску готового застосунку.

1.4.1 Вибір засобів розробки бази даних

Azure DevOps підтримує під'єднання до проєкту системи керування базами даних SQL Server, створеною компанією Microsoft. За даними статистики 2023 року, наведеною на рис. 1.3, SQL Server входить у топ найпопулярніших інструментів для роботи з реляційними базами даних [10].

Microsoft SQL Server має широкий спектр різноманітних інструментів, спрямованих на процеси обробки та керування великим обсягом даних, забезпечення їхнього безпечного зберігання, аналізу, а також бізнес-аналітики з впровадженням штучного інтелекту [11]. Використання доступне для різних операційних систем (Windows, Linux, MacOS).

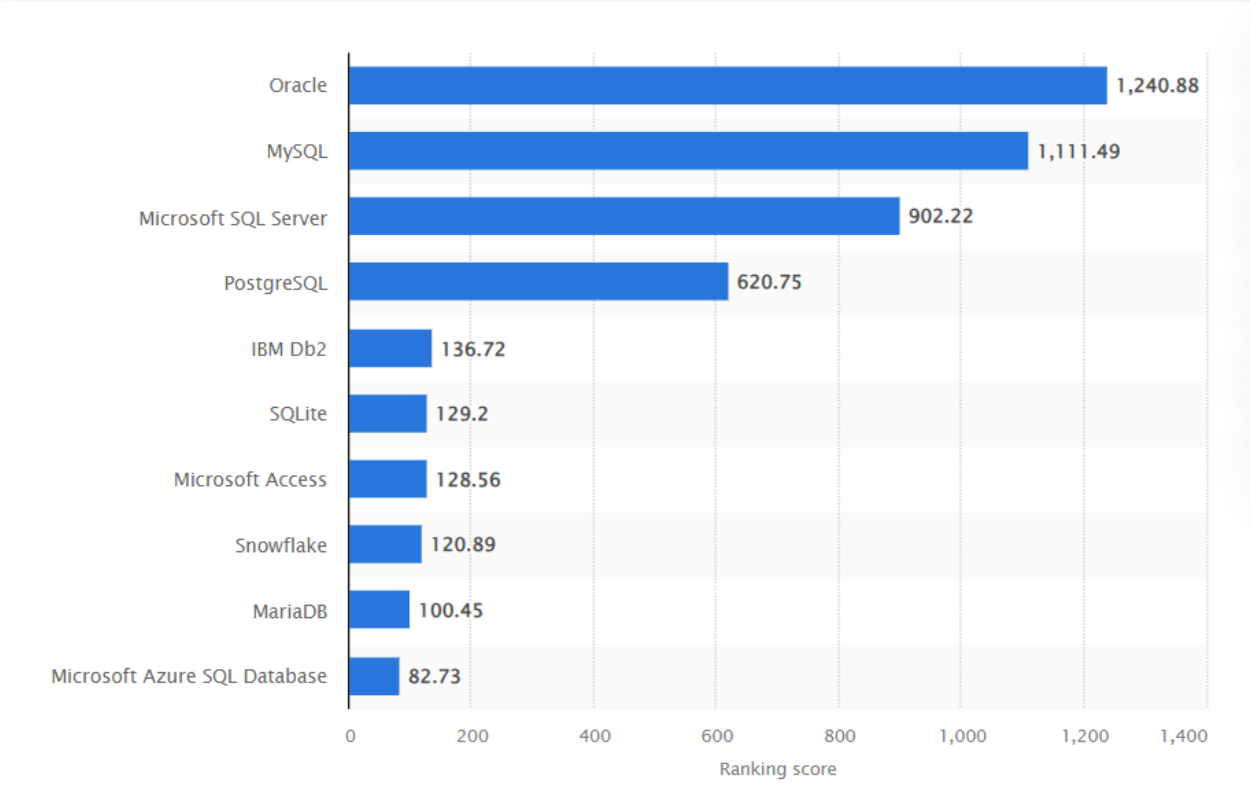

Рисунок 1.2 – Статистика використання СКБД

SQL Server може використовувати процедурну мову T-SQL (Transact-SQL), що є розширенням стандартного набору інструкцій мови SQL, розроблене компанією Microsoft. T-SQL надає можливості використовувати додаткові

оператори, системні функції для обробки даних рядків, вирази застосовувати змінні тощо [12].

SQL Server Management Studio (SSMS) – це інтегроване середовище для керування будь-якою SQL інфраструктурою. Його графічний інтерфейс дозволяє налаштовувати та адмініструвати компоненти Microsoft SQL Server [13]. SSMS має велику кількість функцій, серед яких:

– створення нових баз даних, користувачів та ролей чи підключення до існуючих екземплярів SQL Server;

– створення, редагування та видалення об’єктів SQL бази даних;

– перегляд записів таблиць бази даних;

– генерація та здійснення запитів до таблиць бази даних, використання складних SQL-запитів за допомогою візуального конструктору запитів;

– редактор коду з підказками щодо синтаксису та авторедагуванням;

– доступ до журналів помилок і системних баз даних SQL Server;

– використання Integration Services для керування пакетами, їхнього оновлення та переміщення;

– візуалізація даних за допомогою діаграм та графіків;

– контроль версій та керування резервними копіями SQL Server [14].

Інтуїтивно зрозумілий та зручний графічний інтерфейс в поєднанні з потужним функціоналом робить SQL Server Management Studio важливим інструментом для адміністрування системи керування баз даних.

1.4.2 Вибір засобів розробки backend частини

ASP.NET Core – кросплатформний високопродуктивний open-source фреймворк від компанії Microsoft, що використовує об’єктно-орієнтовану мову програмування C#. За даними опитування 2023 року, наведеними на рис. 1.4, ASP.NET Core входить в топ-10 фреймворків та технологій, що вибирають для розробки IT-фахівці [15].

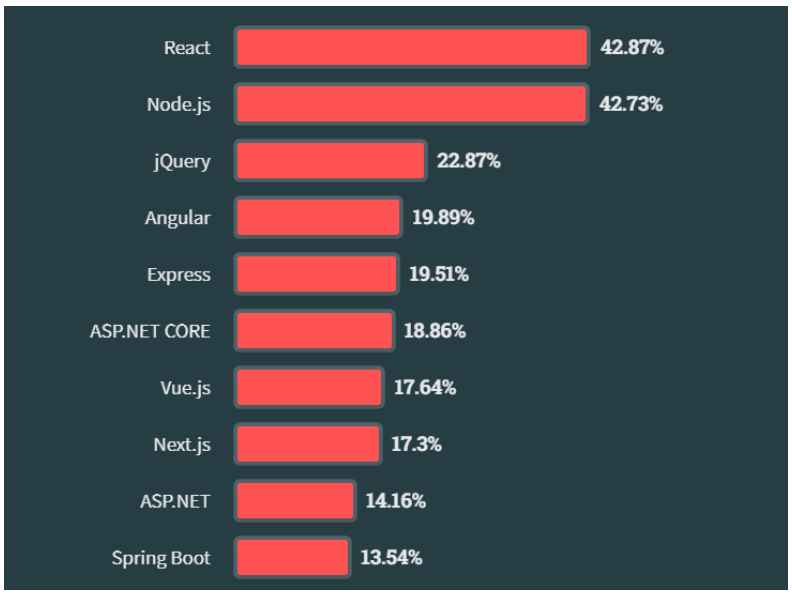

Рисунок 1.3 – Статистика використання фреймворків

За допомогою ASP.NET Core створюються вебзастосунки будь-яких масштабів, серверні застосунки та rest API сервіси (впровадження HTTP-запитів для отримання та використання серверних даних), які можуть виступати в якості backend частини для веб та мобільних застосунків.

Деякі з переваг використання ASP.NET Core для мобільної розробки [16]:

– єдине рішення для Android та iOS платформ з можливістю розробки на Windows, Linux та MacOS;
– можливість використання з поширеними фреймворками для розробки мобільних застосунків: React Native, Flutter, Cordova, Ionic тощо;
– відкритий вихідний код та велика кількість
– реактивність та висока продуктивність з великим обсягом трафіку, можливість балансування навантаження та кешування для роботи з широкою базою користувачів;
– інструменти для забезпечення безпеки персональних даних користувачів (шифрування даних, автентифікація тощо);
– обробка асинхронних API у великих масштабах без очікування.

Використання ASP.NET Core фреймворка забезпечує кросплатформність, надійний рівень безпеки, оптимізацію продуктивності, швидкість роботи, великий

перелік інструментів та функцій для ефективного створення програмного застосунку за короткотривалий період.

1.4.3 Вибір засобів розробки frontend частини

Capacitor.js – це кросплатформенний фреймворк від компанії Ionic для розробки гібридних мобільних застосунків, що дозволяє використовувати веб-технології (HTML, CSS, JavaScript) для створення мобільних застосунків на платформах iOS та Android. Capacitor.js надає API використання функцій мобільного пристрою (камера, геолокація, файлові системи, сповіщення тощо), інтегрується з популярними фреймворками зі створення веб-застосунків (React, Angular та ін.), має велику кількість офіційних плагінів та дозволяє використання сторонніх, а створені мобільні застосунки можуть бути опубліковані в AppStore та PlayMarket. Capacitor.js встановлюється як залежність у React проєкті, надає доступ до нативних API через JavaScript, збирає і генерує проєкти для кожної цільової платформи [17].

Оскільки при створенні вибраного програмного продукту не потрібна глибока інтеграція з нативними функціями мобільних пристроїв на рівні операційної системи, замість опановування широкого інструментарія фреймворка React Native або вивчення спеціалізованої мови програмування (Swift, Kotlin, Objective-C тощо) є можливість використати знайомий стек технологій від React. Комбінація Capacitor.js та React дозволяє швидко створювати мобільні застосунки з доступом до базових можливостей мобільних пристроїв лише за допомогою засобів веб-розробки.

1.5 Маркетингові інструменти в створенні та просуванні програмного продукту

Згідно зі статистикою Startup Genome [18], понад 70% провалів у втіленні створених продуктів на ринок трапляються через неправильно підібрані маркетингові інструменти, відсутність визначеної стратегії та фінансові проблеми.

Маркетинг продуктів потребує комплексного підходу і ретельного довгострокового планування [4].

Першими етапами є визначення цільової аудиторії та бізнес цілей, аналіз ринку на продукти аналоги та попит на товар, а також технології і засоби зв'язку, що використовують конкуренти. Цільова аудиторія персонального планувальника справ з ігровим елементом може бути досить широкою та різною, проте в якості «ідеального користувача» можна вибрати категорію людей від 18 до 25 років. Саме їм потрібна допомога з концентрацією та плануванням робочого процесу протягом дня та нейровідмінних людей в тій самій віковій категорії, яким необхідно структурувати свій день та мати додатковий стимулюючий увагу об'єкт. Напрям початкової бізнес-цілі найвлучніше відображатиме підхід Awareness & Loyalty, спрямований на підвищення рівня обізнаності про продукт та збільшення кількості лояльної аудиторії.

Проведений аналіз маркетингу у соціальних мережах (Social Media Marketing, SMM) щодо просування продуктів-аналогів, розглянутих у підрозд. 1.3 сформовано у табл. 1.5.

Таблиця 1.5 – Аналіз засобів зв'язку продуктів-аналогів

Інструменти	**Finch**	**Edem**	**Habitica**	**Plant Nanny**
Facebook	+	–	–	+
Instagram	+	+	+	+
Tiktok	+	–	–	–
Вебсайт	+	–	+	+
Інше	Discord та Reddit	Gmail для відгуків	Twitter, Google Forms для відгуків	Twitter

Отже, можна зробити висновок, що доцільними заходами для подальшого просування застосунку буде створення відповідного вебсайту та Instagram-сторінки. Контент-стратегія в Instagram має бути зосередженою на візуальну складову та розважальний контент. Оскільки цільова аудиторія застосунку – це молоде покоління, доречним може бути поширення застосунку через платформу Tiktok, що дозволить відрізнятися від конкурентів. Специфіка просування відео в цій соціальній мережі потребує врахування тривалості роликів, частоти викладання

та активності, добору хештегів та музики, а також легкого та ненав’язливого формату контенту.

При створенні вебсайту продукту має враховуватися SEO (Search Engine Optimization) – його оптимізація під пошукові запити користувача. Оскільки переважна більшість запитів оброблюється саме в Google, треба орієнтуватися на критерії просування результатів пошуку перших сторінок, саме для цієї пошукової системи, щоб користувачі з більшою вірогідністю побачили продукт, перейшли на сайт та виконали цільову дію (завантажили мобільний застосунок) [4]. Алгоритм Google враховує якість контенту, ключові слова, мета-теги, заголовки та підзаголовки, оптимізований медіа-контент, відгуки, посилання вебсторінок, а також мову та локацію користувача [19]. У разі грамотного використання SEO можливо суттєво збільшити залученість користувачів та зайняти вигідну позицію на ринку.

Співпраця з іншими брендами та компаніями є дієвим методом у збільшенні охоплення цільової аудиторії, залучення нових користувачів та підвищенні довіри до застосунку, а вказання партнерами прямих посилань на продукт також впливатиме на SEO. Прикладом для розроблюваного програмного продукту може бути колаборація з популярними контент-мейкерами аудиторії: організація обмежених за часом подій (івентів) з підвищеним обсягом нагород за виконання завдань у застосунку, поява лімітованих об'єктів вирощування або їхніх аксесуарів від партнерів тощо [4].

Ще одним важливим пунктом є визначення способу монетизації продукту – його бізнес-моделі. Всі розглянуті у підрозд. 1.2 конкуренти працюють за моделлю Freemium, яка нині є найпопулярнішою. Вона поєднує в собі безкоштовне користування базовими функціями застосунку та преміум-підписку, що надає платний доступ з розширеним переліком функцій та переваг. Цінова політика таких підписок суттєво залежить від доступних функцій і змінюється від $9.99 на рік (Edem), $47.99 (Habitica) до $79,99 (Finch та Plant Nanny) з можливістю спробувати 14-денну безкоштовну преміум-версію. Freemium модель має потенціал

використання і в застосунку з планування справ після розробки більш розвиненого функціоналу.

1.6 Висновки до розділу 1

У розділі проведено аналіз особливостей розробки мобільного застосунку персонального планувальника справ. З’ясовано, що він має свою специфіку, яку потрібно враховувати при його розробці. Необхідними складовими при створенні цього програмного продукту є: зручний інтерфейс користувача; продумана система співвідношення відсотка виконаних завдань, щоб користувач міг відчувати наслідки та задоволення від щоденного виконання своїх цілей, знаходити мотивацію продовжувати користування застосунком; можливість додавання, редагування, видалення завдань та відображення їх на екрані; можливість вказати дату та час для кожної справи, отримування нагадувань, щоб не забувати про важливі завдання протягом дня; вказання пріоритету завдань та розподіл їх на категорії, за якими користувач зможе здійснити їхню фільтрацію, що допоможе ефективніше концентруватися на найважливіших пунктах; можливість використання системних шаблонів та збереження власних завдань у базу даних, щоб мінімізувати час, що витрачається на планування й повторно використовувати створені раніше пункти; авторизація користувачів; захист персональної інформації користувачів та бази даних тощо.

Проведений SWOT-аналіз наявних застосунків із подібним функціоналом допоміг виявити популярні тенденції, відмітити переваги та недоліки серед наявних на ринку програмних продуктів аналогів Finch, Edem, Habitica, Plant Nanny. Тим самим аналіз аналогів допоміг визначити функціональні вимоги створюваного застосунку. Саме на основі повного переліку функціоналу та варіантів взаємодії системи з користувачем відбуватиметься подальша розробка застосунку. Сформульований перелік вимог визначив траєкторію процесу створення застосунку.

Вибір засобів розробки застосунку стосувався засобів розробки бази даних, а також розробки backend та frontend частин мобільного застосунку. Так, для

розробки бази даних обґрунтовано вибір Microsoft SQL Server, який має широкий спектр різноманітних інструментів, спрямованих на процеси обробки та керування великими обсягами даних, забезпечення їхнього безпечного зберігання. Для розробки backend частини вибрано фреймворк ASP.NET Core, який забезпечує кросплатформність, надійний рівень безпеки, оптимізацію продуктивності, швидкість роботи, має великий перелік інструментів та функцій для ефективного створення програмного застосунку за короткотривалий період. Для розробки frontend частини використано фреймворки React та Capacitor.js для швидкого та ефективного створення гібридного мобільного застосунку засобами веб-розробки. Платформа Azure DevOps буде використовуватись для оптимізації робочого процесу та керування циклом розробки створюваного у цій роботі програмного продукту: від створення плану до тестування і запуску готового застосунку.

Крім того, у розділі розглянуто можливі маркетингові інструменти в створенні та просуванні створюваного програмного продукту.

2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1 Архітектура програмного застосунку

Трирівнева архітектура – це концепція проєктування програмного забезпечення, яка поділяє функціонал програми на три рівні: подання даних, бізнес-логіки (BLL) та доступу до даних (DAL). Таке рішення дозволяє підтримувати чітку структуру програми, розроблювати, оновлювати, тестувати та підтримувати модулі одночасно та незалежно один від одного, покращувати рівень захисту програмного застосунку.

