ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Гнучке управління розробкою вебзастосунку для онлайн-запису до барбершопу «Barbercheck»

Ступінь вищої освіти – магістр

Спеціальність – 073 «Менеджмент»

Освітня програма «Agile-технології розробки програмного забезпечення»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Керівники: завідувач кафедрою комп'ютерних наук, доцент, с.н.с., к.е.н.
Сергій МІЧКІВСЬКИЙ
викладач кафедри інформаційного менеджменту, математики та статистики
Олег МУШИНСЬКИЙ

Виконав: здобувач
групи МЕН/Agile-24м
Андрій ТИМЧЕНКО

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.
Здобувач: _____
(підпис)

Київ, 2026 р.

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри інформаційного менеджменту, математики та статистики
_____ Денис БАЛДИК
«___»_____20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Тимченко Андрій Вікторович

Тема роботи	Гнучке управління розробкою вебзастосунку для онлайн-запису до барбершопу «Barbercheck»
Номер та дата наказу про затвердження теми	№ 109-2 від 14 жовтня 2025 року р.
Коротка постановка завдання	Обґрунтування бачення створюваного продукту для розв'язання проблеми в діяльності замовника на основі розробки моделі його організації. Детальний опис особливостей гнучкого управління розробкою вебзастосунку для онлайн-запису до барбершопу «Barbercheck». Розкриття застосування управління зануннями в agile.
Посилання на джерела інформації (не більше п'яти найменувань, які рекомендує науковий керівник)	1. Мушинський Олег. Роль цифрової трансформації в управління знаннями. Сучасні теорія і практика менеджменту та бізнес-адміністрування: збірник тез доповідей VI Всеукраїнської науково-практичної конференції (м. Черкаси, 25 травня 2022 р.). Черкаси: ЧДТУ, 2022. С.166-167 2. Благодир, О., & Лемішовська, О. (2024). Особливості реалізації бізнес процесів суб'єктів господарювання в умовах цифрової трансформації та їх вплив на організацію обліку. *Економіка та суспільство*, (61). https://doi.org/10.32782/2524-0072/2024-61-132
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має містити теоретичне та/або практичне дослідження за темою роботи, яку слід розглядати як складне спеціалізоване завдання або практичну проблематику в галузі управління та адміністрування, яка характеризується комплексністю та невизначеністю умов і потребує застосування Agile-технологій.

Дата видачі завдання «16» жовтня 2025 р.

Керівник Сергій МІЧКІВСЬКИЙ

Керівник Олег МУШИНСЬКИЙ

Здобувач Андрій ТИМЧЕНКО

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
Підготовчий етап			
1	Вибір напряму дослідження та керівника.	01.09.2025 р.	*виконано*
2	Формування теми та призначення керівника.	22.09.2025 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи.	14.10.2025 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу.	16.10.2025 р.	*виконано*
Основний етап			
5	Розробка концепції та змісту кваліфікаційної роботи, погодження їх з науковим керівником	06.11.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження.	08.11.2025 р.	*виконано*
7	Теоретико-методичний аналіз предметної області. Підготовка та подання керівнику розділу 1 кваліфікаційної роботи.	13.11.2025 р.	*виконано*
8	Реалізація гнучкого управління розробкою продукту. Підготовка та подання керівнику розділу 2 кваліфікаційної роботи.	20.11.2025 р.	*виконано*
9	Розробка рекомендацій щодо вдосконалення управління із застосуванням Agile-технологій. Підготовка та подання керівнику розділу 3 кваліфікаційної роботи.	27.11.2025 р.	*виконано*
10	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи.	01.12.2025 р.	*виконано*
11	Доопрацювання кваліфікаційної роботи з урахуваннямзауважень керівника та представлення керівнику доопрацьованого варіанту кваліфікаційної роботи	03.12.2025 р.	*виконано*
Завершальний етап			
12	Представлення рукопису для перевірки на плагіат.	08.12.2025 р.	*виконано*
13	Підготовка презентації та доповіді на передзахист.	22.12.2025 р.	*виконано*
14	Передзахист кваліфікаційної роботи.	23-24.12.2025 р.	*виконано*
15	Технічна самоекспертиза роботи на відповідність вимогам до оформлення та виправлення недоліків.	12-16.01.2026 р.	*виконано*
16	Експертиза роботи керівником та зовнішнім експертом (рецензентом).	20.01.2026 р.	*виконано*
17	Доопрацювання доповіді та презентації для захисту.	22.01.2026 р.	*виконано*
18	Захист кваліфікаційної роботи.	26-30.01.2026 р.	*виконано*

Керівник Сергій МІЧКІВСЬКИЙ
Керівник Олег МУШИНСЬКИЙ
Здобувач Андрій ТИМЧЕНКО

ЗМІСТ

АНОТАЦІЯ

Тимченко А.В. «Гнучке управління розробкою вебзастосунку для онлайн-запису до барбершопу «Barbercheck»

У кваліфікаційній роботі здійснено комплексне дослідження особливостей застосування гнучких методологій управління (Agile) у процесі розробки вебзастосунків на прикладі сервісу онлайн-запису до барбершопу «Barbercheck». Проаналізовано теоретичні засади Agile-підходів, зокрема Scrum, та визначено їх переваги порівняно з традиційними моделями управління (Waterfall) у сфері веб-розробки.

На основі дослідження предметної області ринку барбершопів визначено функціональні та нефункціональні вимоги до вебзастосунку, змодельовано архітектуру системи та розроблено основні модулі продукту. Особливу увагу приділено організації роботи Scrum-команди, плануванню спринтів, оцінці трудомісткості завдань і використанню метрик для моніторингу ефективності процесу розробки.

У роботі обґрунтовано доцільність застосування гнучкого підходу для створення вебзастосунку, запропоновано інструментарій управління процесом розробки та здійснено оцінку економічної ефективності впровадження Agile-методології.

Ключові слова: Agile, Scrum, вебзастосунок, управління проєктами, онлайн-запис, барбершоп, Product Backlog, Sprint, метрики ефективності.

ANNOTATION

Tymchenko A.V. "Agile management of web application development for online booking in a barbershop 'Barbercheck'"

The master's thesis presents a comprehensive study of the application of Agile management methodologies in the process of web application development, using the "Barbercheck" online booking service as a case study. The theoretical foundations of Agile approaches, particularly Scrum, are analyzed, and their advantages over traditional models (Waterfall) in web development are identified.

Based on the analysis of the barbershop service market, functional and non-functional requirements for the web application are defined, the system architecture is modeled, and key product modules are developed. Special attention is paid to Scrum team organization, sprint planning, task estimation, and the use of metrics for monitoring development efficiency.

The study substantiates the feasibility of applying Agile methodologies for web application development, proposes tools for managing the development process, and evaluates the economic effectiveness of Agile implementation.

Key words: Agile, Scrum, web application, project management, online booking, barbershop, Product Backlog, Sprint, performance metrics.

ВСТУП

Актуальність дослідження. Сучасна економіка значною мірою залежить від швидкості та якості впровадження інноваційних інформаційних технологій. Управління розробкою програмного забезпечення (РПЗ) є ключовою компетенцією, яка визначає конкурентоспроможність бізнесу, особливо у високодинамічних галузях, таких як сфера послуг. Ефективне управління РПЗ вимагає використання підходів, що здатні оперативно реагувати на мінливі потреби замовника, технологічні виклики та постійні ринкові зміни. У цьому контексті гнучкі методології, зокрема Scrum, набули широкого визнання завдяки їх здатності забезпечувати інкрементальне доставлення цінності та високу адаптивність процесу. Ця магістерська робота присвячена дослідженню та практичній апробації моделі гнучкого управління процесом створення вебзастосунку для автоматизації онлайн-запису до барбершопу під умовною назвою «Barbercheck».

Метою роботи є теоретичне обґрунтування управління (на основі Scrum) процесом створення вебзастосунку для онлайн-запису до барбершопу «Barbercheck», що забезпечить ефективне досягнення бізнес-цілей проєкту.

Для досягнення поставленої мети визначено такі **завдання**:

1. Проаналізувати теоретичні основи гнучких методологій управління проєктами, зокрема принципи Scrum, та визначити ключові переваги їх застосування у веб-розробці.
2. Вивчити предметну область, провести аналіз вимог до системи онлайн-запису для барбершопу та обґрунтувати архітектурні й технологічні рішення.
3. Розробити модель проєктування та реалізації функціональних модулів вебзастосунку «Barbercheck», орієнтовану на інкрементне та ітеративне доставлення цінності.

4. Визначити оптимальну організаційну структуру Scrum-команди, розробити інструментарій для моніторингу прогресу та оцінки трудомісткості робіт.

5. Здійснити економічне обґрунтування доцільності впровадження гнучкого управління та оцінити економічну ефективність розробленого вебзастосунку.

Об'єкт дослідження: гнучке управління процесом розробки вебзастосунку для онлайн-запису до барбершопу.

Предмет дослідження: Гнучке управління розробкою вебзастосунку для онлайн-запису до барбершопу «Barbercheck».

Методи дослідження: для досягнення поставленої мети використовувалися методи системного аналізу (для визначення загальної структури системи), функціонального моделювання (для деталізації вимог у форматі User Stories), економіко-математичного моделювання (для оцінки ефективності), а також методи порівняльного аналізу (для вибору методологій управління та технологічного стеку).