1) Рівень подання даних (Presentation Layer)

Перший рівень відповідає за взаємодію з користувачем, його головна функція – забезпечити зручний та ефективний інтерфейс. Рівень подання даних не повинен знаходитися в прямому контакті з базою даних, може лише надсилати запити на сервер та отримувати відповідь. Він містить у собі контролери та подання, логіка в яких є найпростішою.

2) Рівень бізнес-логіки (Business Logic Layer)

Другий рівень відповідає за бізнес-логіку, що потрібна для забезпечення основних функцій застосунку (обробка замовлень, обчислення фінансових показників, управління користувачами, валідація даних тощо). Рівень бізнес-логіки не залежить від конкретних інтерфейсу чи бази даних, він може оброблювати інформацію, що надходить з рівня подання даних, виконувати складні операції, повертати дані, отримані з DAL.

3) Рівень доступу до даних (Data Access Layer)

Третій рівень забезпечує доступ до даних та взаємодію з базою даних. Він відповідає за збереження, отримання, оновлення та видалення даних з бази даних. Цей рівень створюється для ізоляції бізнес-логіки від деталей роботи з даними та дозволяє змінювати джерела даних без впливу на решту системи. Рівень доступу до даних може отримувати запити від BLL, повертати на них відповіді, а також надсилати запити до системи керування базами даних [20].

Трирівневий архітектурний патерн розділяє розміщення бази даних, інтерфейсу та серверу застосунку на рівні, в той час як MVC поділяє рівень на три компонента.

Патерн MVC (Model-View-Controller) – це шаблон проєктування архітектури програмного застосунку, який виділяє компоненти: Модель (Model), Подання (View) та Контролер (Controller). Принцип роботи шаблону наведено на рис. 2.1.

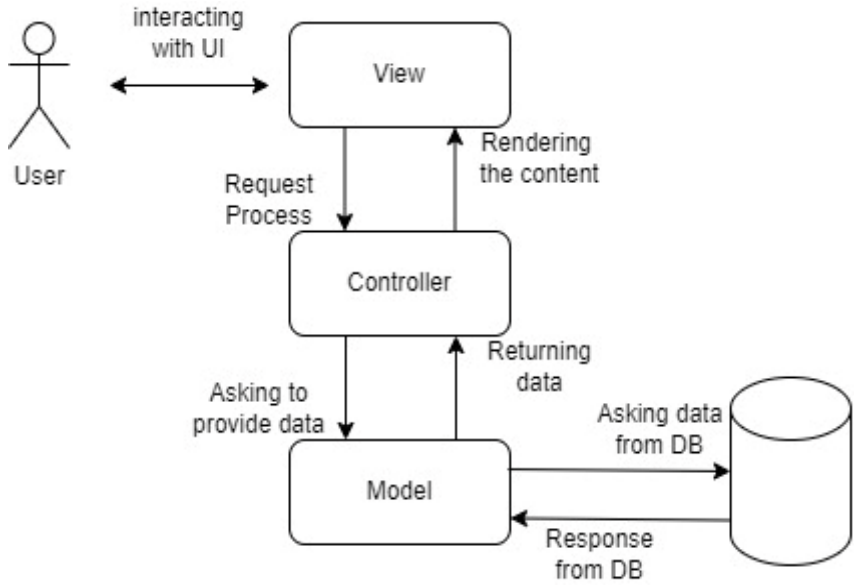

Рисунок 2.1 – Принцип роботи архітектури MVC

Модель (Model) забезпечує доступ до даних, методи роботи з ними (обробка, валідація даних, здійснення запитів в базу даних) та управління бізнес-логікою програми за відсутності безпосередньої взаємодії з користувачем. Не залежить від інших компонентів MVC архітектури, одна модель може використовуватися в декількох поданнях та контролерах.

Подання (View) відповідає за відображення даних, отриманих від моделі у потрібному вигляді та забезпечення взаємодії користувача з застосунком за допомогою графічного інтерфейсу. Подання не оброблює отримані дані, а лише надсилає відповідь користувача в контролер.

Контролер (Controller) є елементом зв’язку між моделлю та поданням. Він отримує запити, викликає методи моделі для обробки відповіді, оновлює подання для динамічного відображення результатів [21].

Застосування архітектурних патернів дозволяє отримати зручну структуру програми, де кожен модуль програми виконує власну функцію та може змінюватися незалежно від інших, що підвищує гнучкість і масштабованість застосунку.

2.2 Проєктування бази даних

Реляційні бази даних – це бази даних, систематизовані та упорядковані у вигляді рядків і стовпців, де кожен стовпець має свої атрибути, а зв'язки між даними встановлюються за допомогою ключів. Реляційні бази даних створюються з використанням SQL і мають систему керування (напр. Microsoft SQL Server). Для створення бази даних потрібно визначити сутності, що мають бути додані, а також їхні атрибути і зв’язки.

Сутність (Entity) – це екземпляр об’єкта, що зберігається в таблиці, має визначені характеристики та може бути однозначно ідентифікованим.

Атрибут (Attribute) – це властивість сутності, яка характеризує її. Кожен атрибут має свою назву та тип даних, що він зберігає.

Ключ (Key) – це атрибут, який використовується для однозначної ідентифікації сутності. Кожна сутність має свій первинний ключ (Primary Key), який дозволяє унікально ідентифікувати її у базі даних. Для створення зв'язків між таблицями використовується зовнішній ключ (Foreign Key), що посилається з одної таблиці на первинний ключ іншої. Ключі в базі даних застосовуються для підтримки цілісності даних.

Зв’язок (Relationship) – це залежність між двома сутностями, що допомагають об’єднувати дані декількох таблиць для виконання складних запитів. Зв’язок може бути один до одного (1:1), один до багатьох (1:N) або багато до багатьох (N:M) [22].

Для збільшення ефективності роботи з базами даних та забезпечення цілісності даних застосовують процес нормалізації бази даних. Існує шість нормальних форм, але в більшості випадків доведення до третьої з них вистачає для

усунення більшості аномалій таблиць та дублювання даних. Щоб досягти третьої нормальної форми таблиць баз даних, потрібно, щоб виконувалися такі умови:

- кожен атрибут таблиці повинен бути атомарним (1 НФ);
- таблиця знаходиться в 1 НФ, а кожен атрибут, що не є ключем таблиці, повинен повністю залежати від кожного потенційного ключа (2 НФ);
- таблиця знаходиться в 2 НФ, не має транзитивних функціональних залежностей (3НФ).

Визначені сутності для предметної області персонального планувальника справ: користувач, завдання користувача, об’єкт вирощування, товари магазину, предмети інвентаря.

Визначимо атрибути кожної з перелічених сутностей:

1. Користувач:
   - ідентифікатор (id);
   - юзернейм користувача;
   - адреса електронної пошти;
   - пароль;
   - посилання на аватар;
   - поточний рівень;
   - поточний прогрес рівня;
   - кількість валюти застосунку;
2. Завдання користувача:
   - ідентифікатор (id);
   - назва завдання;
   - опис завдання;
   - пріоритет;
   - категорія;
   - статус;
   - дата та час;
   - ідентифікаційний номер користувача;

3. Об’єкт вирощення:
   - ідентифікатор (id);
   - ім’я;
   - вид;
   - стадія;
   - прогрес;
   - посилання на зображення;
   - ідентифікаційний номер користувача;
4. Товар магазину
   - ідентифікатор (id);
   - назва товару;
   - опис товару;
   - ціна;
   - посилання на зображення;
5. Предмет інвентаря
   - ідентифікатор (id);
   - ідентифікаційний номер користувача;
   - ідентифікаційний номер товару магазину;
   - кількість;

Тип зв’язку між сутностями бази даних:

Зв’язок Користувач – Завдання: один до багатьох, тому що один користувач може створювати багато завдань.

Зв’язок Користувач – Об’єкт вирощування: один до багатьох, користувач може мати декілька об’єктів вирощування.

Зв’язок Товар магазину – Предмет інвентаря: один до багатьох, один товар може бути в багатьох інвентарях.

Зв’язок Користувач – Предмет інвентаря: один до багатьох, користувач може мати декілька предметів в інвентарі.

Після завершення нормалізації, таблиці бази даних мають вигляд, поданий в табл.2.1 – 2.6.

Таблиця 2.1. – Користувач (ApplicationUser)

Атрибут	Тип даних атрибуту	Опис
id	int, PK	ідентифікатор цієї таблиці, її первинний ключ
UserName	nvarchar	ім’я користувача
Email	nvarchar	електронна адреса користувача
Password	nvarchar	пароль користувача
Level	int	поточний рівень користувача
LevelProgress	int	поточний прогрес рівню користувача
AvatarUrl	nvarchar	посилання на зображення аватару користувача
Money	int	кількість внутрішньої валюти користувача

Таблиця 2.2. – Завдання (Assignment)

Атрибут	Тип даних атрибуту	Опис
id	int, PK	ідентифікатор цієї таблиці, її первинний ключ
Name	nvarchar	назва завдання
Description	nvarchar	опис завдання
Priority	int	пріоритет завдання
Status	int	статус завдання
CategoryId	int, FK	ідентифікаційний номер категорії з таблиці AssignmentCategory
IsTemplate	bit	перевірка, чи зберігається створене завдання, як шаблон
Notify	bit	перевірка, чи потрібно ввімкнути оповіщення для цього завдання
StartsAt	datatime	дата і час оповіщення
UserId	int, FK	ідентифікаційний номер користувача, зовнішній ключ

Таблиця 2.3 – Категорія (AssignmentCategory)

Атрибут	Тип даних атрибуту	Опис
id	int, PK	ідентифікатор цієї таблиці, її первинний ключ
Name	nvarchar	назва категорії

Таблиця 2.4 – Об’єкт вирощування (GrowingObject)

Атрибут	Тип даних атрибуту	Опис
id	int, PK	ідентифікатор таблиці GrowingObject, первинний ключ
Name	nvarchar	ім'я об’єкта
Species	nvarchar	вид об’єкта
Stage	int	стадія розвитку об’єкта
Progress	float	шкала здоров’я об’єкта
ImageUrl	nvarchar	масив з посилань на зображення 4 стадій об’єкта
UserId	int, FK	ідентифікаційний номер користувача, зовнішній ключ

Таблиця 2.5 – Товар магазину (ShopItem)

Атрибут	Тип даних атрибуту	Опис
id	int, PK	ідентифікатор таблиці ShopItem, первинний ключ
Name	nvarchar	назва товару
Description	nvarchar	опис товару
Price	int	ціна товару
ImageUrl	nvarchar	посилання на зображення
UserId	int, FK	ідентифікаційний номер користувача, зовнішній ключ

Таблиця 2.6 – Предмет інвентаря (InventoryItem)

Атрибут	Тип даних атрибуту	Опис
id	int, PK	ідентифікатор таблиці InventoryItem, первинний ключ
UserId	int, FK	ідентифікаційний номер користувача, зовнішній ключ
ItemId	int, FK	ідентифікаційний номер товару магазина, зовнішній ключ
Quantity	int	кількість товару

ER-діаграма реляційної бази даних наведена на рис. 2.2.

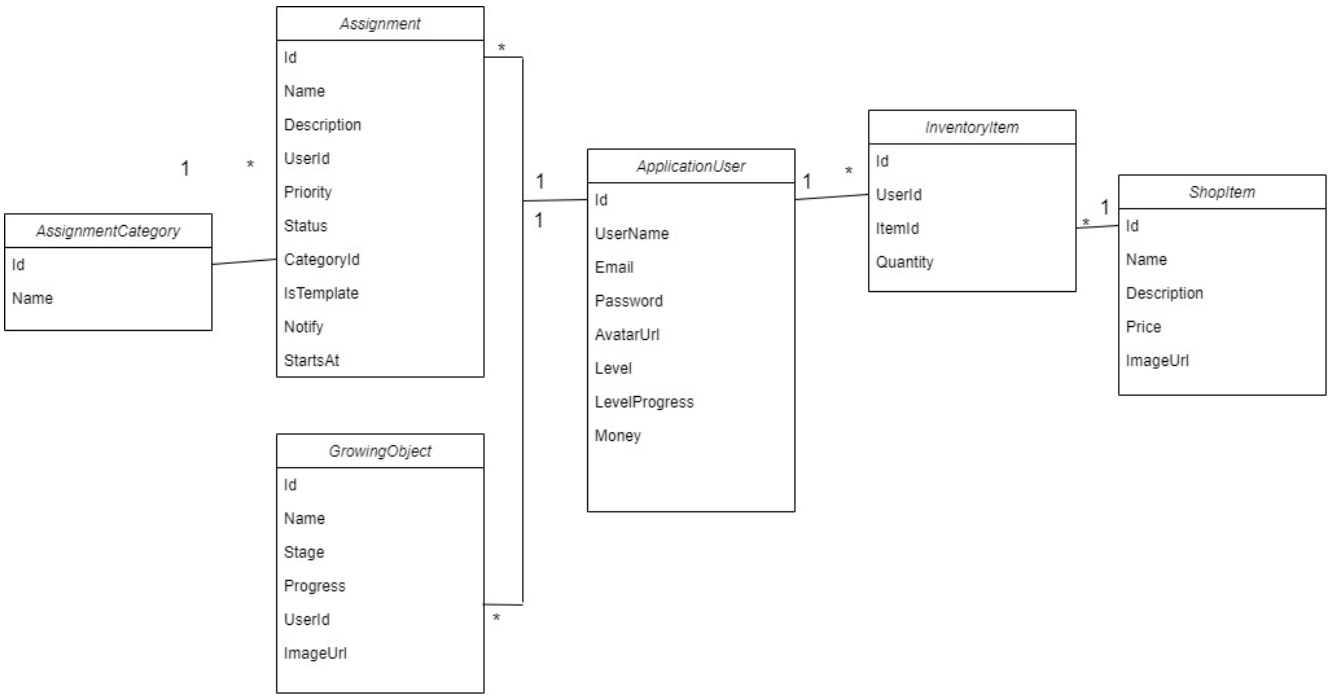

Рисунок 2.2 – ER-діаграма бази даних

2.3 Моделювання програмної системи

UML діаграми – це потужний інструмент, що використовується для візуалізації потреб системи, проєктування її структури та визначення взаємодії між елементами.

Діаграма варіантів використання (Use Case Diagram) описує перелік функціоналу застосунку персонального планувальника справ та взаємодію з користувачем на рис. 2.3. Основні елементи такої діаграми: Актори (Actors), Прецеденти (Use Case), Зв’язки (Relationships). Актором для створеної діаграми є користувач застосунку, прецедентами є функціональні можливості системи, зв’язки є відображенням взаємодії з системою [23]. Зв’язки, що використовувались: асоціація (association, звичайний зв’язок), включення (include, для базового функціоналу), розширення (extend, для додаткового функціоналу).

Рисунок 2.3 – Діаграма варіантів використання функціоналу застосунку

Функціонал створення нового завдання винесено в окрему діаграму варіантів використання, показану на рис. 2.4. для більш зручного перегляду.

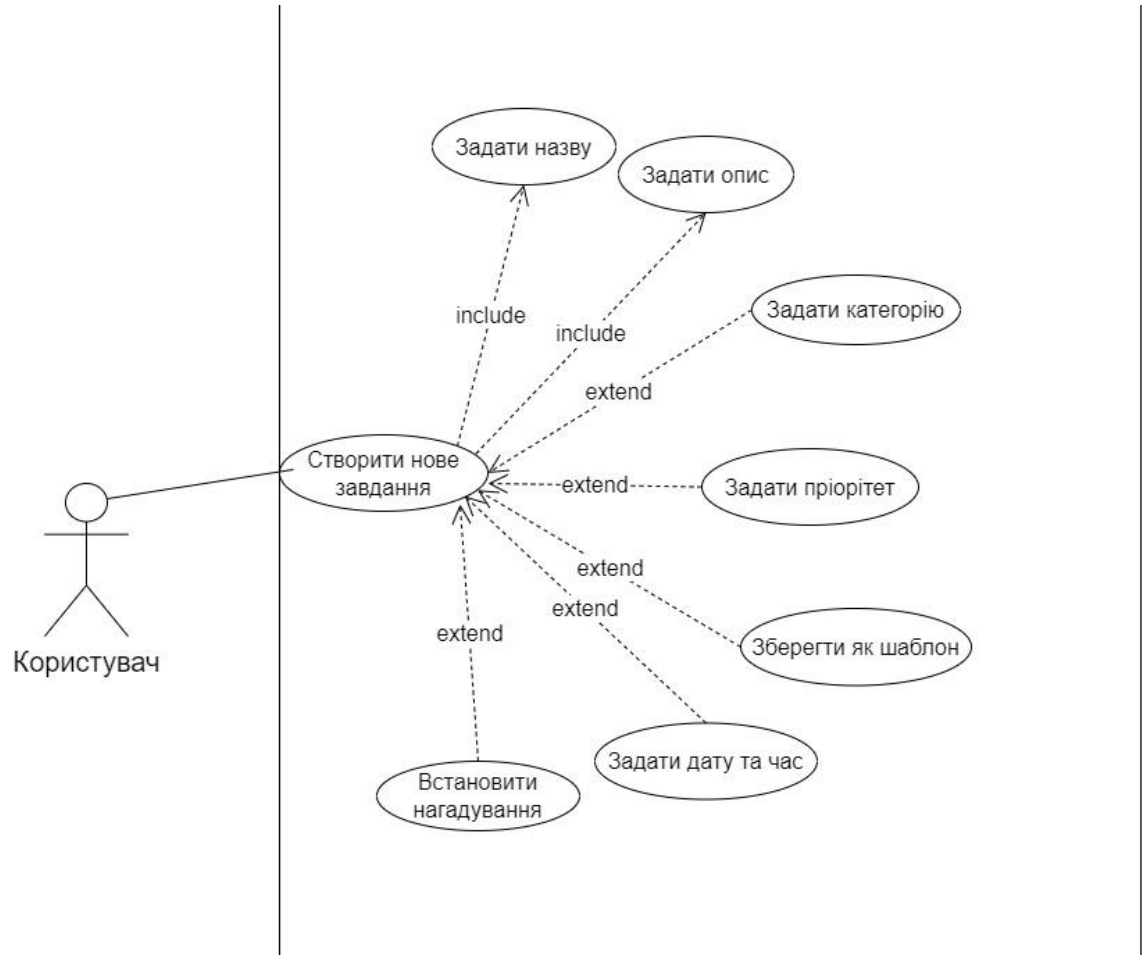

Рисунок 2.4 – Діаграма варіантів використання для створення завдання

2.4 Висновки до розділу 2

У розділі розглянуто архітектурні патерни, що будуть використовуватися для розробки програмного продукту. Структура проєкту складатиметься з трьох рівнів, що відокремлюватимуть функції з керування базою даних, бізнес-логікою та інтерфесом. Крім того, для модульності та незалежності компонентів програми буде застосовано патерн MVC. Така структура дозволятиме змінювати, масштабувати різні частини проєкту без наслідків для інших складових елементів програми.