Практичне значення роботи полягає в тому може бути використана IT-командами для організації ефективного процесу розробки не лише для барбершопу, але й для інших B2B- та B2C-сервісів бронювання. Також, в результаті роботи створено архітектурний та функціональний прототип вебзастосунку «Barbercheck», який є готовим до комерціалізації та безпосереднього впровадження у мережі барбершопів.

Структура та обсяг роботи: магістерська робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків. Загальний обсяг роботи становить 54 сторінки, включаючи перелік використаних джерел розміщений на 5 сторінках, який включає 38 посилань.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ ГНУЧКОГО УПРАВЛІННЯ ТА ВЕБ-РОЗРОБКИ

1.1. Еволюція та принципи гнучких методологій управління проєктами (Agile).

Еволюція гнучких методологій управління проєктами пов’язана з кризою традиційних каскадних (waterfall) підходів у 1980–1990-х роках. У класичній моделі життєвого циклу ПЗ робота організовується як послідовність стадій – від збору вимог до впровадження, – причому зміни на пізніх етапах вважаються небажаними та дуже дорогими. У сфері веб-розробки, де вимоги користувачів змінюються надзвичайно швидко, такі підходи дедалі частіше призводили до затримок, перевищення бюджету й невідповідності кінцевого продукту реальним потребам бізнесу. Саме на цьому ґрунті виникають «легковагові» методи – Scrum, Extreme Programming (XP), Crystal, Feature Driven Development, – які шукають спосіб поєднати інженерну дисципліну з гнучкістю та постійним зворотним зв’язком із замовником [2; 4].

Ключовою віхою еволюції гнучкого управління став Маніфест гнучкої розробки програмного забезпечення 2001 року, підписаний сімнадцятьма практиками розробки в Сноуберді (США) . Маніфест сформулював чотири базові цінності: пріоритет людей і взаємодій над жорсткими процесами та інструментами; орієнтацію на працюючий продукт замість надмірної документації; співпрацю із замовником замість формального дотримання контракту; готовність реагувати на зміни замість сліпого слідування початковому плану [1]. Така інверсія пріоритетів означала відхід від уявлення про проєкт як «закриту» схему, спроєктовану на старті, до розуміння розробки як постійного процесу навчання команди та замовника один в одного.

Маніфест було доповнено дванадцятьма принципами, які конкретизують, як саме повинні поводитися команди, що працюють за підходом Agile. Серед них

– орієнтація на задоволення замовника через ранню та безперервну поставку цінного продукту; готовність вітати зміну вимог навіть на пізніх стадіях; часті інкрементальні релізи; щоденна тісна співпраця замовників та розробників; створення умов, у яких мотивовані фахівці можуть самостійно організовувати свою роботу; визнання того, що найефективніший спосіб комунікації всередині команди – безпосереднє спілкування; використання працюючого програмного забезпечення як основного показника прогресу; підтримка сталого темпу, який дозволяє працювати «у довгу» без вигоряння; постійна увага до технічної досконалості та хорошого дизайну; прагнення до простоти; ставка на самоорганізовані команди; регулярна рефлексія та адаптація процесу на ретроспективах [5]. Для веб-розробки, де бізнес-моделі й інтерфейси змінюються мало не щотижня, ці принципи задають логіку коротких ітерацій, швидкого фідбеку та безперервного вдосконалення.

У межах еволюції Agile важливо виділити Scrum як найбільш поширений фреймворк організації командної роботи. Кен Швабер і Джефф Сазерленд описують Scrum як легковаговий каркас для створення адаптивних рішень у складних умовах [3]. У сучасному Посібнику зі Scrum зазначено, що фреймворк ґрунтується на емпіризмі та lean-мисленні, а його опорні стовпи – прозорість, інспекція й адаптація: команда повинна робити роботу та її результат видимими, регулярно перевіряти, що відбувається, і на основі спостережень змінювати план дій [17; 33]. Для веб-проєктів це означає організацію розробки в короткі спринти (зазвичай 1–4 тижні), по завершенні яких команда презентує інкремент продукту, отримує зворотний зв’язок від стейкхолдерів і оновлює беклог. Таке циклічне планування дозволяє швидко реагувати на метрики поведінки користувачів, зміни в інтерфейсних трендах чи вимоги до безпеки.

Паралельно зі Scrum розвивається Extreme Programming (XP), який робить акцент на інженерних практиках. Кент Бек у другому виданні «Extreme Programming Explained: Embrace Change» описує XP як методологію для малих

команд, що працюють в умовах невизначених і мінливих вимог [4]. XP спирається на п’ять ключових цінностей – комунікація, зворотний зв’язок, простота, сміливість і повага [4]. Цим цінностям відповідає набір практик: парне програмування, безперервна інтеграція, тест-керована розробка, рефакторинг, колективна власність на код, часті невеликі релізи. У контексті веб-розробки, де зміни у фронтенді й бекенді відбуваються майже щодня, XP надає технічні інструменти, які дозволяють зберігати якість коду й швидко вносити зміни без лавиноподібного накопичення дефектів.

Інший важливий напрям еволюції Agile – інтеграція з lean-підходами та візуальним управлінням потоком робіт, що проявилося у Kanban та гібридних моделях. Дошки Kanban з обмеженнями незавершеної роботи (WIP-ліміти) використовуються як у Scrum-командах, так і в чистих Kanban-системах для стабілізації потоку задач і зменшення багатозадачності. Lean-мислення, яке було чітко пов’язане зі Scrum у версії посібника 2020 року, акцентує зменшення «марнотратства» – зайвих очікувань, дублювання, частих перемикань та надлишкової документації [17; 21]. Для веб-команд це означає, наприклад, відмову від створення громіздких технічних завдань заздалегідь на користь коротких user stories, які уточнюються безпосередньо перед реалізацією.

1.2. Детальний огляд методології Scrum: ролі, артефакти, події.

Scrum є найбільш поширеною реалізацією Agile-підходу й офіційно визначений у «Посібнику зі Скраму» як легковаговий фреймворк для розроблення, постачання та підтримки складних продуктів [1]. Його структура складається з трьох взаємопов’язаних блоків: ролей (accountabilities), артефактів та подій (events), які разом забезпечують прозорість, постійну інспекцію й адаптацію. Для веб-розробки це означає побудову командної роботи навколо коротких спринтів, швидкого зворотного зв’язку з користувачами й можливості змінювати продукт без втрати керованості процесом.

У версії Scrum Guide 2020 Scrum-команда описується як єдна структура без субкоманд, що включає три типи відповідальностей: Product Owner, Scrum Master та Developers [1]. Усі вони разом відповідають за створення «Done»-інкремента в кожному спринті, але кожна роль має свій фокус. Product Owner відповідає за максимізацію цінності продукту й керує Product Backlog: формулює Product Goal, збирає та уточнює вимоги, упорядковує елементи за пріоритетністю, взаємодіє зі стейкхолдерами [1; 5]. Для веб-проєктів це часто людина, яка балансує бізнес-цілі (конверсія, утримання, монетизація) з технічними обмеженнями.

Розробники (Developers) – це всі, хто безпосередньо створює інкремент продукту: програмісти, тестувальники, UX-дизайнери, аналітики тощо. Scrum Guide підкреслює, що в них немає формальних внутрішніх ролей: команда відповідає колективно за результат спринту, підтримує Sprint Backlog та щодня адаптує план для досягнення Sprint Goal [1; 3]. Такий підхід особливо важливий у веб-розробці, де зміни у фронтенді, бекенді й інфраструктурі тісно переплетені, а відокремлення «моєї» й «не моєї» роботи призводить до затримок. Scrum Master – «служливий лідер» (servant leader), який допомагає команді й організації зрозуміти Scrum, прибирає перепони, фасилітує події, навчає Product Owner’а ефективно працювати з беклогом, а менеджмент – створювати умови для самоорганізації команд [1; 5]

Артефакти Scrum – це носії інформації, які забезпечують прозорість роботи команди. Scrum визначає три базові артефакти: Product Backlog, Sprint Backlog та Increment [1; 4]. У редакції 2020 року кожен артефакт отримав «зобов’язання» (commitment), що підсилює фокус: Product Backlog пов’язаний з Product Goal, Sprint Backlog – зі Sprint Goal, а Increment – з Definition of Done [1; 6]. Це допомагає командам не лише «вести списки задач», а й постійно утримувати стратегічне бачення продукту, ціль спринту та узгоджені стандарти якості.

Product Backlog – упорядкований список того, що може знадобитися продукту; він є єдиним джерелом роботи для Scrum-команди [1]. Product Owner

постійно його уточнює, додаючи нові user stories, технічні задачі, гіпотези для експериментів, оптимізуючи пріоритети відповідно до фідбеку користувачів, бізнес-метрик чи технічних ризиків. Product Goal описує бажаний майбутній стан продукту – наприклад, «запуск адаптивної мобільної версії платформи з коефіцієнтом конверсії не нижче N%». Кожен спринт має наближати продукт до цього стану [1; 6].

Sprint Backlog – це підмножина Product Backlog, відібрана для конкретного спринту, плюс детальніший план її реалізації. Він формується на Sprint Planning і належить лише розробникам [1; 4]. Основним елементом Sprint Backlog є Sprint Goal – коротке формулювання цінності, яку команда прагне створити за спринт (наприклад, «зменшити час завантаження головної сторінки до 2 секунд»). Упродовж спринту Sprint Backlog є живим артефактом: команда може адаптувати перелік задач, не змінюючи загальну ціль, коли виникають нові знання або технічні обмеження [1].

Increment – це сума всіх завершених елементів Product Backlog, яка відповідає Definition of Done й додає нову цінність до вже існуючого продукту. Кожен спринт має створювати щонайменше один інкремент, потенційно придатний до випуску [1; 4]. Definition of Done – узгоджений у команді список критеріїв готовності: проходження автоматизованих тестів, відсутність критичних дефектів, оновлена документація, розгорнута staging-версія тощо [1; 6]. У веб-розробці чіткий DoD особливо важливий, оскільки без нього «готові» фічі часто виявляються напівзробленими – без адаптивної верстки, аналітики чи належної безпеки.

Події Scrum створюють ритм роботи та дають формальні можливості для інспекції та адаптації артефактів. Scrum-гайд визначає п’ять подій: Спринт (Sprint) як контейнер, а всередині нього – Sprint Planning, Daily Scrum, Sprint Review та Sprint Retrospective [1; 3]. Кожна з них має чітке призначення й timebox,

а відмова від будь-якої події зменшує прозорість і позбавляє команду системної можливості виправляти курс.

Спринт є серцем Scrum: це фіксований проміжок часу тривалістю до одного місяця (на практиці в веб-розробці часто 1–2 тижні), у межах якого створюється завершений інкремент [1]. Під час спринту не слід вносити зміни, які ставлять під загрозу Sprint Goal; склад Sprint Backlog може уточнюватися, але не так, щоб втратити цінність, сформульовану в цілі. Скасувати спринт може лише Product Owner, якщо Sprint Goal втратив сенс, але в реальних командах це відбувається рідко, оскільки короткі ітерації дозволяють швидко змінювати пріоритети між спринтами [2; 3].

Sprint Planning відкриває кожен спринт. У цій події беруть участь всі члени Scrum-команди; разом вони відповідають на три ключові запитання: чому спринт цінний (формування Sprint Goal), що може бути зроблено в цьому спринті (відбір елементів з Product Backlog) та як обрана робота буде виконана (побудова плану реалізації) [1]. У веб-проєктах сюди входить розбиття фіч на технічні задачі, визначення залежностей між модулем фронтенду, API-шаром і базою даних, а також попереднє оцінювання зусиль.

1.3. Порівняльний аналіз Scrum та традиційних (Waterfall) підходів в контексті розробки веб-застосунків

Waterfall-підхід історично був базовою моделлю життєвого циклу програмного забезпечення: система створюється послідовно через етапи аналізу вимог, проєктування, реалізації, тестування й впровадження, при цьому перехід на попередню фазу вважається небажаним винятком. Модель, описана В. Ройсом у 1970 р., добре працює там, де вимоги можна детально зафіксувати на початку й надалі майже не змінюватися [1; 2]. Водночас дослідження практики використання Waterfall показують, що при невизначених бізнес-потребах і високій динаміці ринку цей підхід втрачає ефективність: значні затримки,

накопичення ризиків на пізніх етапах і невідповідність готового продукту актуальним очікуванням замовників стають системною проблемою [3].

Scrum, навпаки, ґрунтується на ітеративно-інкрементальній розробці та емпіризмі: продукт створюється серією коротких спринтів, кожен із яких завершується потенційно придатним до поставки інкрементом. Scrum-команда працює в умовах постійної інспекції й адаптації – цілі спринту, зміст беклогу й навіть технічні рішення можуть переглядатися в міру отримання нових знань про користувача чи технологічні обмеження [4]. З погляду веб-розробки це принципово змінює логіку управління: замість одноразового «заморожування» вимог на старті Scrum передбачає безперервний діалог із замовником через демонстрації інкрементів, огляд результатів на Sprint Review і уточнення Product Backlog.

Ключова відмінність у контексті веб-застосунків – ставлення до змін. У Waterfall зміна вимог на пізній фазі веде до переробки проєктної документації, тестових сценаріїв і, часто, вже реалізованого коду, що суттєво збільшує витрати й строк розробки; тому зміни намагаються мінімізувати й формалізувати через процедури change request [1; 3]. Scrum, навпаки, розглядає зміну вимог як нормальний спосіб уточнення розуміння продукту: Product Owner має право переглядати пріоритети беклогу між спринтами, а команда – адаптувати план усередині спринту, не руйнуючи загальну Sprint Goal.

Висновки до розділу 1

У першому розділі було розглянуто теоретичні основи гнучкого управління проєктами, зокрема принципи Agile та особливості застосування методології Scrum у веб-розробці. Проаналізовано еволюцію підходів до управління розробкою програмного забезпечення та визначено ключові переваги гнучких методологій порівняно з традиційною каскадною моделлю (Waterfall).

Встановлено, що Agile-підхід забезпечує високу адаптивність до змін вимог, скорочення ризиків та підвищення ефективності взаємодії в команді. Особливу увагу приділено ролям, артефактам і подіям Scrum, які формують структурований, але водночас гнучкий процес розробки.

Таким чином, доведено доцільність використання гнучких методологій у сфері веб-розробки як ефективного інструменту управління проєктами в умовах динамічного середовища.

РОЗДІЛ 2
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ УПРАВЛІНСЬКИХ РІШЕНЬ

2.1. Аналіз ринку послуг барбершопів: ключові вимоги до сервісів онлайн-запису

Цифровізація сфери краси та догляду за собою зробила онлайн-запис фактично стандартом для сучасних барбершопів. Профільні огляди барбер-софтів підкреслюють, що спеціалізоване програмне забезпечення сьогодні охоплює не лише запис, а й управління клієнтською базою, розкладом майстрів, нагадуваннями, платежами й аналітикою . За даними ринку, платформи на кшталт SimplyBook.me, Booksy, StyleSeat, Fresha позиціонують онлайн-запис як інструмент збільшення завантаженості крісел, зменшення no-show та зростання повторних візитів [3; 4].

Для барбершопу ключова цінність сервісу онлайн-запису полягає у безперервній доступності та зменшенні ручної роботи адміністратора. Огляди для барбер-індустрії відзначають, що 24/7-запис через сайт, мобільний застосунок або соцмережі, автоматичні нагадування, повторні записи й просте перенесення візитів суттєво підвищують задоволеність клієнтів і стабілізують графік майстрів [1; 5–7]. У статті EasyWeek прямо наголошується, що на практиці майже кожен барбершоп виграє від впровадження онлайн-запису: зростає середній чек і лояльність гостей, а навантаження на персонал зменшується [7].

Аналіз функціональності типових барбер-платформ дозволяє виділити базові вимоги, які сьогодні розглядаються клієнтами як «must have» (табл 2.1). По-перше, це простий та швидкий сценарій запису: клієнт має за кілька кроків обрати послугу, майстра (за бажанням) і час, підтвердити запис та, за потреби, оплатити депозит чи повну суму [1; 6; 8]. Будь-які зайві кроки або неочевидна

навігація призводять до відмови від запису, тому провайдери підкреслюють важливість інтуїтивного інтерфейсу й оптимізованого «шляху користувача».

По-друге, критичною є наявність автоматичних повідомлень: SMS/e-mail-нагадування перед візитом, підтвердження запису, повідомлення про перенесення чи скасування. Практичні кейси показують, що автоматичні нагадування можуть зменшувати кількість пропущених записів на 15–20 % [6; 9; 10]. Для барбершопу це означає більш передбачувану завантаженість і менше «вікон» у розкладі, які важко заповнити в останню хвилину.

По-третє, ринок висуває вимоги до гнучкого керування ресурсами: розкладом майстрів, тривалістю й вартістю послуг, буферами між записами, політикою передоплат і штрафів. Спеціалізовані рішення дозволяють налаштовувати різні тривалості для стрижки, гоління, комплексних послуг, а також автоматично розподіляти записи між майстрами («round-robin»), щоб уникнути перевантаження окремих барберів [6; 11] Для мережевих барбершопів додається вимога роботи з кількома локаціями в одному інтерфейсі.

Четвертий блок вимог пов’язаний із клієнтським досвідом та маркетингом. Сучасні барбер-платформи пропонують персональні профілі майстрів із фото робіт, відгуками й рейтингами, програми лояльності, промокоди, нагадування про час наступної стрижки [5; 6; 9]. Часто реалізується інтеграція із соцмережами (Facebook, Instagram): клієнт може перейти до запису безпосередньо зі сторінки барбершопу. З погляду бізнесу, системи також мають базові аналітичні звіти: завантаженість по днях/майстрах, середній чек, конверсія записів, частка повторних візитів [2; 10].

Нарешті, в умовах зростання кіберзагроз важливою стає безпека й надійність. Огляди онлайн-систем бронювання відзначають, що шифрування каналу (HTTPS/TLS), захист даних клієнтів, безпечні платіжні шлюзи та резервне копіювання – це обов’язкові елементи сучасного рішення [12–14]. Для барбершопу, який часто працює з телефонними номерами, e-mail-адресами,

історією візитів і, за наявності онлайн-оплати, платіжними даними, компромісів щодо безпеки бути не може.