Спроєктовано ER-модель бази даних з сутностями Користувач, Завдання, Категорія завдань, Предмети магазина, Предмети інвентаря, Об’єкт вирощування. Визначені типи зв’язків між таблицями та типи даних їхніх атрибутів. Створена модель є базою для розробки реляційної бази даних проєкта.

На основі сформульованих функціональних вимог побудовані діаграми варіантів використання для ілюстрації можливої взаємодії користувача з системою: авторизації, перегляду, редагування, збереження, видалення та фільтрації завдань, взаємодії з предметами магазина, інвентаря, опцій для налаштування акаунта тощо. Для зручності варіанти створення нових завдань було виділено в окрему діаграму. Розроблені у цьому розділі моделі будуть реалізовані та протестовані в наступному.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Розробка бази даних

Створення та робота з базою даних відбувається на рівні доступу до даних (DAL) архітектури програми. Структура Data Access Layer наведена на рис.3.1.

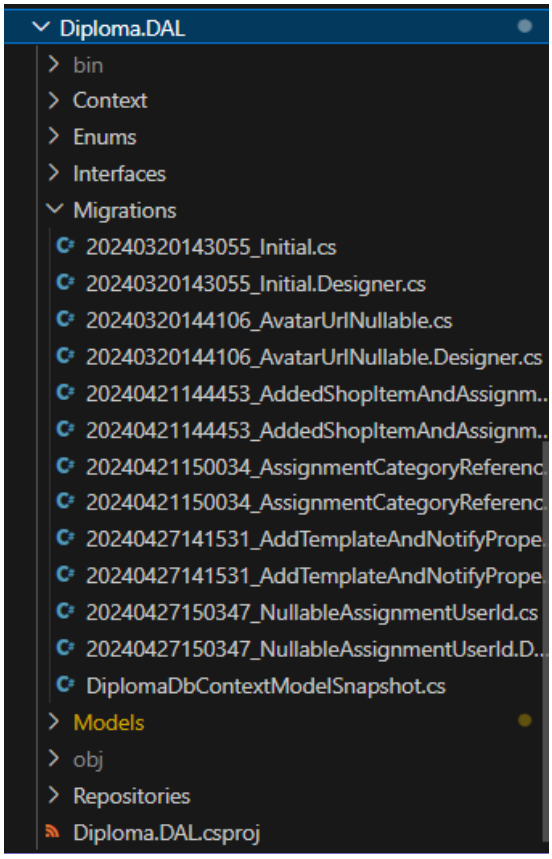

Рисунок 3.1 – Складові частини DAL

Замість створення SQL-запитів, Entity Framework Core (EF) дозволяє працювати з базами даних безпосередньо через програмний код, автоматично створювати, оновлювати, заповнювати даними та видаляти таблиці, використовувати LINQ (Language Integrated Query, запити мовою C#) для здійснення запитів до бази даних. Створення та внесення змін до бази даних здійснюється шляхом згенерованих міграцій. Кожного разу, коли потрібно створити нову таблицю, додати новий стовпець, змінити тип даних існуючого тощо, запускається міграція. Кожен .cs файл міграції з рис.3.1 має метод Up(), що

відповідає за створення та внесення змін, та метод Down(), що містить інструкції для можливості повернення до попередньої версії у разі необхідності.

Створення таблиці бази даних на прикладі таблиці Завдання (Assignment):

```
protected override void Up(MigrationBuilder migrationBuilder)
{
  migrationBuilder.CreateTable(
     name: "Assignment",
     columns: table => new
     {
          Id = table.Column<int>(type: "int", nullable: false)
          .Annotation("SqlServer:Identity", "1, 1"),
          Name = table.Column<string>(type: "nvarchar(max)",
          nullable: false),
          Description = table.Column<string>(type: "nvarchar(max)",
          nullable: false),
          UserId = table.Column<int>(type: "int", nullable: true),
          CategoryId = table.Column<int>(type: "int", nullable:
          true),
          Priority = table.Column<int>(type: "int", nullable:
          false),
          Status = table.Column<int>(type: "int", nullable: false)
       },
      constraints: table =>
       {
          table.PrimaryKey("PK_Assignment", x => x.Id);
          table.ForeignKey(
               name: "FK_Assignment_ApplicationUser_UserId",
               column: x => x.UserId,
               principalTable: "ApplicationUser",
               principalColumn: "Id",
               onDelete: ReferentialAction.Cascade);
          table.ForeignKey(
               name: "FK_Assignment_AssignmentCategory_CategoryId",
               table: "Assignment",
               column: "CategoryId",
               principalTable: "AssignmentCategory",
               principalColumn: "Id",
               onDelete: ReferentialAction.Cascade);
        });
```

Цей програмний код створює таблицю Assignment з полями, для кожного з яких визначається тип даних: id, назва, опис, id користувача, id категорії, пріоритет, статус завдання. Поле id визначається як первинний ключ таблиці, поле UserId – як зовнішній ключ для зв’язку з таблицею користувачів, поле CategoryId – як зовнішній ключ для зв’язку з таблицею категорій.

Крім того, для швидшого доступу до даних для кожної таблиці створюються індекси:

```
  migrationBuilder.CreateIndex(
      name: "IX_Assignment_UserId",
      table: "Assignment",
      column: "UserId");
}
  migrationBuilder.CreateIndex(
      name: "IX_Assignment_CategoryId",
      table: "Assignment",
      column: "CategoryId");
```

Останній метод класу дозволяє скасувати внесені зміни за потреби:

```
protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Assignment");
        }
```

В процесі розробки програми з’являється необхідність додавати нові або змінювати існуючі стовпці створених таблиць. Для цього створюється нова міграція з використанням методів AddColumn(), RenameColumn(), AlterColumn<type>() чи інших в залежності від характеру бажаних змін. Приклад додавання нового стовпця:

```
protected override void Up(MigrationBuilder migrationBuilder)
{
  migrationBuilder.AddColumn<bool>(
     name: "IsTemplate",
     table: "Assignment",
     type: "bit",
     nullable: false,
     defaultValue: false);

protected override void Down(MigrationBuilder migrationBuilder)
{
  migrationBuilder.DropColumn(
     name: "IsTemplate",
     table: "Assignment");
}
```

Цей програмний код додає булевий стовбець, який містить відповідь користувача, чи потрібно зберегти створюване завдання як шаблон.

Щоб зміни, зроблені програмно, зберігалися в базі даних, в консолі пишуться команди Add-Migration та Update-Migration. Для з’єднання з базою даних SQL Server, в файлі appsettings.json зберігається інформація про адресу її локального сервера в ConnectionStrings.

Атрибути для кожної таблиці баз даних, на основі яких відбувається генерація міграцій, зберігаються в моделях. Модель класу Завдання з усіма необхідними полями зображена на рис. 3.2.

```
44 references
public class Assignment
{
    4 references
    public int Id { get; set; }

    8 references
    public string Name { get; set; }

    6 references
    public string Description { get; set; }

    5 references
    public int? CategoryId { get; set; }

    4 references
    public DateTimeOffset StartsAt { get; set; }

    5 references
    public int? UserId { get; set; }

    3 references
    public bool Notify { get; set; }

    4 references
    public bool IsTemplate { get; set; }

    6 references
    public AssignmentPriority Priority { get; set; }

    4 references
    public AssignmentStatus Status { get; set; }

    0 references
    public virtual AssignmentCategory? Category { get; set; }

    1 reference
    public virtual ApplicationUser? User { get; set; }
}
```

Рисунок 3.2 – Модель для таблиці БД Assignment

Властивості, що позначені символом знаку питання, можуть мати значення null. Властивості, що передають об'єкти з оголошенням virtual, можуть перевизначатися в інших класах. Значення накшталт пріоритету та статусу, що

мають декілька варіантів вибору, містяться в окремому каталозі Enums та конвертуються в числові значення для зберігання в базі даних з міркувань безпеки. На прикладі можливих варіантів позначення пріоритету (високий, середній, низький), значення, що зберігається в таблиці змінюється від 0 до 2.

Каталог Context зберігає файл з усіма створеними сутностями бази даних. Оголошення залежностей між ними на прикладі зв'язку Користувач – Завдання (один до багатьох):

```
modelBuilder
    .Entity<Assignment>()
    .HasOne(x => x.User)
    .WithMany(x => x!.Assignments).HasForeignKey(x => x.UserId);
```

Репозиторії відповідають за здійснення операцій в базі даних. Кожен з них має відповідний інтерфейс, необхідний за принципом SOLID для забезпечення залежності класів від абстракції замість реалізації. Приклад класу репозиторія для обробки даних таблиці з завданнями наведений на рис. 3.3.

```
8   public class AssignmentRepository : IAssignmentRepository
15      }
        3 references
16      public async Task<Assignment> CreateAsync(Assignment assignment)
17      {
18          _context.Assignments.Add(assignment);
19          await _context.SaveChangesAsync();
20
21          return assignment;
22      }
        4 references
23      public async Task<IEnumerable<Assignment>> GetAssignmentsByUserId(int? userId)
24      {
25          return await _context.Assignments.Where(x => x.UserId == userId).IgnoreAutoIncludes().ToListAsync();
26      }
        6 references
27      public async Task<Assignment?> GetAssignment(int assignmentId)
28      {
29          return await _context.Assignments.FirstOrDefaultAsync(x => x.Id == assignmentId);
30      }
        4 references
31      public async Task UpdateAsync(Assignment assignment)
32      {
33          _context.Assignments.Update(assignment);
34          await _context.SaveChangesAsync();
35      }
        2 references
36      public async Task<bool> DeleteAsync(int id)
37      {
38          var assignment = await GetAssignment(id);
39          if (assignment != null)
40          {
41              _context.Assignments.Remove(assignment);
42              await _context.SaveChangesAsync();
43              return true;
44          }
45          return false;
46      }
47  }
```

Рисунок 3.3– Файл AssignmentRepository.cs

Інтерфейси оголошують створені в репозиторіях методи. На рівні бази даних здійснюється лише CRUD операції : створення запису, виведення всіх записів та виведення одного запису по id через LINQ запити, оновлення запису, видалення запису. Більш складний функціонал виконуватиметься на рівні бізнес-логіки застосунку.

3.2 Розробка серверної частини з використанням ASP.NET

ASP.NET надає гнучку і потужну інфраструктуру для керування аутентифікацією, авторизацією, рівнем доступа користувачів до функціоналу та багатьма іншими аспектами за допомогою засобів Identity Framework. ASP.NET Core Identity – це API, що працює з акаунтами користувачів, паролями, персональними даними, ролями, токенами, підтвердженням електронної пошти, зовнішніми засобами входу тощо і автоматично створює таблиці для зберігання даних про можливості кожної ролі, cookies-файли користувача, статус акаунта та багато іншого.

IdentityUser є класом, від якого успадковується клас ApplicationUser для створення користувачів застосунку. Його вбудовані атрибути – це id, ім’я користувача, пошта, номер телефону та пароль. Замість введеної користувачем стрічки паролю, система генерує його унікальний хеш (випадковий рядок фіксованої довжини) для надійного і безпечного зберігання в базі даних.

Також Identity має класи UserManager, що має вбудовану бізнес-логіку щодо створення, пошуку, оновлення, видалення користувачів, та SignInManager з методами для входу та виходу з акаунта.

Коли користувач натискає на кнопку «зареєструватися», в контролері відправляється POST запит. Дані, що були введені в текстових полях на сторінці реєстрації та записані в моделі, використовуються для ініціалізації пошти та юзернейма користувача. Інші стандартні необов’язкові атрибути, а також оброблений пароль додаються в базу даних через UserManager асинхронним методом CreateAsync(), що створює користувача. Якщо запис успішно внесено в базу даних – методом SignInAsync() з SignInManager користувач входить в свій

створений обліковий запис та відбувається переадресація на головну сторінку застосунка. В іншому випадку відображається помилка.

```
[HttpPost]
    [Route("register")]
    public async Task<IActionResult> Register(RegisterViewModel
registerViewModel)
    {
        var user = new ApplicationUser()
        {
          Email = registerViewModel.Email,
          UserName = registerViewModel.UserName,
        };
          var result = await _userManager.CreateAsync(user,
          registerViewModel.Password);
        if (result.Succeeded)
        {
          await _signInManager.SignInAsync(user, false);
          return RedirectToAction(nameof (HomeController.Index),
          "Index");
        }
        else
        {
            return BadRequest("Error while registering");
        }
    }
```

Так само для входу та виходу з акаунта використовуються методи PasswordSignInAsync() та SignOut() з SignInManager. Таблиці, попередньо створені Identity для процесів аутентифікації (напр. IdentityUserClaim, IdentityUserToken), заповнюються автоматично та дозволяють користувачам залишатися в своєму обліковому записі після переадресації контролера на іншу сторінку застосунка.

```
[HttpPost]
    [Route("login")]
    public async Task<IActionResult> Login(LoginViewModel
loginViewModel)
    {
        var result =
await _signInManager.PasswordSignInAsync(loginViewModel.UserName,
loginViewModel.Password, false, false);
        if (result.Succeeded)
        {
          return RedirectToAction(nameof (HomeController.Index),
          "Index");
        }
        else
```

```
        {
            return NotFound();
        }
    }

    [HttpPost]
    [Route("logout")]
    public async Task<IActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return RedirectToAction(nameof (LoginController.Index),
"Index");
    }
```

Обробкою завдань займається клас AssignmentService та його інтерфейс з рівня бізнес-логіки. До CRUD операцій, що посилаються на методи з репозиторія з DAL, додаються інші необхідні методи. В цьому сервісі завданням присвоюються значення пріоритету та статусу, проходить фільтрація завдань, а також розподілення на шаблони, якщо поле isTemplate в таблиці бази даних завдань дорівнює true. Приклад процесу фільтрації завдань за категорією:

```
public async Task<IEnumerable<Assignment>> FilterTasksByCategory(int
userId, int categoryId)
    {
        var assignments = await
_assignmentRepository.GetAssignmentsByUserId(userId);
        var filteredAssignments = assignments.Where(assignment =>
assignment.CategoryId == categoryId);
        return filteredAssignments;
    }
```

Так само в AssignmentCategoryService виконуються операції зі створення, видалення, оновлення та зчитування категорій, щоб користувач мав можливість створювати власні категорії та зберігати їх в базі даних.

В контролері з використанням створених методів відправляються GET та POST запити для додавання та редагування завдань користувача. Шаблонні завдання розподіляються на системні та користувальницькі за значенням поля UserId: якщо воно не дорівнює нулю – це шаблон користувача, в іншому випадку – системний шаблон, доданий до бази даних при створенні застосунку. Процес відправлення запитів з контролеру на прикладі редагування завдання:

```
[HttpGet]
    public async Task<IActionResult> EditAssignment(int id)
    {
     var assignmentCategories =
     await _assignmentCategoryService.GetAssignmentCategories();
     var assignment =
     await _assignmentService.GetAssignmentById(id);
        if (assignment == null)
        {
            return NotFound();
        }
        var model = new AssignmentViewModel(assignment)
        {
          Categories = assignmentCategories.Select(x => new
          SelectListItem { Text = x.Name, Value = x.Id.ToString() })
        };
        return View("AddAssignment", model);
    }
```

Цей запит відображає користувачеві сторінку редагування завдання з отриманою з бази даних інформацією про конкретне завдання за його id з використанням створеного в сервісі метода GetAssignmentById(). Також користувач отримує список категорій, з якого він може вибрати потрібну методом Select().