Таблиця 2.1 - Вимоги до сервісу онлайн-запису для проєкту «Barbercheck»

Група вимог ринку	**Інтерпретація для вебзастосунку «Barbercheck»**
24/7 онлайн-запис, мультиканальність	Особистий кабінет клієнта, віджет для сайту барбершопу, посилання з соцмереж
Простий сценарій запису	3–4 кроки: вибір послуги → майстра/будь-якого майстра → час → підтвердження/оплата
Автоматичні нагадування й повторні візити	SMS/e-mail-нагадування, «натисни, щоб перенести», пропозиція повторного запису через X тижнів
Управління ресурсами	Розклад майстрів, кастомні тривалості послуг, буфери, політика передоплат і штрафів
Профілі майстрів, фото, відгуки	Каталог майстрів із портфоліо, рейтингами та фільтрами за стилями/послугами
Аналітика та звітність	Панель адміністратора: завантаженість, no-show, повторні візити, топ-майстри/послуги
Безпека й надійність	HTTPS, шифрування паролів, інтеграція з PCI-DSS-сумісними платіжними сервісами, резервні копії

Таким чином, «Barbercheck» має не лише реалізувати базовий онлайн-запис, а й відповідати очікуванням ринку щодо комплексного управління клієнтами й ресурсами, маркетингових можливостей та безпеки. Це визначає зміст функціональних і нефункціональних вимог, деталізованих у наступних підрозділах.

2.2. Моделювання та деталізація функціональних вимог вебзастосунку «Barbercheck» (User Stories, Acceptance Criteria)

Аналіз ринку дозволяє сформувати бачення «Barbercheck» як системи, що об’єднує три основні ролі: клієнта (існуючого або нового гостя барбершопу), майстра (барбера) та адміністратора/власника салону. Відповідно до підходу Agile, функціональні вимоги до системи доцільно моделювати у вигляді користувацьких історій, де кожна історія описує потребу конкретної ролі та бізнес-цінність, яку вона приносить [15].

Для клієнта ключовими сценаріями є швидкий пошук барбершопу чи майстра, перегляд доступних слотів, запис, перенесення чи скасування візиту, а також перегляд історії відвідувань. Для майстра – перегляд власного розкладу, блокування часу, підтвердження чи коригування записів, перегляд картки клієнта. Для адміністратора – управління каталогом послуг, графіком роботи, політикою передоплат, а також доступ до аналітичних звітів.

Базова структура користувацької історії для «Barbercheck» відповідає класичному шаблону: «Як [роль], я хочу [дія], щоб [цінність]». Прийнятні критерії (Acceptance Criteria) формалізують, що саме вважати виконанням історії – з погляду UX, бізнес-логіки та граничних ситуацій (табл 2.2). Для прикладу наведемо кілька репрезентативних історій, які формують «скелет» функціональності системи.

Перша історія описує запис нового клієнта без реєстрації:

- Як гість, я хочу мати можливість записатися онлайн без створення облікового запису, щоб швидко забронювати зручний час у барбершопі.

Acceptance Criteria для цієї історії включають:

- гість може обрати барбершоп, послугу та час;
- система запитує мінімальний набір даних (ім’я, телефон, за бажанням e-mail);
- після підтвердження виводиться екран із деталями запису й надсилається SMS/e-mail-повідомлення;
- той самий слот часу стає недоступним для інших користувачів;
- у календарі відповідного майстра з’являється новий запис зі статусом «підтверджено».

Друга історія стосується управління записами зареєстрованого клієнта:

Як зареєстрований клієнт, я хочу переглядати свої майбутні та минулі записи в особистому кабінеті, щоб легко переносити візити й повторно записуватися до улюбленого майстра.

Критерії приймання:

- в особистому кабінеті доступний розділ «Мої записі» з фільтрами за статусом;
- для майбутніх візитів доступні кнопки «перенести» та «скасувати» з урахуванням політики штрафів/передоплат;– для минулих візитів доступна дія «записатися ще раз» з автоматичною підстановкою послуги й майстра;
- усі зміни записів синхронізуються з календарем майстрів і системою нагадувань.

Третя історія адресована майстру:

Як майстер, я хочу бачити свій щоденний і тижневий розклад у зручному календарі, щоб планувати свою зайнятість і вчасно готуватися до прийому клієнтів.

Acceptance Criteria:

- майстер бачить календар із відображенням усіх записів, включно з ім’ям клієнта, послугою, тривалістю;
- є можливість переключення режимів «день/тиждень»;
- майстер може блокувати особистий час (перерви, відпустка), після чого відповідні слоти стають недоступними для запису;
- зміни миттєво відображаються на стороні клієнта (у віджеті онлайн-запису).

Четверта історія стосується адміністратора салону:

Як адміністратор барбершопу, я хочу налаштовувати тривалість і вартість послуг для кожного майстра, щоб гнучко керувати завантаженістю та ціновою політикою.

Критерії приймання:

- у панелі адміністратора є довідник послуг із можливістю задавати окремі налаштування для різних майстрів;

- зміни у тривалості автоматично впливають на розрахунок доступних слотів у календарі;– зміни у вартості відображаються в інтерфейсі клієнта до моменту завершення запису;
- система зберігає історію змін для аудиту.

Таблиця 2.2 - «User Story – Acceptance Criteria (стисло) – Бізнес-цінність».

User Story (стисло)	**Acceptance Criteria (ключові)**	**Бізнес-цінність**
Онлайн-запис без реєстрації	Мінімальні поля, підтвердження, блокування слоту	Зменшення бар'єру входу для нових клієнтів
Особистий кабінет із історією візитів	Перенесення, скасування, повторний запис	Підвищення лояльності та повторних візитів
Календар майстра з блокуванням особистого часу	Режими день/тиждень, блокування, синхронізація з клієнтським інтерфейсом	Зручність планування, зниження конфліктів у графіку
Налаштування послуг та цін адміністратором	Окремі параметри для майстрів, історія змін	Гнучке управління доходом і завантаженістю

Таким чином, функціональна модель «Barbercheck» побудована навколо прозорих користувацьких історій із чіткими критеріями приймання, що спрощує планування спринтів у Scrum, постановку задач у Jira/Trello та оцінювання готовності інкрементів.

2.3. Визначення та моделювання архітектурних вимог до системи: надійність, масштабованість, безпека

Архітектурні вимоги до «Barbercheck» формуються не лише з функціональних сценаріїв, а й з очікувань ринку щодо якості онлайн-сервісів. Огляди систем онлайн-бронювання прямо наголошують на трьох ключових атрибутах: надійність (доступність і відмовостійкість), масштабованість (здатність системи витримувати ростуче навантаження) та безпека (захист персональних і платіжних даних) [12; 16–18].

Надійність для онлайн-запису означає, що система має бути доступною у години активності клієнтів (типово 9:00–22:00) з мінімальними простоями. Практичні рекомендації для сервісів бронювання вказують на цілі доступності не

нижчі за 99,5–99,9 % на місяць, а також на потребу захисту від втрати даних через резервне копіювання й кластеризацію бази даних [16; 20]. Для «Barbercheck» доцільно закласти вимогу: у разі збою окремого веб-вузла система продовжує роботу завдяки балансуванню навантаження, а база даних має як мінімум щоденний backup і механізми відновлення.

Масштабованість є критичною, якщо система орієнтується не на один барбершоп, а на багатолокаційну мережу або модель SaaS. Технічні гіди з проєктування систем бронювання (наприклад, для готелів) описують цільові показники: сотні/тисячі одночасних користувачів, низька латентність пошуку слотів (частки секунди) та горизонтальна масштабованість окремих сервісів [16]. Для «Barbercheck» це трансформується у вимоги:

– backend-сервіс має бути безстанним, щоб його можна було масштабувати горизонтально (додавати інстанси за потреби);

– операції «пошук доступних слотів», «створення запису» та «перенесення візиту» мають оброблятися в межах 300–500 мс за типової завантаженості;

– архітектура має підтримувати переведення окремих функцій (наприклад, розсилки нагадувань) у фонові черги, щоб не блокувати основні запити користувачів.

Безпека, за висновками профільних статей, включає кілька шарів: шифрування каналу передачі даних (TLS), захист персональної інформації, інтеграція з безпечними платіжними шлюзами, захист від типових веб-загроз (SQL-ін’єкції, XSS, CSRF), а також контроль доступу на рівні ролей [12–14; 18]. Для «Barbercheck» це конкретизується так:

– вся взаємодія клієнтських додатків із сервером відбувається по HTTPS;

– паролі зберігаються лише у вигляді стійких хешів із сіллю;

– рольова модель (клієнт, майстер, адміністратор, супер-адміністратор платформи) реалізується на рівні API й інтерфейсу

– дані про картки не зберігаються в системі, а оплата реалізується через інтеграцію з PCI-DSS-сумісним провайдером;

– застосовується журналювання критичних дій (створення, перенесення й скасування записів, зміни в довіднику послуг, у налаштуваннях тарифів).

З точки зору архітектурного стилю «Barbercheck» доцільно реалізувати як модульний моноліт із чітким розділенням шарів: презентаційний (web-/mobile-клієнти), API-шар, бізнес-логіка (керування записами, розкладом, тарифами), шар доступу до даних. Така n-layer-архітектура рекомендована в офіційних гідах з побудови веб-застосунків як відправна точка, яку надалі можна еволюційно розділити на мікросервіси у разі росту навантаження [19]. Це дозволяє вже на старті проєкту чітко виділити ядро доменної логіки (записи, розклади, послуги, користувачі) та приховати деталі конкретної СУБД чи фреймворку.

Таким чином, архітектурні вимоги до «Barbercheck» формуються як сукупність цільових атрибутів якості та відповідних їм технічних рішень. У системі з помірним навантаженням барбершопів надмірно складні мікросервісні схеми не є обов’язковими, однак дотримання принципів відмовостійкості, горизонтальної масштабованості й «security by design» забезпечить надійну основу для подальшого розвитку продукту.

Висновки до розділу 2

У другому розділі проведено аналіз предметної області та обґрунтовано вимоги до вебзастосунку для онлайн-запису до барбершопу «Barbercheck». Досліджено особливості ринку послуг барбершопів та визначено ключові потреби користувачів, що впливають на функціональність системи. Сформовано функціональні та нефункціональні вимоги до вебзастосунку, виконано моделювання основних бізнес-процесів та деталізацію вимог у форматі User Stories. Також обґрунтовано архітектурні рішення, що забезпечують надійність, масштабованість і безпеку системи.

У результаті дослідження створено цілісне бачення майбутнього продукту та закладено основу для його ефективної реалізації із застосуванням гнучких підходів управління.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ «BARBERCHECK»

3.1. Розробка архітектури системи: вибір технологічного стеку (Frontend/Backend) та структури бази даних

Класичні веб-стеки на кшталт LAMP (Linux, Apache, MySQL, PHP) історично домінували завдяки простоті розгортання й великій кількості готових CMS-рішень. Сьогодні вони доповнюються більш сучасними варіаціями: заміна PHP на Node.js або Python (Stack на основі Django чи Flask), використання Nginx замість Apache або перехід на контейнеризовані середовища [1].(Паралельно сформувалися Java-центровані стекі на базі Spring Boot і Jakarta EE, а також .NET-стек (ASP.NET Core, EF Core, SQL Server або PostgreSQL), що добре вписуються в хмарну інфраструктуру Azure й інші платформи [2].

Суттєву популярність отримали JavaScript-орієнтовані стеки, які дозволяють використовувати одну мову на фронтенді й бекенді. MERN-стек (MongoDB, Express, React, Node.js) дає можливість будувати повноцінні SPA й прогресивні веб-додатки, використовуючи React як бібліотеку для створення компонентного інтерфейсу, Express як легковаговий серверний фреймворк, а MongoDB як документаційну NoSQL-базу [3; 4]. Аналогічні конфігурації – MEAN (з Angular) або MEVN (з Vue) – застосовуються там, де командам зручніше працювати з іншими frontend-фреймворками. Для високонавантажених застосунків часто використовується зв’язка Node.js/TypeScript з NestJS на бекенді та React/Next.js або Angular на фронті, із застосуванням реляційних баз (PostgreSQL, MySQL) або гібридних рішень на кшталт PostgreSQL + Redis.

Окремим напрямом стали JAMstack-архітектури (JavaScript, APIs, Markup), де основний вміст попередньо збирається статичним генератором (Gatsby, Next.js у режимі SSG, Nuxt, Hugo) й доставляється через CDN, а динамічність

забезпечується запитами до API-сервісів [5; 6]. Такий підхід дає дуже високу продуктивність і безпеку, оскільки немає традиційного «живого» серверного движка, але вимагає перебудови мислення: бекенд перетворюється на набір сервісів (headless CMS, payment-API, auth-платформи), а бізнес-логіка частково переноситься на клієнт. Для контентних платформ, маркетингових сайтів, блоги та документації JAMstack часто є оптимальним вибором.

Архітектурні рішення визначають, як компоненти стеку співвідносяться між собою. Традиційною є багатошарова (n-layer) архітектура, де виділяють, як мінімум, презентаційний шар (UI), прикладну логіку (business layer) та шар доступу до даних; у .NET-керівництві з сучасних веб-застосунків така структура розглядається як відправна точка для більш просунутих підходів – «чистої» (clean) або гексагональної архітектури [2]. Чиста й гексагональна архітектури орієнтуються на доменну модель, відокремлюючи бізнес-логіку від інфраструктурних деталей (баз даних, веб-фреймворків), що дозволяє легше тестувати й змінювати технічний стек без переписування всього застосунку [7; 8].

Мікросервісна архітектура стала популярною для великих веб-платформ, де система розбивається на набір незалежно розгортуваних сервісів із власними базами даних, які взаємодіють через легковагові протоколи (HTTP/REST, gRPC, повідомлення). Огляди архітектурних патернів підкреслюють переваги мікросервісів у масштабованості й незалежності розгортання, але й застерігають від зростання операційної складності: необхідність продуманої спостережуваності, централізованого логування й управління транзакціями [7; 9; 10]. Для невеликих веб-проєктів у багатьох випадках більш доцільним виявляється «модульний моноліт» – один процес розгортання з чітко розділеними внутрішніми модулями, який за потреби можна еволюційно «розрізати» на мікросервіси [11]

Вимоги, сформовані у розділі 2, фактично задають рамки для архітектурних рішень: вебзастосунок «Barbercheck» має забезпечувати зручний інтерфейс для клієнтів і майстрів, водночас бути надійним, масштабованим і безпечним для роботи кількох барбершопів. З огляду на це доцільно обрати технологічний стек, орієнтований на сучасну фронтенд-розробку та ефективну серверну частину.

Для клієнтської частини доцільно використати React + TypeScript, що відповідає сучасним рекомендаціям офіційної документації: React забезпечує компонентний підхід до побудови інтерфейсу, а TypeScript додає статичну типізацію й підвищує надійність коду. Це дозволяє створити повторно використовувані компоненти (форма запису, календар, картки майстрів, особистий кабінет), а також прозоро описати контракти даних між фронтендом і бекендом.

Серверну частину доцільно реалізувати на Node.js з фреймворком NestJS. NestJS базується на TypeScript, використовує модульну архітектуру (modules, controllers, providers) і добре пристосований до побудови масштабованих вебзастосунків. У термінології NestJS бізнес-логіка «Barbercheck» природно розкладається на модулі: AuthModule, BookingModule, ScheduleModule, BarbersModule, AnalyticsModule, NotificationModule тощо. Комунікація з клієнтом відбувається через REST API (HTTP/JSON) та WebSocket-канал для синхронізації в реальному часі.

Як СУБД обрано PostgreSQL. Це реляційна база даних з багатим набором типів даних (integer, numeric, text, timestamp, jsonb тощо), що добре підходить для моделювання сутностей «користувач», «майстер», «барбершоп», «запис», «послуга» й дозволяє гнучко працювати з часовими інтервалами та аналітичними вибірками. Для доступу до даних може використовуватися ORM (Prisma/TypeORM), що спрощує міграції та підтримку схеми.

На рівні логічної архітектури система являє собою багатошарову вебсистему з чітким поділом на клієнтський, прикладний та даний шар (рис. 3.1).

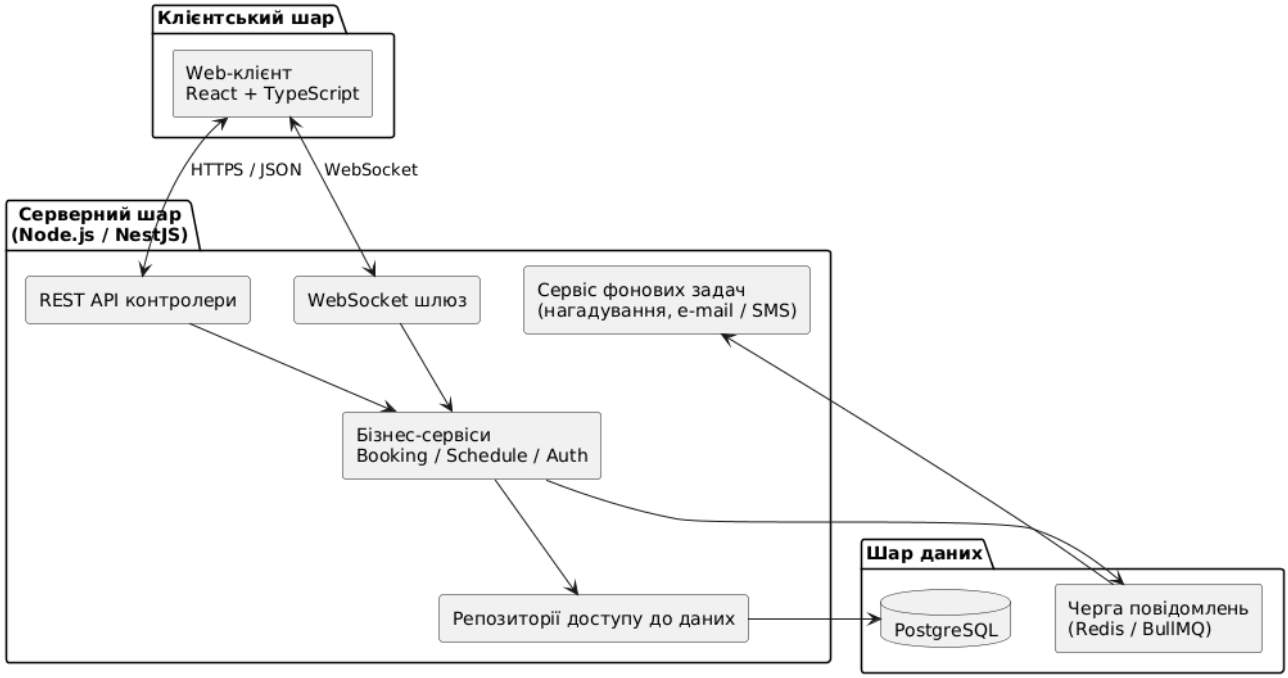

Рис. 3.1 – Узагальнена архітектура вебзастосунку «Barbercheck»