Для збереження змін після введення даних в поля завдання, коли користувач натискає на кнопку «Готово» відправляється POST запит:

```
    [HttpPost]
     public async Task<IActionResult>
     EditAssignment(AssignmentViewModel model)
    {
        //валідація введених даних
        if (ModelState.IsValid)
        {
          //отримання дати і часу в форматі utc для різних часових
          поясів, об'єднання їх в один об'єкт DataTime
          DateTimeOffset.TryParse(model.StartDate, out
          DateTimeOffset date);
          DateTimeOffset.TryParse(model.StartTime, out
          DateTimeOffset time);
          var startsAt = new DateTime(date.Year, date.Month,
          date.Day, time.Hour, time.Minute, 0);

          var user = await _userManager.GetUserAsync(User);
          var assignment = new Assignment()
          {
               //ініціалізація полів завдання
                Name = model.Name,
```

```
                Description = model.Description,
                Priority = (AssignmentPriority)model.Priority,
                UserId = model.UserId,
                CategoryId = model.CategoryId,
                IsTemplate = model.IsTemplate,
                Notify = model.Notify,
                Status = AssignmentStatus.New,
                StartsAt = new DateTimeOffset(startsAt)
            };
    //оновлення завдання і переадресація на головну сторінку
            await _assignmentService.UpdateAssignment(assignment);
            return Redirect("/");
        }
        else
        {
    //якщо валідація не була успішною, користувач отримує ту саму ж
    сторінку; перелік категорій прогружається знову.
            var assignmentCategories = await
_assignmentCategoryService.GetAssignmentCategories();
            model.Categories = assignmentCategories.Select(x => new
SelectListItem { Text = x.Name, Value = x.Id.ToString() });
            return View(model);
        }
    }
```

Зміни інформації про користувача або об’єкт вирощування відбуваються аналогічно.

Методи роботи з об’єктом вирощування знаходяться в GrowingObjectService. Разом з CRUD-операціями тут відбувається оновлення прогресу та стадії об'єкта. Процент прогресу об'єкта збільшується кожен день, коли користувач виконує 70% від загальної кількості завдань. Такий день в системі вважається успішним (Successful Day), за кожний день об’єкт отримує певний процент прогресу в залежності від стадії рослини. Принцип роботи на прикладі стадії 2, коли об’єкт отримує 1/3 прогресу на день:

Додавання відсотка прогресу об’єкту в разі успішного дня:

```
    if (isSuccessfulDay)
    {
        GrowingObject.Progress += 34;
        GrowingObject.LastProgressUpdate = DateTime.Today;
//оновлення дати останнього додавання прогресу
```

Перехід на наступну стадію після заповнення прогресу:

```
        if (GrowingObject.Progress >= 100)
        {
          if (GrowingObject.Stage<4)
```

```
            {
              GrowingObject.Progress -= 100;
              GrowingObject.Stage += 1;
             }
          }
      //Збереження змін в базі даних
      await _ GrowingObject
      Repository.UpdateGrowingObjectProgress(Progress);
      await _ GrowingObject
      Repository.UpdateGrowingObjectStage(Stage);
```

Віднімання відсотка прогресу в разі неуспішного дня:

```
   else
    {
        GrowingObject.Progress -=25;
        GrowingObject.LastProgressUpdate = DateTime.Today;
//оновлення дати останнього додавання прогресу
     }
```

Перехід на попередню стадію, якщо прогрес опускається до 0:

```
        if (GrowingObject.Progress <= 0)
        {
          if (GrowingObject.Stage>1)
          {
            GrowingObject.Stage -= 1;
           }
        }
```

Контролер отримує дані стадії об’єкту та відображає зображення по індексу від 0 до 3 відповідно до стадії. Посилання на зображення зберігаються в масиві, а самі зображення – на безкоштовному хостинг сервері imgur.com. Отримується посилання шляхом GET запиту на API imgur.com.

Аналогічно з додаванням прогресу до об’єкту вирощування, за кожне виконане завдання додається 10% прогресу рівня користувача. Коли прогрес досягає 100%, рівень користувача підвищується на 1 та користувач отримує ігрову валюту:

```
if (ApplicationUser.LevelProgress == 100)
{
    ApplicationUser.Level +=1 ;
    UpdateApplicationUserLevel(userId, ApplicationUser.Level);
    AddCoinsToUser(userId, 50);
}
```

За ігрову валюту користувач може придбати предмети з магазину. Контролер використовує методи з ShopItemService, щоб переглянути всі предмети магазину, переглянути конкретний предмет та придбати предмет в потрібній кількості:

```
//Перегляд всіх елементів та вибору конкретного продукту
[Route("shop")]
    public async Task<IActionResult> Index ()
    {
       var user = await _userManager.GetUserAsync(User);
       var models = await _shopItemService.GetShopItems();
       var viewModels = models.Select(x => new ShopItemViewModel());
       return View(viewModels);
     }

//Перегляд конкретного продукту по id
    [HttpGet]
    public async Task<IActionResult> ViewShopItem(int id)
    {
        var shopItem = await _shopItemService.GetShopItemById(id);
        if (shopItem == null)
        {
            return NotFound();
        }
        var model = new ShopItemViewModel(shopItem);

        return View("ShopItem", model);
    }


//Купівля та додавання предмету в інвентар
    [HttpPost]
    public Task<ActionResult> BuyItem(ShopItemViewModel model)
{
        var user = await _userManager.GetUserAsync(User);
        if (user.Money>= model.Price * model.Quantity)
    {

        user.Money -= model.Price * model.Quantity;
        await _userManager.UpdateAsync(user);

        var inventoryItem = new ShopItem()
            {
                ItemId = model.ItemId,
                ItemName = model.ItemName,
                UserId = modelUserId,
                Quantity = model.Quantity
            };
         await _inventoryService.CreateInventoryItem(inventoryItem);
        return Redirect("/");
```

```
    }
    else
    {
        return BadRequest("ErrorMessage");
    }
```

Всі сторінки відображаються на екрані з використанням подання (View), в якості якого виступають компоненти React – JavaScript файли, що повертають React-елементи. За допомогою бібліотеки axios підтримується взаємодія зі створеними серверними елементами та відправляються запити. Компоненти отримують інформацію з моделей, відображають та структурують інтерфейс по блоках, а хуки UseState() та UseEffect() дозволяють створювати та оновлювати стан функціональних компонентів, реагувати на події взаємодії користувача з інтерфейсом. Контролери здійснюють навігацію по створених сторінках інтерфейсу, а в App.js визначається маршрутизатор Router зі списком Route-елементів, що відповідають за відображення компонентів.

Приклад екрану, що відображає загальний список завдань користувача:

```
export default function Assignments() {
  const [assignments, setAssignments] = useState([])

    const fetchAssignments = async () => {
        const { data, errorMessage } = await
assignmentService.getAllAssignments();

        setAssignments(data)
    };
    useEffect(() => {
      fetchAssignments();
    });
  return (
    <div className='mx-2'>
      <div className='row d-flex justify-content-between'>
        <a className="col-3"><i className="fa-solid far fa-filter
green-color"></i></a>
        <div className="col-6 text-center">Мої завдання</div>
        <a className="col-3 text-end">
          <Link to="create">
            <i className="fa-solid far fa-circle-plus green-
color"></i>
          </Link>
        </a>
      </div>
      {assignments.map(assignment =>
          <div className="mt-2 pt-2 pb-2 d-flex bg-light rounded">
```

```
            <div className="form-check">
                <input className="form-check-input" type="checkbox"
value="" id="flexCheckChecked" checked/>
                <label className="form-check-label"
for="flexCheckChecked"></label>
            </div>
            <div className="ms-2">{assignment.name}</div>
            <a className="ms-auto"><i className="fas fa-pencil
green-color"></i></a>
        </div>
      )}
    </div>
  );
}
```

Щоб запустити проєкт на мобільному пристрої, потрібно підключити фреймворк Capasitor.js до існуючого React застосунку шляхом використання переліку команд пакетного менеджера npm в консолі:

```
npm install -g @capacitor/cli
npx cap init
npm install @capacitor/core @capacitor/ios @capacitor/android
npx cap add ios
npx cap add android
```

Після запуску React-проєкту (npm run build) потрібно синхронізувати веб-застосунок з нативним мобільним проєктом за допомогою npx cap sync та відкрити в середовищі AndroidStudio для операційної системи Android або в Xcode для iOS.

Відкриємо застосунок на прикладі iOS (команда npx cap open ios).

3.3 Результат роботи програми

Перші сторінки застосунку здійснюють авторізацію користувача. Щоб увійти в існуючий обліковий запис, користувач має ввести інформацію, з якою він реєструвався в системі (рис.3.4)

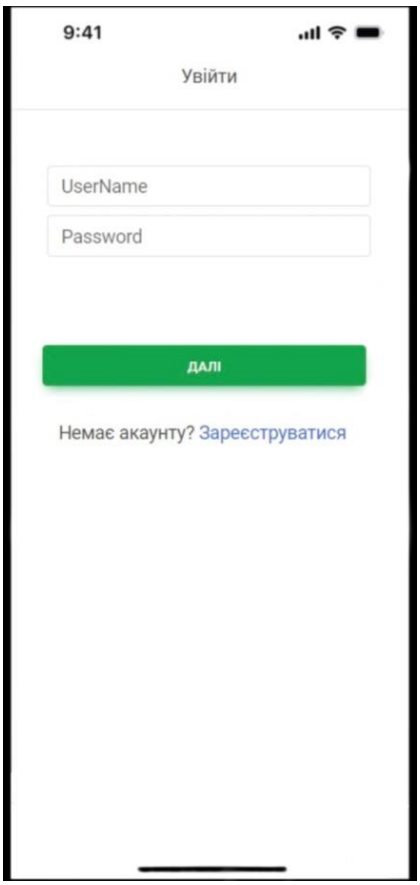

Рисунок 3.4 – Сторінка авторизації

Якщо користувач хоче створити новий обліковий запис, він може натиснути на кнопку «Зареєструватися». Реєстрація складається з двох сторінок: на першій користувач вводить свої дані, а на другій вказує ім’я та вид рослини. Вигляд сторінок реєстрації показано на рис. 3.5.

Після валідації даних для входу в обліковий запис відбувається переадресація на головну сторінку. Користувач може побачити свій перелік завдань та кількість активних, що залишилося виконати. На головній сторінці знаходиться рослина в поточній стадії розвитку (стадія 3 на рис.3.6) та шкала прогресу. Панель навігації має 4 кнопки, що відповідають за екрани домашньої сторінки, завдань, магазину та профілю користувача. Додатково з домашньої сторінки можна перейти на налаштування або інвентар.

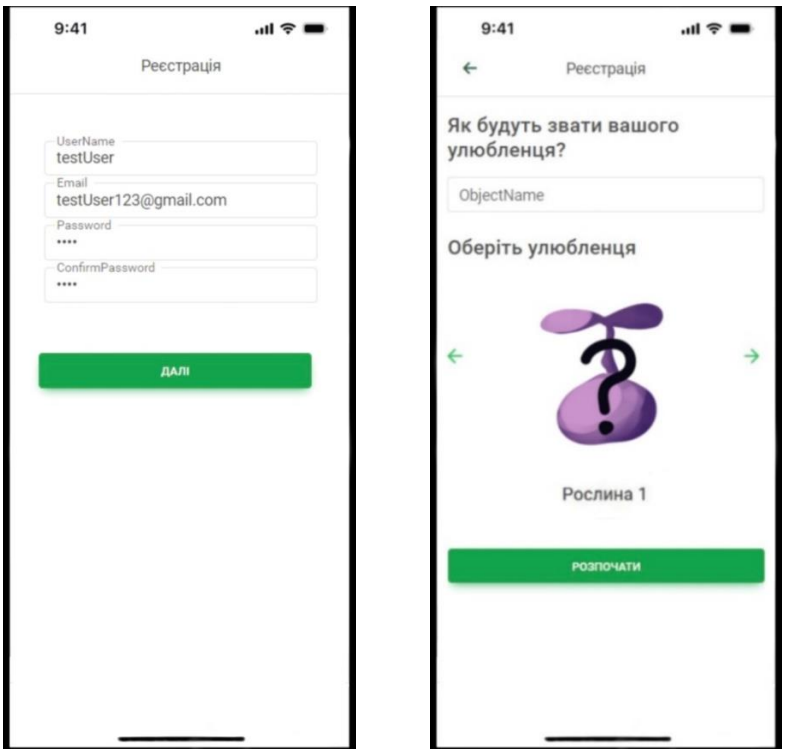

Рисунок 3.5 – Перша і друга сторінки реєстрації

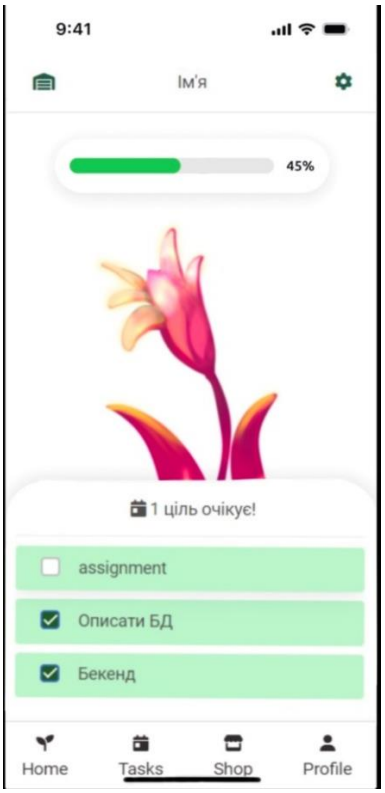

Рисунок 3.6 – Домашня сторінка застосунку

На сторінці «Мої завдання» (рис. 3.7) є повний перелік створених завдань з можливістю позначати виконані завдання натисканням на чекбокс, редагувати та

фільтрувати їх за категоріями. Нові завдання створюються через кнопку «+» у верхньому правому куті екрану.

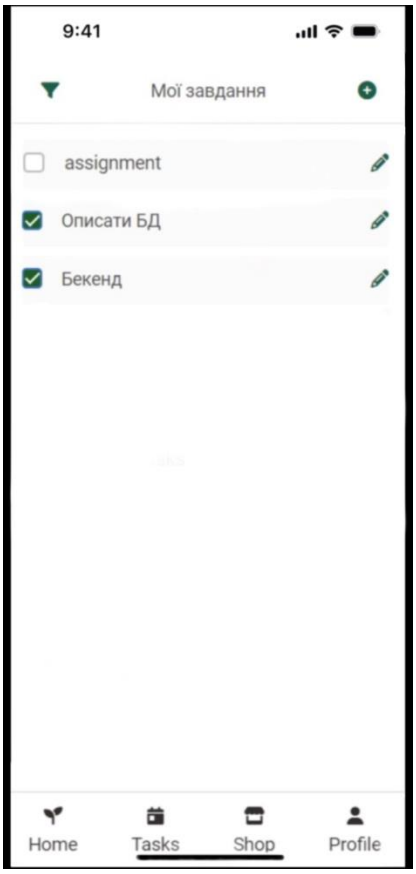

Рисунок 3.7 – Сторінка «Мої завдання»

При створенні завдання користувач переадресовується на сторінку вибору (рис. 3.8), де можна вибрати створити нове завдання і власноруч ввести потрібні деталі чи вибрати з готових шаблонів. Сторінка, на якій створюється нове завдання (рис. 3.9), має обов’язкові поля назви та опису завдання, а також додаткові опціональні поля з вибором пріоритету, категорії, дати, а також можливість для нагадувань та збереження шаблону завдання. У шаблонах поля, окрім дати, вже є заповненими, проте кожне зі значень можна змінити в будь-який момент.

Рисунок 3.8 – Сторінка вибору створення завдання

Після введення потрібних даних користувач зберігає завдання натисканням кнопки «Готово» або видаляє кнопкою «Видалити».

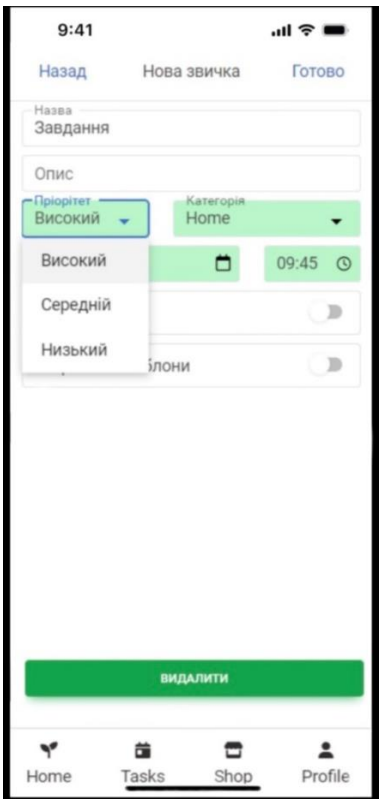

Рисунок 3.9 – Сторінка створення нового завдання

На рис. 3.10 зображено приклад валідації введених даних та відображення помилки, тому що опис завдання зазначено обов’язковим полем для створення нового завдання. Повідомлення про помилку відображаються аналогічним чином для всіх тестових полів застосунку, де це передбачено.