Джерело :розроблено автором

У такій схемі:

- React-клієнт відповідає за відображення інтерфейсу, маршрутизацію між сторінками (запис, календар, кабінет), взаємодію з REST API та підписку на WebSocket-події (оновлення записів у реальному часі).
- NestJS Backend реалізує REST-контролери (авторизація, список послуг, CRUD для записів, аналітика), WebSocket-шлюз (події «запис створено», «перенесено», «скасовано»), бізнес-сервіси (правила бронювання, буфери, перевірки конфліктів у графіку), репозиторії для доступу до PostgreSQL, а також модуль фонового оброблення подій (черга нагадувань, e-mail/SMS).
- PostgreSQL зберігає весь доменний стан системи; черга (напр. Redis + BullMQ) використовується для відкладеної обробки сповіщень та інших неінтерактивних задач.

Схема даних «Barbercheck» повинна підтримувати роботу кількох барбершопів, їхніх майстрів, клієнтів, послуг та календаря записів. Концептуальна модель може бути описана такими основними сутностями:

- users – облікові записи (логін, пароль, роль: client / barber / admin / owner).
- barbershops – барбершопи/філії.
- barbers – картки майстрів (прив'язка до користувачів та барбершопів).
- clients – профілі клієнтів (контакти, історія візитів).
- services – послуги (назва, тривалість, базова вартість).
- appointments – записи (дата/час початку, тривалість, статус, послуга, клієнт, майстер).
- workdays / timeslots – розклад роботи майстрів.
- notifications – журнал надісланих сповіщень (тип, канал, статус).
- audit_log – журнал змін критичних даних (за потреби).

У вигляді ER-діаграми (рис. 3.2) це можна представити так:

На фізичному рівні таблиця appointments, наприклад, може мати поля:

- id (PK, UUID),
- barbershop_id (FK),
- barber_id (FK),
- client_id (FK),
- service_id (FK),
- start_time (timestamp with time zone),
- end_time (timestamp with time zone),
- status (enum: created / confirmed / cancelled / no_show),
- created_at, updated_at.

Використання типів timestamp, integer, numeric, text та uuid у PostgreSQL відповідає офіційним рекомендаціям щодо моделювання таблиць.

Рис. 3.2 – Концептуальна ER-діаграма основних сутностей «Barbercheck»

Джерело: розроблено автором

3.2. Проєктування інтерфейсу користувача (UI/UX) для клієнтської частини

Інтерфейс «Barbercheck» має бути максимально «легким» для кінцевого клієнта: користувач очікує записатися за кілька кліків, бачити прозорий календар і мати простий особистий кабінет без перевантаження зайвими деталями. При проєктуванні UI/UX застосовується підхід mobile-first, адаптивна верстка (React + CSS/utility-фреймворк), а також послідовна дизайн-система з базовою палітрою та повторно використовуваними компонентами (кнопки, інпути, картки, модальні вікна).

Головні сценарії для клієнта:

1. Онлайн-запис – вибір барбершопу (якщо їх кілька), послуги, майстра (або «будь-який») і часу.

2. Перегляд і керування записами – список майбутніх візитів із можливістю перенесення/скасування, а також історія візитів.

3. Особистий кабінет – перегляд профілю, уподобань (улюблений майстер, стандартні послуги), налаштування сповіщень.

Навігаційна модель може бути представлена у вигляді діаграми потоків (рис. 3.3):

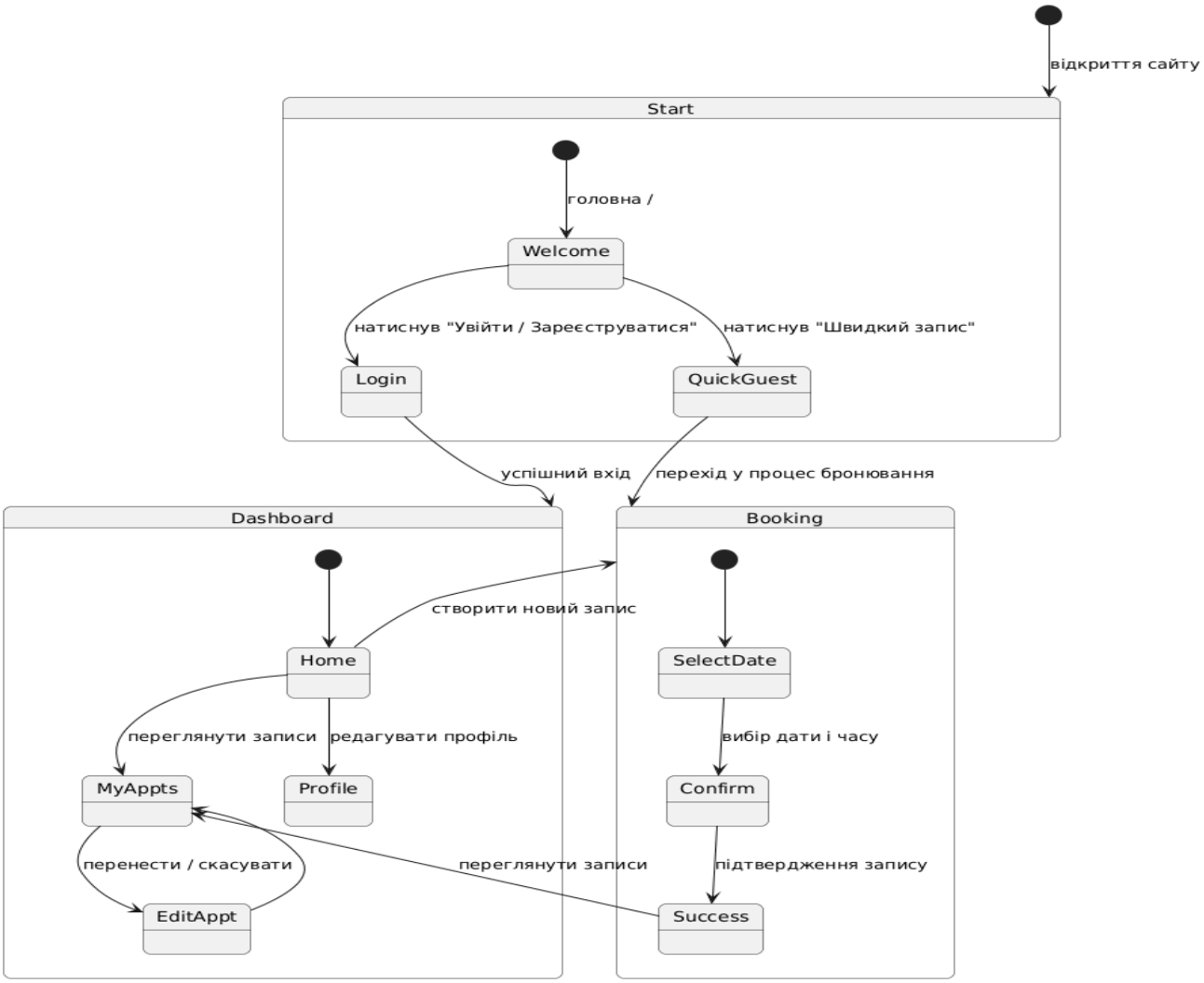

Рис. 3.3 – Узагальнений потік користувача у вебзастосунку «Barbercheck»

Джерело: розроблено автором

На основі цього потоку можна створити низку прототипів (wireframes):

Сторінка запису: зверху – вибір барбершопу і послуги, нижче – список майстрів з міні-картками (фото, ім’я, рейтинг), праворуч – календар зі слотами; кнопка «Підтвердити» стає активною лише після вибору всіх обов’язкових параметрів.

Календар клієнта: відображає майбутні й минулі записи у вигляді списку чи карток з датами; для майбутніх візитів – кнопки «Перенести»/«Скасувати», для минулих – «Записатися знову».

Особистий кабінет: вкладки «Профіль», «Мої записи», «Налаштування сповіщень».

З технічного погляду кожен екран реалізується як компонент або набір компонентів React:

- BookingPage (форма запису, календар),
- MyAppointmentsPage,
- ProfilePage,
- спільні компоненти: TimeSlotGrid, BarberCard, ServiceSelector, Toast/Alert, Modal.