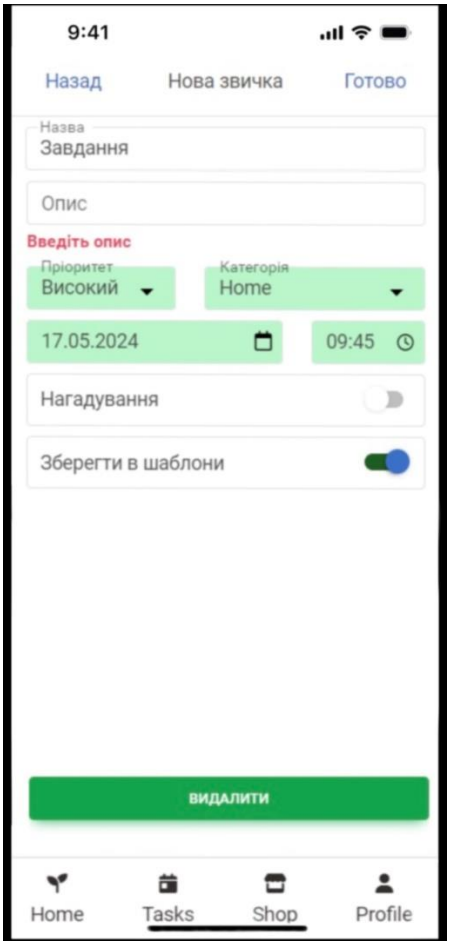

Рисунок 3.10 – Перевірка введення даних

Якщо користувач хоче вибрати завдання з шаблону, відбувається переадресація на сторінку «Мої шаблони» (рис. 3.11), де всі шаблони поділяються на створені користувачем та системні, що мають всі користувачі застосунку. Шаблонні завдання можна редагувати, а також додавати в перелік активних завданнь аналогічно з новими кнопкною «Готово». Відміннюстю від сторінки створення нового завдання є автоматичне заповнення полів інформацією з бази даних.

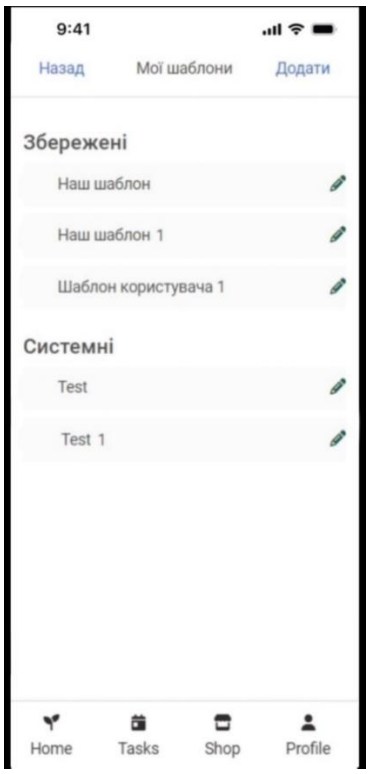

Рисунок 3.11 – Сторінка «Мої шаблони»

За допомогою панелі навігації користувач може перейти на сторінку магазину (рис. 3.12). На ній є доступні для придбання товари, а також поточний баланс користувача в правому верхньому куті екрану.

Рисунок 3.12 – Сторінка «Магазин»

При натисканні на конкретний товар відображається віконце інформації про товар (рис. 3.13) з його вартістю в залежності від вибраної кількості. Придбані товари зберігаються в інвентарі, що має аналогічний сторінці магазину вигляд.

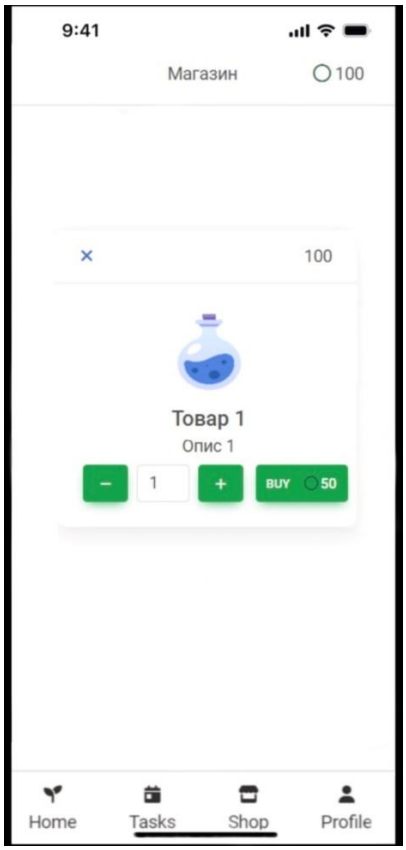

Рисунок 3.13 – Сторінка інформації про товар

Останньою сторінкою, на яку можна перейти з навігаційної панелі є профіль користувача (рис. 3.14), де відображається інформація про вид, стадію та дату вибору рослини, вказану як її день народження. Також на цій сторінці користувач може ознайомитися, скільки днів він вже виконує завдання, побачити свій рівень та поточний показник прогресу рослини, названий її здоров'ям. З неї, як і з домашньої сторінки, є можливість перейти до налаштувань, натиснув на іконку в правому верхньому куті екрану.

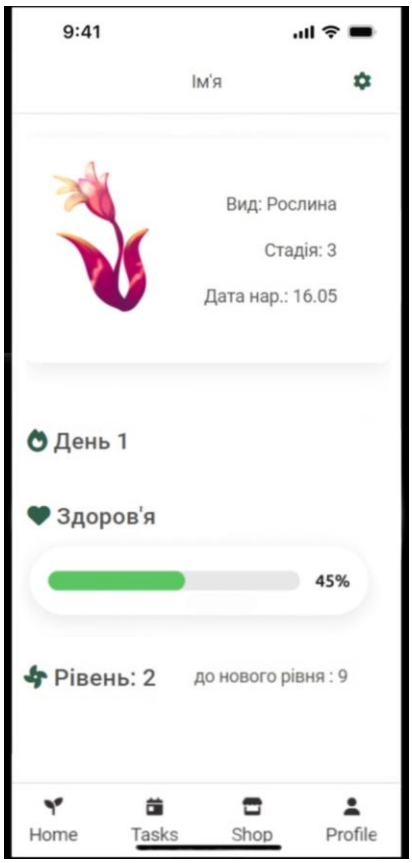

Рисунок 3.14 – Сторінка профілю користувача

Сторінка налаштувань (рис. 3.15) дозволяє зміни інформацію про користувача та рослину, зберегти зміни кнопкою «Зберегти» або вийти з акаунта.

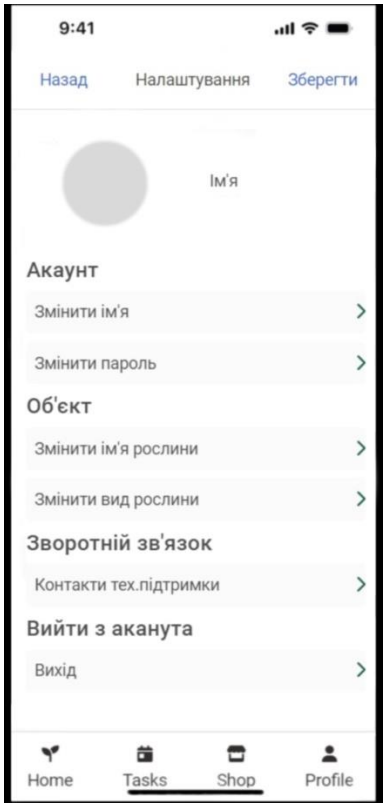

Рисунок 3.15 – Сторінка налаштування

3.4 Тестування програмного продукту

Для здійснення ручного тестування програмного продукту, потрібно застосувати тест-кейси та чеклісти. Тест-кейс – це описана послідовність дій, спрямована на перевірку тестувальником будь-якого функціоналу або властивості програмного продукту. Структура тест-кейсу складається з назви, загального огляду, кроків для виконання тесту та очікуваного результату.

Тест-кейс 1 на рис. 3.16 перевіряє процес реєстрації користувача:

#	Test Title	Test Summary	Test Steps	Test Data	Expected Result
1	Реєстрація	Реєстрація нового користувача з валідними даними	1. Відкрити застосунок 2. Натиснути кнопку "Зареєструватися" 3. В полі ім'я ввести user 5. В полі пошта ввести "test1234@gmail.com" 6. В полі пароль ввести "test1234" 7. В полі повторити пароль ввести "12345678" 8. Натиснути кнопку "Далі" 9.В полі ім'я рослини ввести plant 10. Натиснути кнопку "Розпочати"	Відкрити сайт, не мати існуючого користувача з заданими даними	Реєстрацію здійснено. Відбулася переадресація на домашню сторінку застосунку.

Рисунок 3.16 – Тест-кейс 1

Тест-кейс 2 на рис. 3.17 перевіряє процес авторизації користувача:

2	Авторизація з існуючого акаунту	Зробити вхід до існуючого облікового запису з валідними даними	1. Відкрити застосунок 2. В полі пошта ввести "test1234@gmail.com" 3. В полі пароль ввести "test1234" 4. Натиснути кнопку "Далі"	Мати зареєстрованого користувача з такими даними: логін "test1234@gmail.com", пароль "test1234"	Успішна авторизація. Переадресація на домашню сторінку застосунку.

Рисунок 3.17 – Тест-кейс 2

Тест-кейс 3 на рис.3.18 перевіряє процес додавання нового завдання:

3	Додати нове завдання	Створити нове завдання з заповненням всіх необхідних полів	1. Зайти на сторінку "Мої завдання" 2. Натиснути на + -> створити 3. Ввести назву завдання "Завдання 1" 4. Ввести опис завдання "Тестування роботи застосунку" 5. Вказати високий пріоритет 6. Натиснути "Готово"	Мати зареєстрованого користувача	Успішне внесення нового запису в базу даних. Відображення нового завдання на екрані.

Рисунок 3.18 – Тест-кейс 3

Тест-кейс 4 на рис. 3.19 перевіряє процес фільтрування списку завдань:

4	Фільтрувати список завдань	Здійснити фільтрацію списку завдань за категорією	1. Зайти на сторінку "Мої завдання" 2. Натиснути іконку фільтру в лівому верхньому куті екрану 3. Обрати "По категорії" -> "Дім"	Мати створені завдання з вказаною категорією "дім"	Успішна фільтрація списку, відображаються лише завдання категорії "дім"

Рисунок 3.19 – Тест-кейс 4

Тест-кейс 5 на рис. 3.20 перевіряє процес створення завдання з шаблону:

5	Створити завдання з шаблону	Обрати створення завдання з шаблонів користувача	1. Натиснути кнопку "+" -> "Вибрати з шаблонів" 2. Натиснути на Шаблон 1 з шаблонів "Збережені" 3. Редагувати назву з "Шаблон 1" на "тест-кейс 5" 4. Обрати дату "20 травня" 5. Натиснути "Готово"	Мати збережені шаблони користувача	Створено нове завдання з заповненими шаблоном полями. Назва змінена з шаблонної на "тест-кейс 5", виставлена дата

Рисунок 3.20 – Тест-кейс 5

Тест-кейс 6 на рис. 3.21 перевіряє процес купівлі товару в магазині.

6	Здійснити покупку в магазині	Придбати Товар 1 в магазині за 50 монет	1. Відкрити сторінку "Магазин" 2. Натиснути на Товар 1 3. Обрати кількість 1 4. Натиснути "Buy"	Наявність в базі даних товару 1 вартістю 50 монет	50 монет списалося з балансу користувача, придбаний товар можна переглянути в інвентарі.

Рисунок 3.21 – Тест-кейс 6

Тест-кейс 7 на рис. 3.22 перевіряє процес збереження завдання в шаблони:

7	Зберегти завдання як шаблон	Зберегти створене раніше завдання в шаблони користувача	1. Натиснути іконку редагування завдання 2. Ввімкнути опцію "Зберегти в шаблон" 3. Натиснути кнопку "Готово"	Наявність нешаблонного завдання	Завдання записано в БД як шаблон та відображається у вікні Збережені

Рисунок 3.22 – Тест-кейс 7

Тест-кейс 8 на рис.3.23 перевіряє процес редагування завдання.

8	Редагувати завдання зі списку	Змінити поля завдання зі списку активних	1. Натиснути іконку редагування завдання 2. Змінити поле "Назва" на "тест-кейс 8" 3. Змінити поле "Опис" на "тестове завдання" 4. Змінити пріоритет з високого на низький 5. Змінити категорію на "Робота" 6. Обрати поточні дату та час 7. Натиснути "Готово"	Мати активне завдання з заповненими даними	Всі змінені поля завдання відображають нові коректні значення на екрані та в базі даних

Рисунок 3.23 – Тест-кейс 8

Тест-кейс 9 на рис. 3.24 перевіряє процес видалення завдання.

9	Видалити завдання	Обрати зі списку та видалити завдання	1. Натиснути іконку редагування завдання 2. Натиснути "Видалити"	Мати створене завдання	Завдання видалено зі списку за з бази даних

Рисунок 3.24 – Тест-кейс 9

Тест-кейс 10 на рис.3.25 перевіряє виходу з облікового запису.

10	Вийти з акаунта	Здійснити вихід з поточного облікового запису	1. Зайти на сторінку "Налаштування" 2. Обрати "Вихід" 3. Натиснути "Вийти з акаунта"	Мати зареєстрованого користувача	Здійснено вихід з акаунта, переадресація на сторінку логіну

Рисунок 3.2 – Тест-кейс 10

Чекліст – це список, що містить ряд необхідних під час тестування перевірок частин програмного продукту. Чеклісти складаються з номеру, назви, статусу та коментарів. Добір тестових ситуацій для чекліста відбувається з використанням негативного та позитивного тестування, де один і той самий елемент перевіряється введенням валідних та невалідних вхідних даних. Створений для перевірки функціоналу застосунку чекліст та результати показані на рис. 3.26.

№	Name	Status	Comment
1	Реєстрація з валідними даними	pass	
2	Вказання різних даних в полях "Пароль" та "Підтвердити пароль"	pass	не дозволяє зареєструватися, повідомлення про помилку
3	Реєстрація без вказання ім'я об'єкта	fail	дозволяє зареєструватися з пустим полем ObjectName
4	Логін з невалідними даними	pass	не дозволяє зайти в акаунт
5	Логін з валідними даними	pass	
6	Вдмітити завдання як виконане	pass	
7	Створити завдання з порожнім полем назви	pass	не дозволяє створити, повідомлення про помилку
8	Вибрати дату та час раніші за поточні	pass	минулі дата та час не вибираються
9	Фільтрувати список завдань за категорією "Дім"	pass	
10	Купити товар за 100 монет при балансі в 50 монет	pass	не дозволяє купити за нестачі коштів
11	Вказати -1 при купівлі товару	pass	не дозволяє купити в кількості -1
12	Зберегти завдання як шаблон	pass	
13	Використати системний шаблон для створення нового завдання	pass	
14	Редагувати системний шаблон	fail	зміни не збереглися
15	Видалити активне завдання	pass	
16	Видалити завдання з шаблону	pass	
17	Виконати 7 з 10 завдань на день	pass	рівень прогресу збільшено
18	Редагувати ім'я користувача	pass	
19	Досягти 100% прогресу об'єкта	pass	стадія змінилася с 2 на 3
20	Вийти з облікового запису	pass	

Рисунок 3.26 – Чекліст

В результаті тестування чеклісту 18 кейсів пройшли перевірку зі статусом pass, 2 – зі статусом fail. Деталі задокументовано в баг-репорті на рис. 3.27 – 3.28.

Id	1
Опис	Внесення змін в системний шаблон
Передумова	Мати зареєстрованого користувата
Кроки	1. На сторінці "Мої завдання" обрати нове завдання з шаблонів 2. Вибрати для редагування системний шаблон 3. Змінити назву та категорію шаблона 4. Натиснути кнопку "Готово"
Очікуваний результат	Створення шаблону зі змінами як "Збережений користувачем шаблон"
Отриманий результат	Зміни ніяк не збереглися в базі даних

Рисунок 3.27 – Баг-репорт частина 1

Id	2
Опис	Реєстрація з порожнім полем ім'я об'єкта вирошування
Передумова	Відкрити сторінку реєстрації
Кроки	1. Ввести дані користувача в системі 2. Натиснути кнопку "Далі" 3. Залишити поле ObjectName порожнім 4. Натиснути кнопку "Розпочати"
Очікуваний результат	Авторизацію не здійснено, повідомлення про порожній рядок
Отриманий результат	Авторизація пройшла успішно, переадресація на домашню сторінку

Рисунок 3.28 – Баг-репорт частина 2

3.5 Висновки до розділу 3

У розділі описано процес розробки бази даних, backend та frontend частин застосунку з прикладами та поясненнями фрагментів програмного коду.

Реалізований функціонал програми проілюстовано скріншотами основних складових частин застосунку: сторінок авторизації та реєстрації користувача в системі, домашньої сторінки, сторінок з переліком завдань користувача та шаблонів, вибору та створення нових завдань, магазину, профілю користувача, налаштувань. Після цього проведено ручне тестування за допомогою тест-кейсів та чеклістів для перевірки коректності роботи створеного мобільного застосунку персонального планувальника справ. Створено 10 тест-кейсів та 20 чеклістів, за допогою яких було протестовано основний функціонал застосунку. Під час виконання тестових ситуацій виявлено 2 функціональні помилки, інформацію про них сформовано в баг-репорт.

ВИСНОВКИ

Наразі все частіше мобільні пристрої стали використовуватися для успішного планування справ протягом дня та контролю за їх виконанням. У кваліфікаційній роботі розроблено мобільний застосунок персонального планувальника справ як помічник у подоланні щоденних труднощів та пошуку мотивації у різні періоди життя.