Чітка компонентна структура відповідає рекомендаціям офіційної документації React щодо побудови інтерфейсів: розбивка на дрібні компоненти, що отримують дані через props і піднімають події вгору за деревом.

3.3. Реалізація функціоналу управління для адміністратора та барберів

Окрема частина «Barbercheck» – це панель управління для адміністратора та інтерфейс барберів. Вони можуть бути реалізовані як:

- окремий розділ того ж вебзастосунку (admin area),
- або як окремий фронтенд (admin SPA), що використовує ті самі API.
- Для барбера ключовими є:
- перегляд власного розкладу у вигляді календаря (день/тиждень),

- підтвердження/коментування записів,
- блокування особистого часу (перерви, відпустка),
- базовий перегляд картки клієнта (історія візитів, примітки).

Інтерфейс може бути спрощеною версією календаря адміністратора: барбер бачить лише «свої» записи та деяку агреговану статистику (кількість записів за тиждень, середній чек).

Для адміністратора барбершопу:

- управління каталогом послуг (назви, тривалість, вартість, прив'язка до барберів),
- управління майстрами (розклад, доступність, ролі),
- аналітика й звітність: завантаженість по днях, no-show, повторні візити, рейтинг майстрів,
- налаштування політики передоплат і штрафів, часових буферів.

Логіку доступів між ролями доцільно описати у вигляді простої діаграми варіантів використання (рис. 3.4).

На бекенді NestJS це природно транслюється у набір модулів і контролерів:

- AdminController – ендпоїнти для управління послугами, майстрами, барбершопами;
- BarberController – ендпоїнти для календаря барбера, блокування часу;
- AnalyticsController – ендпоїнти для дашбордів.

Модульна архітектура NestJS (modules/controllers/providers) дозволяє розділити відповідальності: кожен контролер працює через сервіси (BookingService, ScheduleService, AnalyticsService), а ті – через репозиторії, згідно з рекомендованими практиками.

Для сервісу онлайн-запису критично важливо, щоб:

1. клієнти отримували сповіщення (підтвердження, нагадування, зміни запису),

2. розклади в інтерфейсі барберів та адміністратора оновлювалися в реальному часі, без ручного перезавантаження сторінки.

Сповіщення доцільно реалізувати як окремий фоновий процес, що споживає події з черги. Загальний сценарій можна показати на діаграмі послідовностей (рис. 3.5).

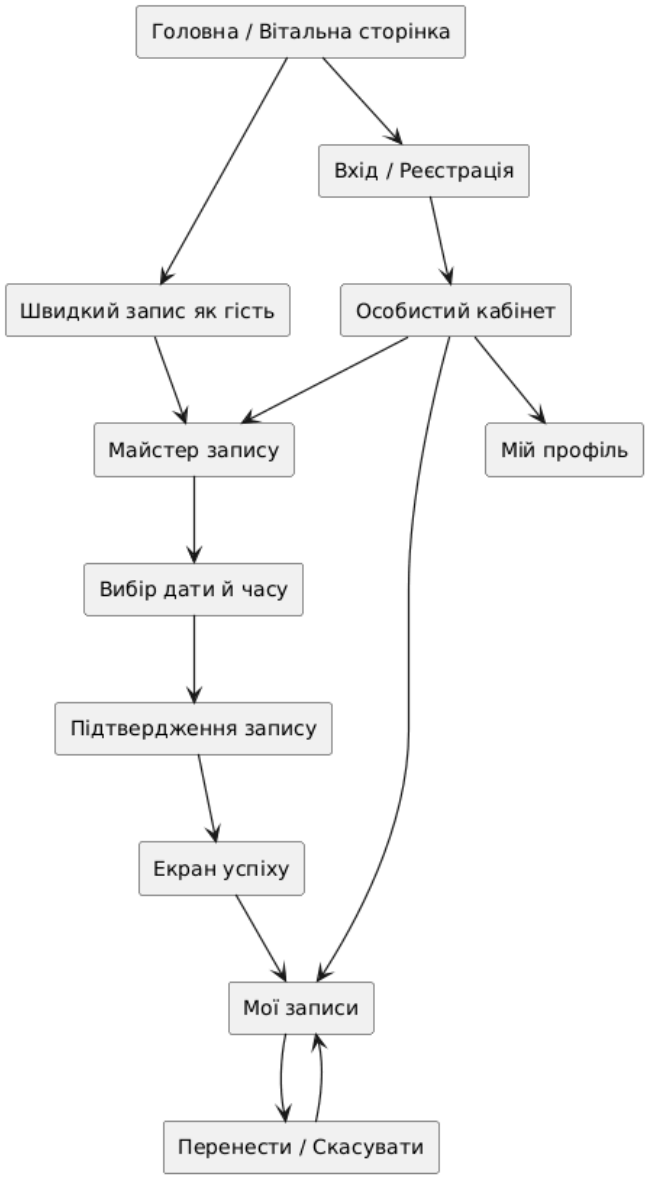

Рис. 3.4 – Узагальнені варіанти використання для ролей системи «Barbercheck»

Джерело: розроблено автором

Рис. 3.5 – Послідовність обробки події створення запису й відправки сповіщення

Джерело: розроблено автором

Аналогічно працюють події appointment.rescheduled та appointment.cancelled: бекенд публікує подію в чергу, воркер формує відповідні повідомлення (оновлені час/статус), а результат зберігає в таблиці notifications. Такий підхід знімає навантаження з основного API (користувач не чекає, поки SMS буде реально надіслано) й відповідає патернам побудови масштабованих сервісів.

Щоб барбери та адміністратор бачили оновлення календаря без перезавантаження сторінки, використовується канал WebSocket (або сумісна бібліотека, як-от Socket.IO). NestJS має вбудовану підтримку WebSocket-шлюзів (Gateways), що дозволяє організувати підписки на події на кшталт appointments:update для конкретного барбера чи барбершопу.

У результаті, як тільки новий запис створюється або змінюється, бекенд одразу транслює оновлені дані всім підписаним клієнтам (панель барбера, панель адміністратора). Це дозволяє:

- уникнути конфліктів у розкладі (два адміністратори не забронюють один і той самий слот),

- підвищити «живість» інтерфейсу (відображення заповненості дня в реальному часі),
- краще інтегрувати CRM-функції (наприклад, миттєве сповіщення про новий запис у внутрішньому чаті).

Сукупність цих механізмів (черга подій, фонові воркери, WebSocket-оновлення) робить «Barbercheck» не просто «формою для запису», а повноцінною інтерактивною платформою для управління завантаженістю барбершопу.

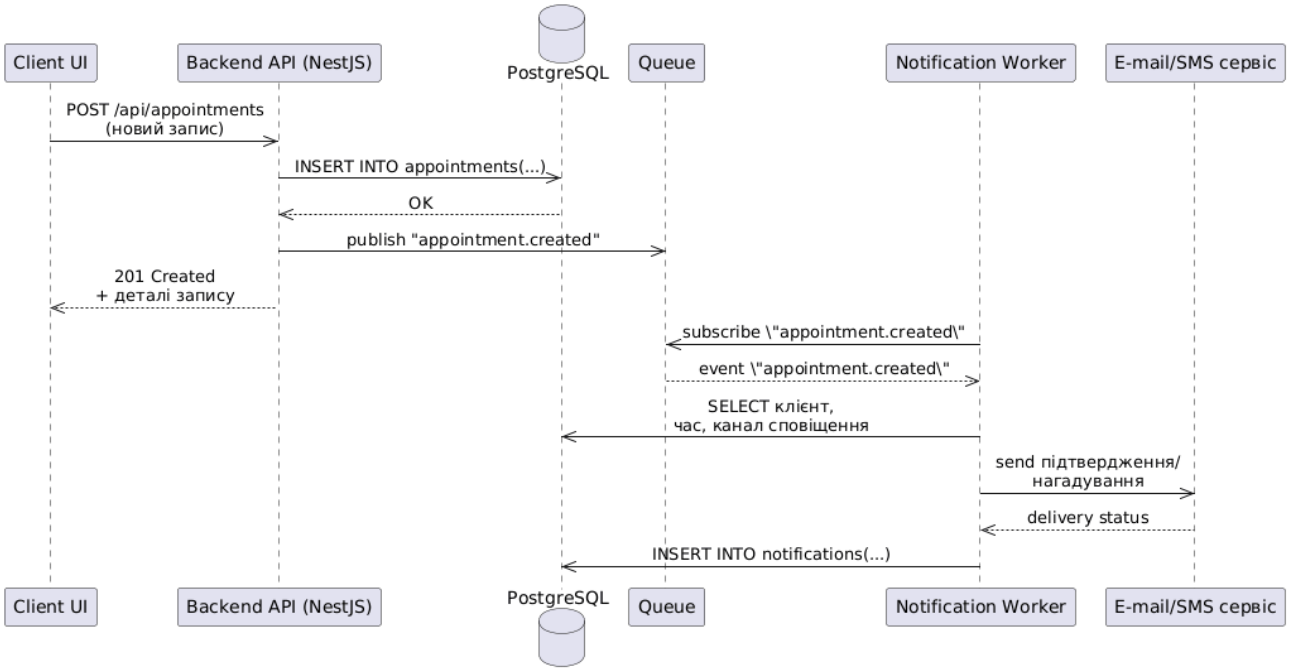

Рис. 3.6 – Синхронізація календаря барбера в реальному часі через WebSocket

Джерело: розроблено автором

Висновки до розділу 3

У третьому розділі здійснено проєктування та реалізацію вебзастосунку «Barbercheck». Розроблено архітектуру системи, визначено технологічний стек та структуру бази даних.