Під час аналізу предметної області було з’ясовано специфіку розробки мобільного застосунку персонального планувальника справ. Після визначення переваг та недоліків функціоналу декількох продуктів-аналогів за допомогою SWOT-таблиць були сформовані власні функціональні вимоги для забезпечення конкурентоспроможності та унікальності створюваного програмного продукту. Обгрунтовано вибір засобів ASP.NET для серверної частини застосунку, використання SQL Server як системи керування базами даних, React.js для створення сторінок засобами веб-технологій та фреймворка Capacitor.js для підтримки платформ iOS та Android. Вибрані засоби забезпечують розробку швидкого високопродуктивного кросплатформного застосунку з можливістю подальшого розширення та ускладнення його функціоналу. Також розглянуто можливі маркетингові інструменти в створенні та просуванні створеного програмного продукту.

Проєкт втілює принципи трирівневої архітектури та MVC патерну, щоб зробити програмний код модульним та незалежним від змін в різних його складових частинах. Для проєктування бази даних визначені сутності, зв’язки та атрибути таблиць, побудована ER-діаграма. Функціонал, який надалі було реалізувано і протестувано, змодельований у створених UML-діаграмах варіантів використання.

Розроблений програмний продукт відповідає визначеним функціональним вимогам, забезпечує безпечну авторизацію користувача вбудованими методами ASP.NET, створення списку завдань на день з можливістю перегляду, редагування, видалення кожного його елементу, збереження завдань в шаблони з подальшим

їхнім використанням, придбання предметів з магазину, а також містить ігровий елемент у вигляді рослини з різними стадіями розвитку та систему рівнів для заохочення користувачів користуватися застосунком та виконувати щоденний плану.

Для підвищення якості продукту та перевірки його функціоналу, згідно з вимогами було проведено ручне тестування з використанням тест-кейсів та чеклістів, більшість з яких пройшли перевірку, а виявлені помилки були задокументовані в баг-репортах для ефективного усунення.

Подальшу розробку застосунку варто спрямувати на вдосконалення механіки розвитку та зміни стадій об’єкта вирощування, збільшити різноманітність варіантів об’єктів для вибору, розширити асортимент товарів в магазині та продумати варіанти їхнього використання. Крім того, технології штучного інтелекту можуть стати потужним інструментом для генерації шаблонів та підказок про щоденні справи користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Антіпіна Н.С. VR та AR технології. *Кібербезпека в сучасному світі*: матер. III всеукр. наук.-практ. конф. (м. Одеса, 19 листопада 2021 р.). Одеса, 2021. С. 115-120. URL: https://dspace.onua.edu.ua/items/1f42da14-5c08-4cdf-bc3c-6332c36bdb47

2. Антіпіна Н.С. Хмарні ігрові технології. *Інформаційне суспільство: проблеми та перспективи*: матер. VI всеукр. наук.-практ. конф. (м. Одеса, 28 травня. 2021 р.). Одеса, 2021. С. 10-14.

3. Антіпіна Н.С. Технології розробки ігрових застосунків засобами Unity. *Кібербезпека в сучасному світі*: матер. IV всеукр. наук.-практ. конф. (м. Одеса, 25 листопада 2022 року.). Одеса, 2022. С. 6-10.

4. Трофименко О.Г., Антіпіна Н.С. Маркетингові інструменти в створенні та просуванні програмних продуктів. *Актуальні тенденції розвитку освіти, науки та технологій*: матер. VII міжнар. наук.-практ. конф. Харків-Бахмут, 15 травня 2024 р. Т.2. С. 84-86. URL: https://www.nnppi.in.ua/index.php/abit/2-uncategorised/270-naukovi-konferentsiyi

5. Lally P., Wardle J. How are habits formed: modelling habit formation in the real world. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ejsp.674

6. Що таке функціональні вимоги: приклади, визначення, повний посібник. URL: https://visuresolutions.com/uk/блог/функціональні-вимоги/

7. Azure Active Directory is now Microsoft Entra ID. URL: https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id

8. Aadya Singh Azure DevOps Revolutionizing the Software Development Industry 23 грудня 2023 р. URL: https://www.linkedin.com/pulse/unplugged-azure-devops-revolutionizing-software-industry-aadya-singh--ymaxf/

9. Official Microsoft Azure Website. Azure Artifacts. URL: https://azure.microsoft.com/en-us/products/devops/artifacts

10. Ranking of the most popular relational database management systems worldwide. URL: https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/

11. Переваги платформи бізнес-аналітики SQL Server. URL: https://www.microsoft.com/uk-ua/sql-server/sql-business-intelligence/

12. Порівняння мови SQL у Access із мовою TSQL у SQL Server. URL: https://support.microsoft.com/uk-ua/topic/порівняння-мови-sql-y-access-із-мовою-tsql-y-sql-server-f09f180f-c005-4ff3-812f-14a5eb7902c8

13. What are the benefits of working with SQL Server through SSMS (SQL Server Management Studio) instead of working directly in the Command Line Interface? URL: https://www.quora.com/What-are-the-benefits-of-working-with-SQL-Server-through-SSMS-SQL-Server-Management-Studio-instead-of-working-directly-in-the-Command-Line-Interface

14. SQL Server Management Studio (SSMS): Everything to Know in 2024. URL: https://geekflare.com/sql-server-management-studio-ssms-guide/

15. Developer Survey 2023. URL: https://survey.stackoverflow.co/2023/

16. Why Choose ASP.NET for Mobile App Development? URL: https://www.linkedin.com/pulse/why-choose-aspnet-mobile-app-development-inestweb-v1zoc/

17. How To Set Up a New Project With React & Capacitor.js. URL: https://www.tiaeastwood.com/how-to-set-up-a-new-project-with-react-and-capacitor-js

18. The Global Startup Ecosystem Report 2022. URL: https://startupgenome.com/article/the-state-of-the-global-startup-economy

19. Rogers A. What Is SEO Marketing? Definition, Importance, and Types. 1 жовтня 2023 р. URL: https://www.shopify.com/blog/seo-marketing

20. Shrestha Matina What is the 3-Tier Architecture? 1 травня 2023 р. URL: https://medium.com/@shrestha.matina.20/what-is-the-3-tier-architecture-4520522e0720

21. Поліщук Ю.К. Шаблон проєктування MVC. URL: https://informatika.udpu.edu.ua/?page_id=5450

22. Реляційні бази даних: усе, що необхідно про них знати. URL: https://foxminded.ua/reliatsiini-bazy-danykh/

23. Каграманова Ю. Як будувати UML-діаграми. 27 жовтня 2022 р. URL: https://dou.ua/forums/topic/40575/

ДОДАТКИ

Додаток А. Фрагменти програмного коду

ApiController

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using Diploma.Models;
using Diploma.BLL.Interfaces;
using Diploma.DAL.Models;
using Microsoft.AspNetCore.Identity;
namespace Diploma.Controllers.Api;
[Route("api/[controller]")]
public class TestController : ControllerBase
{
    private readonly ILogger<TestController> _logger;
    private readonly IAssignmentService _assignmentService;
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly SignInManager<ApplicationUser> _signInManager;
    public TestController(ILogger<TestController> logger,
     IAssignmentService assignmentService,
      UserManager<ApplicationUser> userManager,
      SignInManager<ApplicationUser> signInManager)
    {
        _logger = logger;
        _assignmentService = assignmentService;
        _userManager = userManager;
        _signInManager = signInManager;
    }
    [HttpGet]
    [Route("index")]
    public IActionResult Index()
    {
        return Ok();
    }
    [HttpGet]
    [Route("error")]
    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore =
true)]
    public IActionResult Error()
    {
        return Ok(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
    }
    [HttpPost]
    [Route("createAssignment")]
    public async Task<IActionResult> CreateAssignment(Assignment assignment)
    {
        await _assignmentService.CreateAssignment(assignment);
        return Ok();
    }
    [HttpPost]
    [Route("register")]
    public async Task<IActionResult> Register(RegisterViewModel registerViewModel)
    {
        var user = new ApplicationUser()
        {
            Email = registerViewModel.Email,
            UserName = registerViewModel.UserName,
        };
```

```
        var result = await _userManager.CreateAsync(user, registerViewModel.Password);
        if (result.Succeeded)
        {
            await _signInManager.SignInAsync(user, false);
            return RedirectToAction(nameof (HomeController.Index), "Index");
        }
        else
        {
            return BadRequest("Error");
        }
    }
    [HttpPost]
    [Route("login")]
    public async Task<IActionResult> Login(LoginViewModel loginViewModel)
    {
        var result = await _signInManager.PasswordSignInAsync(loginViewModel.UserName,
loginViewModel.Password, false, false);
        if (result.Succeeded)
        {
            return Ok();
        }
        else
        {
            return NotFound();
        }
    }
    [HttpPost]
    [Route("logout")]
    public async Task<IActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return Ok();
    }
}
AssignmentsController
using Diploma.BLL.Interfaces;
using Diploma.DAL.Enums;
using Diploma.DAL.Models;
using Diploma.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.OutputCaching;
[OutputCache(NoStore = true)]
public class AssignmentsController : Controller
{
    private readonly IAssignmentService _assignmentService;
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly IAssignmentCategoryService _assignmentCategoryService;
    public AssignmentsController(IAssignmentService assignmentService,
    UserManager<ApplicationUser> userManager,
    IAssignmentCategoryService assignmentCategoryService)
    {
        _assignmentService = assignmentService;
        _userManager = userManager;
        _assignmentCategoryService = assignmentCategoryService;
    }
    public async Task<IActionResult> Index()
    {
        var user = await _userManager.GetUserAsync(User);
        var models = await _assignmentService.GetAssignments(3);
        var viewModels = models.Select(x => new AssignmentViewModel(x));
        return View(viewModels);
    }
```

```
    [HttpGet]
    public async Task<IActionResult> AddAssignment(int? templateId)
    {
        if (templateId.HasValue)
        {
           var assignmentCategories = await
_assignmentCategoryService.GetAssignmentCategories();
            var assignment = await _assignmentService.GetAssignmentById(templateId.Value);
             if (assignment == null)
             {
                 return NotFound();
             }
             var model = new AssignmentViewModel(assignment, true)
             {
                Categories = assignmentCategories.Select(x => new SelectListItem { Text =
x.Name, Value = x.Id.ToString() })
             };
             return View(model);
        }
        else
        {
           var assignmentCategories = await
_assignmentCategoryService.GetAssignmentCategories();
             var viewModel = new AssignmentViewModel()
             {
                 Categories = assignmentCategories.Select(x => new SelectListItem {
Text = x.Name, Value = x.Id.ToString() })
             };
             return View(viewModel);
        }
    }
    [HttpPost]
    public async Task<IActionResult> AddAssignment(AssignmentViewModel model)
    {
        if (ModelState.IsValid)
        {
             DateTimeOffset.TryParse(model.StartDate, out DateTimeOffset date);
             DateTimeOffset.TryParse(model.StartTime, out DateTimeOffset time);
             var startsAt = new DateTime(date.Year, date.Month, date.Day,
time.Hour, time.Minute, 0);
             var user = await _userManager.GetUserAsync(User);
             var assignment = new Assignment()
             {
                 Name = model.Name,
                 Description = model.Description,
                 Priority = (AssignmentPriority)model.Priority,
                 UserId = 3,
                 CategoryId = model.CategoryId,
                 IsTemplate = model.IsTemplate,
                 Notify = model.Notify,
                 Status = AssignmentStatus.New,
                 StartsAt = new DateTimeOffset(startsAt)
             };
             await _assignmentService.CreateAssignment(assignment);

             return Redirect("/");
        }
        else
        {
             var assignmentCategories = await
_assignmentCategoryService.GetAssignmentCategories();
             model.Categories = assignmentCategories.Select(x => new SelectListItem
{ Text = x.Name, Value = x.Id.ToString() });
             return View(model);
```

```
        }
    }
    [HttpGet]
    public async Task<IActionResult> EditAssignment(int id)
    {
        var assignmentCategories = await
_assignmentCategoryService.GetAssignmentCategories();
        var assignment = await _assignmentService.GetAssignmentById(id);
        if (assignment == null)
        {
            return NotFound();
        }
        var model = new AssignmentViewModel(assignment)
        {
            Categories = assignmentCategories.Select(x => new SelectListItem {
Text = x.Name, Value = x.Id.ToString() })
        };
        return View("AddAssignment", model);
    }
    [HttpPost]
    public async Task<IActionResult> EditAssignment(AssignmentViewModel model)
    {
        if (ModelState.IsValid)
        {
            DateTimeOffset.TryParse(model.StartDate, out DateTimeOffset date);
            DateTimeOffset.TryParse(model.StartTime, out DateTimeOffset time);
            var startsAt = new DateTime(date.Year, date.Month, date.Day,
time.Hour, time.Minute, 0);
            var user = await _userManager.GetUserAsync(User);
            var assignment = new Assignment()
            {
                Name = model.Name,
                Description = model.Description,
                Priority = (AssignmentPriority)model.Priority,
                UserId = 3,
                CategoryId = model.CategoryId,
                IsTemplate = model.IsTemplate,
                Notify = model.Notify,
                Status = AssignmentStatus.New,
                StartsAt = new DateTimeOffset(startsAt)
            };
            await _assignmentService.CreateAssignment(assignment);

            return Redirect("/");
        }
        else
        {
            var assignmentCategories = await
_assignmentCategoryService.GetAssignmentCategories();
            model.Categories = assignmentCategories.Select(x => new SelectListItem
{ Text = x.Name, Value = x.Id.ToString() });
            return View(model);
        }
    }
    public async Task<IActionResult> TemplateAssignments()
    {
        var user = await _userManager.GetUserAsync(User);
        var userTemplateAssignments = await
_assignmentService.GetTemplateAssignments(3);
        var defaultTemplateAssignments = await
_assignmentService.GetTemplateAssignments(null);
        return View(new TemplateAssignmentsViewModel(userTemplateAssignments,
defaultTemplateAssignments));
    }
```

```
    public IActionResult ChooseAddingAssignment()
    {
        return View();
    }
}
```

GrowingObjectController

```
using Diploma.BLL.Interfaces;
using Diploma.DAL.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
namespace Diploma.Controllers.Api;
[Route("api/[controller]")]
public class GrowingObjectController : ControllerBase
{
    private readonly IGrowingObjectService _growingObjectService;
    private readonly UserManager<ApplicationUser> _userManager;
    public GrowingObjectController(IGrowingObjectService growingObjectService,
UserManager<ApplicationUser> userManager)
    {
        _growingObjectService = growingObjectService;
        _userManager = userManager;
    }
    [HttpPost]
    public async Task<IActionResult> CreateGrowingObject(GrowingObject
growingObject)
    {
       var growingObject = await
_growingObjectService.CreateGrowingObject(growingObject);
        return Ok(growingObject);
    }
    [HttpGet]
    public async Task<IActionResult> GetGrowingObject()
    {
        var user = await _userManager.GetUserAsync(User);
        if (user == null)
        {
            return NotFound();
        }
        var growingObject = await _growingObjectService.GetGrowingObjects(user.Id);
        return Ok(growingObject);
    }
    [HttpPut]
    public async Task<IActionResult> UpdateGrowingObject(GrowingObject growingObject)
    {
        return Ok(await _growingObjectService.UpdateGrowingObject(growingObject));
    }
    [HttpDelete]
    public async Task<IActionResult> DeleteGrowingObject(int id)
    {
        return Ok(await _growingObjectService.DeleteGrowingObject(id));
    }
}
```

HomeController

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using Diploma.Models;
namespace Diploma.Controllers;
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;
    public HomeController(ILogger<HomeController> logger)
    {
```

```
        _logger = logger;
    }
    public IActionResult Index()
    {
        return View();
    }
    [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
    public IActionResult Error()
    {
        return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
    }
```

IAssignmentCategoryService

```
using Diploma.DAL.Models;
namespace Diploma.BLL.Interfaces;
public interface IAssignmentCategoryService
{
    Task<AssignmentCategory> CreateAssignmentCategory(AssignmentCategory
assignmentCategory);
    Task<IEnumerable<AssignmentCategory>> GetAssignmentCategories();
    Task<AssignmentCategory> UpdateAssignmentCategory(AssignmentCategory
assignmentCategory);
    Task<bool> DeleteAssignmentCategory(int id);
}
```

IAssignmentService

```
using Diploma.DAL.Enums;
using Diploma.DAL.Models;
namespace Diploma.BLL.Interfaces;
public interface IAssignmentService
{
    Task<Assignment> CreateAssignment(Assignment assignment);
    Task<IEnumerable<Assignment>> GetAssignments(int userId);
    Task<IEnumerable<Assignment>> GetTemplateAssignments(int? userId);
    Task<Assignment?> GetAssignmentById(int id);
    Task<Assignment> UpdateAssignment(Assignment assignment);
    Task<bool> DeleteAssignment(int id);
    Task<bool> SetTaskPriority(int assignmentId, AssignmentPriority priority);
    Task<bool> SetTaskStatus(int assignmentId, AssignmentStatus status);
    Task<Assignment?> ChooseTaskForUser(int assignmentId, int userId);
}
```

IGrowingObjectService

```
using Diploma.DAL.Models;
namespace Diploma.BLL.Interfaces;
public interface IGrowingObjectService
{
    Task<GrowingObject> CreateGrowingObject(GrowingObject growingObject);
    Task<IEnumerable<GrowingObject>> GetGrowingObjects(int userId);
    Task<GrowingObject> UpdateGrowingObject(GrowingObject growingObject);
    Task<bool> DeleteGrowingObject(int id);
}
```

IShopItemService

```
using Diploma.DAL.Models;
namespace Diploma.BLL.Interfaces;
public interface IShopItemService
{
    Task<ShopItem> CreateShopItem(ShopItem shopItem);
    Task<IEnumerable<ShopItem>> GetShopItems(int userId);
    Task<ShopItem> UpdateShopItem(ShopItem shopItem);
    Task<bool> DeleteShopItem(int id);
}
```

ApplicationUserService

```
using Diploma.BLL.Interfaces;
using Diploma.DAL.Models;
using Microsoft.AspNetCore.Identity;
namespace Diploma.BLL.Services;
public class ApplicationUserService : IApplicationUserService
{
    private readonly UserManager<ApplicationUser> _userManager;

    public ApplicationUserService(UserManager<ApplicationUser> userManager)
    {
        _userManager = userManager;
    }
}
```

AssignmentCategoryService

```
using Diploma.BLL.Interfaces;
using Diploma.DAL.Interfaces;
using Diploma.DAL.Models;
namespace Diploma.BLL.Services;
public class AssignmentCategoryService : IAssignmentCategoryService
{
    private readonly IAssignmentCategoryRepository _assignmentCategoryRepository;
    public AssignmentCategoryService(IAssignmentCategoryRepository
assignmentCategoryRepository)
    {
        _assignmentCategoryRepository = assignmentCategoryRepository;
    }
    public async Task<AssignmentCategory>
CreateAssignmentCategory(AssignmentCategory assignmentCategory)
    {
        await _assignmentCategoryRepository.CreateAsync(assignmentCategory);
        return assignmentCategory;
    }
    public async Task<IEnumerable<AssignmentCategory>> GetAssignmentCategories()
    {
        return await _assignmentCategoryRepository.GetAssignmentCategories();
    }
    public async Task<AssignmentCategory>
UpdateAssignmentCategory(AssignmentCategory assignmentCategory)
    {
        await _assignmentCategoryRepository.UpdateAsync(assignmentCategory);
        return assignmentCategory;
    }
    public async Task<bool> DeleteAssignmentCategory(int id)
    {
        return await _assignmentCategoryRepository.DeleteAsync(id);
    }
}
```

AssignmentService

```
using Diploma.BLL.Interfaces;
using Diploma.DAL.Enums;
using Diploma.DAL.Interfaces;
using Diploma.DAL.Models;
namespace Diploma.BLL.Services;
public class AssignmentService : IAssignmentService
{
    private readonly IAssignmentRepository _assignmentRepository;
    public AssignmentService(IAssignmentRepository assignmentRepository)
    {
        _assignmentRepository = assignmentRepository;
    }
    public async Task<Assignment> CreateAssignment(Assignment assignment)
    {
        await _assignmentRepository.CreateAsync(assignment);
```

```
    return assignment;
}
public async Task<IEnumerable<Assignment>> GetAssignments(int userId)
{
    return await _assignmentRepository.GetAssignmentsByUserId(userId);
}
public async Task<Assignment?> GetAssignmentById(int id)
{
    return await _assignmentRepository.GetAssignment(id);
}
public async Task<Assignment> UpdateAssignment(Assignment assignment)
{
    await _assignmentRepository.UpdateAsync(assignment);
    return assignment;
}
public async Task<bool> DeleteAssignment(int id)
{
    return await _assignmentRepository.DeleteAsync(id);
}
public async Task<IEnumerable<Assignment>> GetTemplateAssignments(int? userId)
{
   var assignments = await _assignmentRepository.GetAssignmentsByUserId(userId);
    return assignments.Where(assignment => assignment.IsTemplate);
}
public async Task<bool> SetTaskPriority(int assignmentId, AssignmentPriority priority)
{
    var assignment = await _assignmentRepository.GetAssignment(assignmentId);
    if (assignment == null)
    {
        return false;
    }
    assignment.Priority = priority;
    await _assignmentRepository.UpdateAsync(assignment);
    return true;
}
public async Task<bool> SetTaskStatus(int assignmentId, AssignmentStatus status)
{
    var assignment = await _assignmentRepository.GetAssignment(assignmentId);
    if (assignment == null)
    {
        return false;
    }
    assignment.Status = status;
    await _assignmentRepository.UpdateAsync(assignment);
    return true;
}
public async Task<Assignment?> ChooseTaskForUser(int assignmentId, int userId)
{
   var genericAssignment = await _assignmentRepository.GetAssignment(assignmentId);
    if (genericAssignment == null)
    {
        return null;
    }
    var userAssignment = new Assignment
    {
        Name = genericAssignment.Name,
        Description = genericAssignment.Description,
        Status = AssignmentStatus.InProgress,
        Priority = AssignmentPriority.High,
        UserId = userId
    };
    await _assignmentRepository.CreateAsync(userAssignment);
    return userAssignment;
}
```

```
    public async Task<IEnumerable<Assignment>> FilterTasksByCategory(int userId,
int categoryId)
    {
        var assignments = await _assignmentRepository.GetAssignmentsByUserId(userId);
        var filteredAssignments = assignments.Where(assignment => assignment.CategoryId ==
categoryId);
        return filteredAssignments;
    }
}
```

GrowingObjectService

```
using Diploma.BLL.Interfaces;
using Diploma.DAL.Interfaces;
using Diploma.DAL.Models;
namespace Diploma.BLL.Services;
public class GrowingObjectService : IGrowingObjectService
{
    private readonly IGrowingObjectRepository _growingObjectRepository;
    public GrowingObjectService(IGrowingObjectRepository growingObjectRepository)
    {
        _growingObjectRepository = growingObjectRepository;
    }
    public async Task<GrowingObject> CreateGrowingObject(GrowingObject growingObject)
    {
        await _growingObjectRepository.CreateAsync(growingObject);
        return growingObject;
    }
    public async Task<IEnumerable<GrowingObject>> GetGrowingObjects(int userId)
    {
        return await _growingObjectRepository.GetGrowingObjectsByUserId(userId);
    }
    public async Task<GrowingObject> UpdateGrowingObject(GrowingObject growingObject)
    {
        await _growingObjectRepository.UpdateAsync(growingObject);
        return growingObject;
    }
    public async Task<bool> DeleteGrowingObject(int id)
    {
        return await _growingObjectRepository.DeleteAsync(id);
    }
}
```

ShopItemService

```
using Diploma.BLL.Interfaces;
using Diploma.DAL.Interfaces;
using Diploma.DAL.Models;
namespace Diploma.BLL.Services;
public class ShopItemService : IShopItemService
{
    private readonly IShopItemRepository _shopItemRepository;
    public ShopItemService(IShopItemRepository shopItemRepository)
    {
        _shopItemRepository = shopItemRepository;
    }
    public async Task<ShopItem> CreateShopItem(ShopItem growingObject)
    {
        await _shopItemRepository.CreateAsync(growingObject);
        return growingObject;
    }
    public async Task<IEnumerable<ShopItem>> GetShopItems(int userId)
    {
        return await _shopItemRepository.GetShopItemsByUserId(userId);
    }
    public async Task<ShopItem> UpdateShopItem(ShopItem growingObject)
```

```
    {
        await _shopItemRepository.UpdateAsync(growingObject);
        return growingObject;
    }
    public async Task<bool> DeleteShopItem(int id)
    {
        return await _shopItemRepository.DeleteAsync(id);
    }
}
```

IAssignmentCategoryRepository

```
using Diploma.DAL.Models;
namespace Diploma.DAL.Interfaces;
public interface IAssignmentCategoryRepository
{
    Task<AssignmentCategory> CreateAsync(AssignmentCategory assignmentCategory);
    Task<IEnumerable<AssignmentCategory>> GetAssignmentCategories();
    Task<AssignmentCategory?> GetAssignmentCategory(int assignmentCategoryId);
    Task UpdateAsync(AssignmentCategory assignmentCategory);
    Task<bool> DeleteAsync(int id);
}
```

IAssignmentRepository

```
using Diploma.DAL.Models;
namespace Diploma.DAL.Interfaces;
public interface IAssignmentRepository
{
    Task<Assignment> CreateAsync(Assignment assignment);
    Task<IEnumerable<Assignment>> GetAssignmentsByUserId(int? userId);
    Task<Assignment?> GetAssignment(int assignmentId);
    Task UpdateAsync(Assignment assignment);
    Task<bool> DeleteAsync(int id);
}
```

IGrowingObjectRepository

```
using Diploma.DAL.Models;
namespace Diploma.DAL.Interfaces;
public interface IGrowingObjectRepository
{
    Task<GrowingObject> CreateAsync(GrowingObject growingObject);
    Task<IEnumerable<GrowingObject>> GetGrowingObjectsByUserId(int userId);
    Task<GrowingObject?> GetGrowingObject(int growingObjectId);
    Task UpdateAsync(GrowingObject growingObject);
    Task<bool> DeleteAsync(int id);
}
```

IShopItemRepository

```
using Diploma.DAL.Models;
namespace Diploma.DAL.Interfaces;
public interface IShopItemRepository
{
    Task<ShopItem> CreateAsync(ShopItem shopItem);
    Task<IEnumerable<ShopItem>> GetShopItemsByUserId(int userId);
    Task<ShopItem?> GetShopItem(int shopItemId);
    Task UpdateAsync(ShopItem shopItem);
    Task<bool> DeleteAsync(int id);
}
```

AssignmentCategoryRepository

```
using Diploma.DAL.Context;
using Diploma.DAL.Interfaces;
using Diploma.DAL.Models;
using Microsoft.EntityFrameworkCore;
namespace Diploma.DAL.Repositories;
```

```
public class AssignmentCategoryRepository : IAssignmentCategoryRepository
{
    private readonly DiplomaDbContext _context;

    public AssignmentCategoryRepository(DiplomaDbContext context)
    {
        _context = context;
    }
    public async Task<AssignmentCategory> CreateAsync(AssignmentCategory category)
    {
        _context.AssignmentCategories.Add(category);
        await _context.SaveChangesAsync();
        return category;
    }
    public async Task<IEnumerable<AssignmentCategory>> GetAssignmentCategories()
    {
       return await _context.AssignmentCategories.IgnoreAutoIncludes().ToListAsync();
    }
    public async Task<AssignmentCategory?> GetAssignmentCategory(int categoryId)
    {
       return await _context.AssignmentCategories.FirstOrDefaultAsync(x => x.Id ==
categoryId);
    }
    public async Task UpdateAsync(AssignmentCategory category)
    {
        _context.AssignmentCategories.Update(category);
        await _context.SaveChangesAsync();
    }
    public async Task<bool> DeleteAsync(int id)
    {
        var category = await GetAssignmentCategory(id);
        if (category != null)
        {
            _context.AssignmentCategories.Remove(category);
            await _context.SaveChangesAsync();
            return true;
        }
        return false;
    }
}
```

AssignmentRepository

```
using Diploma.DAL.Context;
using Diploma.DAL.Interfaces;
using Diploma.DAL.Models;
using Microsoft.EntityFrameworkCore;

namespace Diploma.DAL.Repositories;

public class AssignmentRepository : IAssignmentRepository
{
    private readonly DiplomaDbContext _context;
    public AssignmentRepository(DiplomaDbContext context)
    {
        _context = context;
    }
    public async Task<Assignment> CreateAsync(Assignment assignment)
    {
        _context.Assignments.Add(assignment);
        await _context.SaveChangesAsync();
        return assignment;
    }
    public async Task<IEnumerable<Assignment>> GetAssignmentsByUserId(int? userId)
```

```
    {
        return await _context.Assignments.Where(x => x.UserId ==
userId).IgnoreAutoIncludes().ToListAsync();
    }
    public async Task<Assignment?> GetAssignment(int assignmentId)
    {
        return await _context.Assignments.FirstOrDefaultAsync(x => x.Id == assignmentId);
    }
    public async Task UpdateAsync(Assignment assignment)
    {
        _context.Assignments.Update(assignment);
        await _context.SaveChangesAsync();
    }
    public async Task<bool> DeleteAsync(int id)
    {
        var assignment = await GetAssignment(id);
        if (assignment != null)
        {
            _context.Assignments.Remove(assignment);
            await _context.SaveChangesAsync();
            return true;
        }
        return false;
    }
}
```

GrowingObjectRepository

```
using Diploma.DAL.Context;
using Diploma.DAL.Interfaces;
using Diploma.DAL.Models;
using Microsoft.EntityFrameworkCore;

namespace Diploma.DAL.Repositories;

public class GrowingObjectRepository : IGrowingObjectRepository
{
    private readonly DiplomaDbContext _context;
    public GrowingObjectRepository(DiplomaDbContext context)
    {
        _context = context;
    }
    public async Task<GrowingObject> CreateAsync(GrowingObject growingObject)
    {
        _context.GrowingObjects.Add(growingObject);
        await _context.SaveChangesAsync();
        return growingObject;
    }
    public async Task<IEnumerable<GrowingObject>> GetGrowingObjectsByUserId(int userId)
    {
        return await _context.GrowingObjects.Where(x => x.UserId ==
userId).IgnoreAutoIncludes().ToListAsync();
    }
    public async Task<GrowingObject?> GetGrowingObject(int growingObjectId)
    {
        return await _context.GrowingObjects.FirstOrDefaultAsync(x => x.Id ==
growingObjectId);
    }
    public async Task UpdateAsync(GrowingObject growingObject)
    {
        _context.GrowingObjects.Update(growingObject);
        await _context.SaveChangesAsync();
    }
    public async Task<bool> DeleteAsync(int id)
    {
```

```
        var growingObject = await GetGrowingObject(id);
        if (growingObject != null)
        {
            _context.GrowingObjects.Remove(growingObject);
            await _context.SaveChangesAsync();
            return true;
        }
        return false;
    }
}
```

ShopItemRepository

```
using Diploma.DAL.Context;
using Diploma.DAL.Interfaces;
using Diploma.DAL.Models;
using Microsoft.EntityFrameworkCore;

namespace Diploma.DAL.Repositories;

public class ShopItemRepository : IShopItemRepository
{
    private readonly DiplomaDbContext _context;
    public ShopItemRepository(DiplomaDbContext context)
    {
        _context = context;
    }
    public async Task<ShopItem> CreateAsync(ShopItem shopItem)
    {
        _context.ShopItems.Add(shopItem);
        await _context.SaveChangesAsync();
        return shopItem;
    }
    public async Task<IEnumerable<ShopItem>> GetShopItemsByUserId(int userId)
    {
       return await _context.ShopItems.Where(x => x.UserId ==
userId).IgnoreAutoIncludes().ToListAsync();
    }
    public async Task<ShopItem?> GetShopItem(int shopItemId)
    {
       return await _context.ShopItems.FirstOrDefaultAsync(x => x.Id == shopItemId);
    }
    public async Task UpdateAsync(ShopItem shopItem)
    {
        _context.ShopItems.Update(shopItem);
        await _context.SaveChangesAsync();
    }
    public async Task<bool> DeleteAsync(int id)
    {
        var shopItem = await GetShopItem(id);
        if (shopItem != null)
        {
            _context.ShopItems.Remove(shopItem);
            await _context.SaveChangesAsync();
            return true;
        }
        return false;
    }
}
```

Program.cs

```
using Diploma.BLL.Interfaces;
using Diploma.BLL.Services;
using Diploma.DAL.Context;
using Diploma.DAL.Interfaces;
```

```
using Diploma.DAL.Models;
using Diploma.DAL.Repositories;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddScoped<IAssignmentRepository, AssignmentRepository>();
builder.Services.AddScoped<IAssignmentService, AssignmentService>();
builder.Services.AddScoped<IGrowingObjectRepository, GrowingObjectRepository>();
builder.Services.AddScoped<IGrowingObjectService, GrowingObjectService>();
builder.Services.AddScoped<IShopItemRepository, ShopItemRepository>();
builder.Services.AddScoped<IShopItemService, ShopItemService>();
builder.Services.AddScoped<IAssignmentCategoryRepository,
AssignmentCategoryRepository>();
builder.Services.AddScoped<IAssignmentCategoryService, AssignmentCategoryService>();
// Add services to the container.
builder.Services.AddDbContext<DiplomaDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
builder.Services.AddIdentity<ApplicationUser, IdentityRole<int>>()
    .AddEntityFrameworkStores<DiplomaDbContext>();
builder.Services.AddControllersWithViews()
    .AddNewtonsoftJson(options =>
        options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore
    )
    .AddRazorRuntimeCompilation();
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
var app = builder.Build();
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var context = services.GetRequiredService<DiplomaDbContext>();
    context.Database.Migrate();
}
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production
scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}
else
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();
app.MapControllers();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
app.Run();
```

Додаток Б. Копії документів про апробацію результатів роботи

РОЗДІЛ 1.
ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ФУНКЦІОНУВАННЯ КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Антіпіна Надія

Національний університет «Одеська юридична академія»

студентка 3 курсу факультету кібербезпеки та інформаційних технологій

ТЕХНОЛОГІЇ РОЗРОБКИ ІГРОВИХ ЗАСТОСУНКІВ ЗАСОБАМИ UNITY

Комп'ютерні ігри посідають значне місце у списку розваг сучасної онлайн-аудиторії, що зумовлює невпинний розвиток сектору їх розробки. Доволі популярними є ігри-слешери від третьої дійової особи. Мільйони користувачів Steam віддають перевагу саме такому жанру [1].

Поширеними мовами розробки ігор є C++ та C# з відповідними платформами Unreal Engine та Unity 3D [2].

Розробників приваблює використання Unity через низку переваг [3]:

- легкість вивчення, інтуїтивно зрозумілий інтерфейс платформи;
- підтримка мови C#, яка, з одного боку, має простий синтаксис, а, з іншого, є об'єктно-орієнтованою та має вбудований прибиральник сміття;
- наявність бібліотеки численних плагінів, безкоштовних асетів з моделями персонажів та анімаціями, які суттєво прискорюють розробку;
- інструменти для створення складних анімацій і реалістичної фізики, що дозволяє створювати максимально наближену до реальності графіку, тощо.

Для розроблення гри у жанрі slasher треба створити такі компоненти:

- сітка навігації та карти місцевості;
- налаштування руху камери;
- налаштування анімацій, руху та атак агентів;
- функціонал ворожих персонажів.

Сітка навігації (NavMesh). Інструментарій Unity пропонує компоненти навігаційної сітки з широким обсягом можливостей. Саме навігаційна сітка визначає, по яких поверхнях ігровий персонаж (агент) пересуватиметься. Розробнику при конструюванні NavMesh треба вибрати предмети оточення (шари, англ. layers), які будуть дійовим полем персонажа, і коректно задати частину оточення, що не є доступною для його пересування (стіни, паркани, дерева, перепони тощо). Модифікатор навігаційної сітки надає можливість змінювати її параметри стосовно окремих об'єктів. Його параметр overall area (загальна площа) дозволяє задавати тип області доступності певного об'єкта: walkable, not walkable, jump тощо. Існують також налаштування параметрів агента: розмір моделі, радіус дії, можливість підійматися по кутових поверхнях.

Створення контролерів руху гравця та камери. Unity має пакет cinemachine для налаштування інтерактивного руху камери. Його використання дозволяє легко змоделювати складну поведінку камери без написання коду. Параметр world space прив'язує камеру до дійового простору гри. За допомогою покажчика миші користувач зможе рухати камеру по трьох орбітах і мати тривимірний контроль перегляду дій персонажа. Для надання можливості переміщення персонажа потрібно створити відповідний контролер і скрипт керування. Контролер можна розглядати як двигун, який приводить в рух персонажа, а написаний код дає вказівку, куди саме йти. Обов'язковим є створення функції update(), яка збирає дані рухів по горизонталі та по вертикалі засобами натискання клавіш WASD або стрілок. Переміщення здійснюється за формулою, що враховує значення напрямку, швидкості і часу. Після виконання відповідних математичних операцій персонаж може пересуватися вперед, назад чи то вбік залежно від того, як користувач направлятиме камеру [4]. При проходженні камери крізь непрозорі об'єкти, наприклад стіни, для уникнення блокування відображення гравця використовують розширення cinemachine collider, яке перевіряє, чи стикається камера зараз з чимось і відсуває її від цього об'єкта.

Механізм бою гравця. Скрипт задає параметри анімації (швидкість, тригери для атак тощо) персонажа, а відповідний, пов'язаний з цим скриптом аніматор

– характер поведінки ігрових персонажів. Так, у функції рухів update() є умова, яка при натисканні користувачем клавіш атаки запускає метод Attack(). Створення цього методу складається з трьох базових етапів: відтворити анімацію атаки, визначити ворожі об'єкти в радіусі атаки, завдати їм удар.

Одним зі складових інтерфейсу Unity є вікно аніматора, в якому розміщений і контролер анімації. У вікні аніматора створюються основні компоненти стану персонажа (entry, any state, exit), а також сюди можна додати наявні анімації. Щоб встановити усталено анімацію очікування з моменту запуску гри, її слід зв'язати зі станом входу (entry). Для використання інших анімацій потрібно створити переходи за умовою перевірки параметрів 4 типів: float, bool, int, trigger. Наприклад, для входу і виходу з анімацію руху відбувається перевірка параметру швидкості типу float, значення якого задається в скрипті. Подібним чином відбувається перехід зі стану очікування у стан бігу, стрибка чи то польоту.

Будь-який стан (any state) після активування може переривати поточну анімацію. Для виконання атаки перевіряється умова щоразу, коли в коді запускається виконання функції Attack(). При цьому прописаний всередині неї тригер вмикається і з'являється анімація удару. Після закінчення атаки тригер вимикається й анімація зникає [5].

Щоб програма розрізняла ворожі об'єкти (персонажі, бази), вони повинні належати до створених для них шарів. Програмно у функції атаки викликається метод OverlapCirleAll() для 2D (або OverlapSphereAll для 3D). Відповідний метод створює коло (чи то сферу) з вхідними параметрами: положення точки атаки, радіус атаки та назва шару з ворожими об'єктами. Метод «збирає» всі об'єкти, що потрапляють у радіус атаки, помічає ворогів і формує відповідний масив цілей для ударів. За уникнення великої серії (спаму) атак встановлюється обмеження на їхню кількість та час.

Окремі скрипти відповідають за стан здоров'я головного героя та стан здоров'я ворогів. Вони враховують значення змінних, як факторів здоров'я. Максимальне їхнє значення при завантаженні рівня гри зменшується при враженнях.

Коли показник здоров'я знизиться до нуля, спрацює функція відтворення анімації смерті та ліквідація ворога як активного ігрового об'єкта [6].

Створення функціоналу ворожих об'єктів. Засобами Unity можна наділити ворогів такими функціями:

- *патрулювання*, коли гравець далеко;
- *переслідування*, коли гравець потрапляє в зону бачення;
- *атака*, як тільки гравець потрапляє в зону атаки.

Для керування агентами NavMesh потрібно імпортувати у скрипт модуль UnityEngine.AI. Першочерговою задачею ворогів є пошук гравця засобами методу Find(). За допомогою Physics.checkSphere ворожі юніти перевіряють наявність гравця в заданих радіусах зору та атаки і, якщо їх там немає, викликається функція патрулювання. Якщо гравець є у полі зору, але він не у зоні атаки, викликається функція переслідування. Якщо гравець є в обох діапазонах, запускається функція атаки. У *функції патрулювання* генерується випадкове значення кінцевої точки маршруту в заданому діапазоні для осей X і Z. Метод SetDestination() через параметр позиції допомагає NavMesh агентам знайти найкоротший шлях. Після досягнення пункту призначення генерується нова точка для персонажа. *Функція переслідування* використовує той самий метод SetDestination(), але з аргументом позиції гравця як кінцевої цілі. *Функція атаки* забезпечує зупинку ворога перед здійсненням атаки, встановлення інтервалу між ударами та перевірку того, чи відбулася атака [7].

Отже, розробка ігор в середовищі Unity є доволі цікавою і динамічною, завдяки широкому набору засобів. До речі, 50% всіх двовимірних і тривимірних мобільних ігор створено саме в Unity [2]. Є безплатна версія Unity та база безкоштовних ресурсів, створених ентузіастами цієї спільноти.

Ігровий рушій Unity3D є зручним для роботи з 3D-простором, а інтуїтивно зрозумілий інтерфейс значно спрощує процес знайомства з платформою для розробників-початківців. Підключення власних скриптів, написаних мовою C#, допомагає розширити функціонал гри та зробити якісний продукт для потенційного споживання.

Список використаних джерел:

1. Most played Third-Person Slasher games URL: https://steamdb.info/graph/?tagid=3814.

2. Найкращі мови програмування ігор. URL: https://ciksiti.com/uk/chapters/5613-the-15-best-programming-languages-for-games-development.

3. Official Unity website. URL: https://unity.com.

4. Create Third-Person Controller in Unity! URL: https://medium.com/eincode/create-third-person-controller-in-unity-838809d1a4da.

5. Animation controllers in Unity. URL: https://learn.unity.com/tutorial/animator-controllers.

6. Modular Health System in Unity. URL: https://medium.com/nerd-for-tech/tip-of-the-day-modular-health-system-unity-8f5d2f187027.

7. Full 3D enemy AI in 6 minutes. URL: https://youtu.be/UjkSFoLxesw.

Ключові слова: розробка ігор, середовище Unity3D

Key words: game development, environment Unity3D

***Науковий керівник**: к.т.н., доцент Трофименко О.*

Міністерство освіти і науки України
Українська інженерно-педагогічна академія
Харківський національний університет ім. В. Н. Каразіна
Навчально-науковий професійно-
педагогічний інститут УІПА (м. Бахмут)
Дубницький інститут технологій в Дубниці над Вахом
Тбіліський державний університет ім. Іване Джавахішвілі
Азербайджанський технологічний університет
Кременчуцький національний університет ім. Михайла Остроградського
Інститут проблем машинобудування ім. А. М. Підгорного
Національної академії наук України

АКТУАЛЬНІ ТЕНДЕНЦІЇ РОЗВИТКУ ОСВІТИ, НАУКИ ТА ТЕХНОЛОГІЙ

Матеріали VII Міжнародної науково-практичної конференції

Частина 2

15 травня 2024 року

м. Бахмут, м. Харків

фотореалістичною графікою та складними ігровими механіками, варто розглянути Unreal Engine. Але якщо потрібна простота та легкість вивчення, Godot може стати найкращим другом у світі розробки ігор.

Список використаних джерел

1. Digital twin applications and use cases. URL: https://unity.com/solutions/digital-twin-applications-and-use-cases.
2. Aladdin J. Unity for Non-Gamers: Exploring Applications in Different Fields. URL: https://medium.com/@Jamal_Aladdin/unity-for-non-gamers-exploring-applications-in-different-fields-d42c06170586.
3. Visual Studio C# integration. URL: https://docs.unity.cn/530/Documentation/Manual/VisualStudioIntegration.html.
4. Unity vs Unreal Engine: pros and cons. URL: https://kevurugames.com/blog/unity-vs-unreal-engine-pros-and-cons/.
5. Степорук Н. Unity проти Unreal Engine – який рушій обрати для гри та чому. URL: https://gamedev.dou.ua/articles/unity-or-unreal-engine/.
6. Creating Games with Godot. URL: https://school.gdquest.com/products/learn_to_code_with_godot_3.
7. Berezovskyi O. Godot vs Unity 2022. URL: https://pinglestudio.com/blog/full-cycle-development/godot-vs-unity-2022.

МАРКЕТИНГОВІ ІНСТРУМЕНТИ В СТВОРЕННІ ТА ПРОСУВАННІ ПРОГРАМНИХ ПРОДУКТІВ

Трофименко О. Г., к.т.н., доц.
Антіпіна Н. С.
Національний університет «Одеська юридична академія» (м. Одеса)

Згідно з даними досліджень Startup Genome [1], понад 70% провалів у втіленні створених продуктів на ринок трапляються через неправильно підібрані маркетингові інструменти, відсутність визначеної стратегії та фінансові проблеми. Маркетинг продуктів потребує комплексного підходу і ретельного довгострокового планування.

Важливими складовими у створенні і просуванні програмних продуктів є аналіз і визначення цільової аудиторії та бізнес-цілей, аналіз ринку на продукти-аналоги та попит на товар, а також технології і засоби зв'язку, що використовують конкуренти. Цільова аудиторія стосується потенційних споживачів створюваного програмного продукту.

Визначення цільової вікової групи користувачів створюваного програмного забезпечення дозволяє краще зрозуміти клієнта, виявити та реалізувати його потенційні запити й очікування. Завдяки функції таргетування, програмні продукти спрямовані на задоволеність користувачів і збільшення їхньої аудиторії. Тобто правильне націлювання на потенційного користувача забезпечує успіх відповідного програмного продукту. Так, підхід «обізнаність і лояльність» (Awareness & Loyalty) спрямовує на підвищення рівня обізнаності про продукт та збільшення кількості лояльної до нього аудиторії [2].

Використання платформ соціальних мереж є важливим для підвищення пізнаванності бренду та лояльності до нього. Оскільки мільярди користувачів активно задіяні на таких платформах, як Facebook, Instagram і Twitter, бренди мають унікальну можливість швидко зв'язатися зі своєю цільовою аудиторією. Маркетинг у соціальних мережах (Social Media Marketing, SMM) також відомий як цифровий маркетинг та електронний маркетинг) стосується використання соціальних мереж, на яких користувачі обмінюються інформацією, для розширення цільової аудиторії програмного продукту та збільшення відвідуваності вебсайту [3]. Окрім того, що SMM надає компаніям можливість взаємодіяти з наявними клієнтами та залучати нових, він має спеціальну аналітику даних, яка дозволяє маркетологам відстежувати успіх їхніх зусиль і визначати ще більше способів залучення.

Контент-стратегія в соціальних мережах має бути зосередженою на візуальну складову та розважальний контент. Участь у змістовних бесідах, поширення релевантного вмісту та оперативні відповіді на запити клієнтів можуть створити сильну присутність в Інтернеті та зміцнити довіру до продукту. Крім того, створення візуально привабливого вмісту, яким можна поділитися, допомагає розширити потенційну цільову аудиторію й залучити нових підписників. Успішні кампанії в соціальних мережах зазвичай мають інтерактивні елементи: конкурси, опитування та створений користувачами контент, що спрямовані на заохочення активної участі і зміцнення зв'язків між брендом і його підписниками [2].

Якщо цільовою аудиторією застосунку є молоде покоління, то доречним може бути поширення створюваного програмного застосунку через платформу Tiktok, що дозволить відрізнятися від конкурентів. Специфіка просування відео в цій соціальній мережі потребує врахування тривалості роликів, частоти викладання та активності, добору гештегів та музики, а також легкого та ненав'язливого формату контенту.

При створенні вебсайту продукту має враховуватися SEO (Search Engine Optimization) – його оптимізація під пошукові запити користувача. Оскільки переважна більшість запитів обробляється саме в Google, варто орієнтуватися на критерії просування результатів пошуку перших сторінок саме для цієї пошукової системи, щоб користувачі з більшою вірогідністю побачили продукт, перейшли на сайт та виконали цільову дію (завантажили мобільний застосунок). Алгоритм Google враховує якість контенту, ключові слова, мета-теги, заголовки та підзаголовки, оптимізований медіа-контент, відгуки, посилання вебсторінок, а також мову та локацію користувача [4]. У разі грамотного використання SEO можливо суттєво збільшити залученість користувачів та зайняти вигідну позицію на ринку.

Співпраця з іншими брендами та компаніями є ще одним дієвим методом у збільшенні охоплення потенційної цільової аудиторії, залучення нових користувачів та підвищенні довіри до застосунку. А зазначення партнерами прямих посилань на продукт також впливатиме на SEO. Прикладом для створюваного програмного продукту може бути колаборація з популярними контент-мейкерами аудиторії: організація обмежених за часом подій (івентів) з

підвищеним обсягом нагород за виконання завдань у застосунку, поява лімітованих об'єктів вирощування або їхніх аксесуарів від партнерів тощо.

Ще однією важливою складовою є визначення способу монетизації продукту – його бізнес-моделі. Наразі поширеною й ефективною є модель Freemium (походить від free – «вільний» і premium – «дорогий»), яка поєднує в собі безкоштовне користування базовими функціями застосунку та преміум-підписку, що надає платний доступ з розширеним переліком функцій та переваг. Цінова політика таких підписок суттєво залежить від доступних функцій з можливістю спробувати кількаденну безкоштовну преміум-версію. Модель Freemium добре позначається на просуванні: вона приваблює клієнтів без відчутних вкладень у рекламу. У вартість продукту закладаються не витрати на рекламу, а витрати на поширення безплатної версії [5].

Зазвичай рука об руку з пошуковою оптимізацією та маркетингом у соціальних мережах йде стратегія контент-маркетингу. Як показують дослідження [6], створення контенту є найефективнішою маркетинговою стратегією, оскільки дозволяє зміцнити лояльність до бренду та залучити більше потенційних клієнтів. Завоювання довіри цільової аудиторії відбувається шляхом створення відповідного контенту, який спонукає потенційних клієнтів до використання програмного продукту. Контент охоплює цільові сторінки у соціальних мережах, блоги, відео, подкасти та інфографіку на теми, пов'язані з галуззю створюваного програмного продукту.

Як і будь-яка частина успішного бізнесу, маркетинг програмного продукту має бути зосереджений на клієнті. Успішна маркетингова стратегія складається з трьох компонентів: розуміння цільової аудиторії, знання своїх конкурентів і просування. Це означає, що поки існує програмний продукт потрібно постійно займатися його підтримкою, перевіряти відгуки клієнтів, оновлювати та покращувати функціонал.

Список використаних джерел

1. The Global Startup Ecosystem Report 2022. URL: https://startupgenome.com/article/the-state-of-the-global-startup-economy.

2. Cultivating Brand Loyalty: Strategies that Drive Awareness and Engagemen. URL: https://aicontentfy.com/en/blog/cultivating-brand-loyalty-strategies-that-drive-awareness-and-engagement.

3. Social Media Marketing (SMM): What It Is, How It Works, Pros and Cons. URL: https://www.linkedin.com/pulse/social-media-marketing-smm-what-how-works-pros-cons-webmediatricks-gp9pf.

4. Rogers A. What Is SEO Marketing? Definition, Importance, and Types. URL: https://www.shopify.com/blog/seo-marketing.

5. Фріміум: як заробляють на безплатних версіях. URL: https://fractus.com.ua/uk/blog/frimium-yak-zaroblyajut-na-bezplatnih-versiyah/.

6. How to Build a Product Marketing Strategy for Your Software Solution. URL: https://www.altexsoft.com/blog/product-marketing-strategy/.