Спроєктовано інтерфейс користувача з урахуванням принципів зручності використання та реалізовано основний функціонал для клієнтів, адміністраторів і барберів. Забезпечено відповідність системи визначеним раніше вимогам.

Отримані результати підтверджують можливість практичної реалізації вебзастосунку із застосуванням сучасних технологій та підходів до розробки.

РОЗДІЛ 4
ОРГАНІЗАЦІЯ ГНУЧКОГО ПРОЦЕСУ РОЗРОБКИ ТА ОЦІНКА ЕФЕКТИВНОСТІ

4.1. Організація Scrum-команди: розподіл ролей, комунікаційні матриці та регламенти проведення мітингів (Planning, Daily Scrum, Review, Retrospective).

У проєкті «Barbercheck» Scrum використовується як базовий фреймворк організації командної роботи. Scrum-команда в актуальній редакції Scrum Guide складається з Product Owner'а, Scrum Master'а та розробників (Developers); усі разом вони утворюють єдину Scrum Team, відповідальну за створення цінного інкременту продукту кожен спринт [1; 2].

Для невеликого вебзастосунку на кшталт «Barbercheck» доцільний склад 5–7 осіб:

- Product Owner – відповідає за цінність продукту, формує та пріоритизує Product Backlog (у нашому випадку – історії онлайн-запису, аналітики, інтерфейсів адміністратора);
- Scrum Master – фасилітує Scrum-події, допомагає команді дотримуватися принципів Scrum, знімає перешкоди;
- Developers – кросфункціональна група, до якої входять backend- і frontend-розробники, іноді QA-інженер та UX/UI-дизайнер, які спільно відповідають за інкремент [1].

Для прозорості взаємодії доцільно зафіксувати комунікаційну матрицю участі в основних подіях Scrum. У табл. 4.1 використано позначення: В – веде/фасилітує; У – обов'язковий учасник; І – інформується за результатами.

Backlog Refinement формально не є обов'язковою подією Scrum, але широко рекомендується як практична активність [1; 3]

Канали комунікації для «Barbercheck» можуть бути такими:

- стратегічні рішення та беклог – Jira, Confluence;
- щоденна координація – Slack/Teams;
- код і рев'ю – Git-платформа (GitHub / GitLab);
- синхронні мітинги – відеоконференції (Zoom/Meet).

Це відповідає сучасним рекомендаціям щодо інструментальної підтримки Scrum-команд, де цифрові дошки (Jira, Trello) та VCS (Git) забезпечують прозорість прогресу й історії змін [4; 5]

Таблиця 4.1 – Матриця участі ролей у Scrum-подіях проєкту «Barbercheck»

Подія / Роль	**Product Owner**	**Scrum Master**	**Developers**
Sprint Planning	В/У	В/У	У
Daily Scrum	І (за потреби)	В/У	У
Sprint Review	В	В/У	У
Sprint Retrospective	У	В	У
Backlog Refinement*	В/У	У	У

Орієнтуючись на Scrum Guide та практичні гайди [1; 6; 7], для «Barbercheck» можна зафіксувати такі регламенти:

Sprint Planning (2–4 години для двотижневого спринту). Мета – визначити Sprint Goal і відібрати з Product Backlog'у User Stories у Sprint Backlog. Вхідні артефакти: пріоритизований Product Backlog, дані про Velocity, актуальний стан продукту. Вихід: Sprint Goal, список історій зі Story Points, початковий план задач.

Daily Scrum (до 15 хвилин щодня). Проводиться в один і той самий час, бажано зранку. Developers синхронізують роботу, відповідаючи неформально на запитання «що зроблено / що планується / які є перешкоди». Product Owner може бути присутнім як слухач, але не веде мітинг [1]

Sprint Review (1–2 години). Мета – продемонструвати інкремент стейкхолдерам (власники барбершопів, внутрішні користувачі) та отримати зворотний зв'язок. Для «Barbercheck» тут показуються нові можливості:

наприклад, календар барбера, екран статистики тощо. Результатом є оновлений Product Backlog і потенційні зміни дорожньої карти.

Sprint Retrospective (1–1,5 години). Команда рефлексує над процесом: що пішло добре, що заважає, які експерименти з покращення запровадити в наступному спринті. Практики рекомендують виходити щонайменше з 1–2 конкретними діями на наступний спринт [7]

Оскільки «Barbercheck» розробляється за гнучкими підходами (Scrum), вибір інструментів управління проєктом має безпосередній вплив на прозорість беклогу, планування спринтів і координацію роботи команди. Сучасні огляди Agile-інструментів відзначають, що Jira стала де-факто стандартом для технічних команд з комплексними беклогами, тоді як Trello залишається популярним легковаговим засобом візуалізації задач у форматі канбан-дошок [21–24].

Jira пропонує гнучкі Scrum- і Kanban-дошки, ієрархію задач (епіки – історії – підзадачі), звіти щодо спринтів і швидкості команди, а також глибоку інтеграцію з Git-репозиторіями та CI/CD-сервісами [21; 25]. Це робить її зручним інструментом для технічної частини проєкту «Barbercheck»: тут може вестися повний реєстр User Stories із підрозділу 2.2, баг-репорти, технічні задачі (рефакторинг, налаштування інфраструктури, автоматичні тести). Для кожної історії задаються Story Points, прив’язка до спринту та Definition of Done, а коментарі й вкладення дозволяють зберігати контекст рішень.

Trello, за даними оглядів, є простішим, але дуже наочним інструментом, орієнтованим на невеликі команди або некодерські активності – дизайн, маркетинг, комунікації [22; 24]. У контексті «Barbercheck» Trello доцільно використати як «зовнішню» дошку для стейкхолдерів (власники барбершопів, маркетологи): тут можуть відображатися високорівневі епіки («реліз MVP», «запуск мобільної версії», «інтеграція з платіжною системою»), запити від пілотних клієнтів, ідеї майбутніх фіч. Такий поділ дозволяє технічній команді

зберігати деталізацію в Jira, а бізнес-команді – мати простий і візуальний огляд статусу проєкту.

Git виступає базовим інструментом керування версіями коду. Згідно з офіційною документацією GitLab і GitHub, Git – це розподілена система керування версіями, яка дозволяє кожному розробнику мати повну локальну копію репозиторію з усією історією та працювати незалежно, використовуючи гілки (branches) для експериментів і подальшого злиття в основну гілку [26–28]. Огляди переваг Git підкреслюють його швидкодію, гнучкість гілкування та надійність (наявність повної історії в кожного учасника), що особливо важливо для команд, які працюють у швидкому ітеративному режимі [29; 30].

Коміти з Git інтегруються з задачами Jira: у повідомленні до коміту вказується ключ задачі (наприклад, BARB-23), що дає змогу автоматично пов’язувати зміни в коді з конкретними історіями чи дефектами. Така інтеграція рекомендована як краща практика для Agile-команд, оскільки спрощує трасування змін і аудит [21; 25].

Таблиця 4.2 - Загальна логіка використання інструментів у проєкті «Barbercheck»

Інструмент	**Основна роль у проєкті «Barbercheck»**	**Адаптація під гнучке управління**
Jira	Ведення беклогу, планування спринтів, облік задач і багів	Scrum-дошка з епіками 1.x/2.x, Story Points, релізні версії
Trello	Візуальний роадмап і комунікація зі стейкхолдерами, задачі дизайну/маркетингу	Канбан-дошка «Ідеї → У роботі → На релізі → Готово»
Git	Керування версіями коду, гілкування, злиття, підтримка CI/CD	Git-workflow з feature/*-гілками, pull-request’ами, тегами релізів

У результаті поєднання Jira/Trello та Git створює цілісну екосистему гнучкого управління для «Barbercheck»: вимоги відображаються у вигляді користувацьких історій, розбиваються на реалізовані інкременти в спринтах, а кожна зміна в коді має чіткий зв’язок із бізнес-цінністю. Це не лише підвищує прозорість та передбачуваність розробки, а й спрощує подальшу підтримку й розвиток системи. Таким чином, організація Scrum-команди для «Barbercheck» забезпечує чіткий розподіл відповідальності, прозору комунікацію та регулярні цикли зворотного зв’язку, що важливо для швидкозмінних вимог до вебзастосунку.

4.2. Планування та оцінка трудомісткості (Story Points, Velocity) на основі гнучкого підходу.

У гнучких методологіях оцінка трудомісткості виконується не у годинах, а в Story Points – відносних одиницях вимірювання зусилля, що враховують складність, обсяг роботи, ризики й невизначеність [8; 9]. Такі оцінки дозволяють команді сфокусуватися на порівняльній відносності завдань («ця історія вдвічі складніша за базову») замість неточних спроб передбачити точний час.

Для «Barbercheck» базовою може бути історія «як клієнт, я хочу записатися онлайн як гість» – вона отримує, наприклад, 3 SP. Тоді складніша історія «як адміністратор, я хочу бачити дашборд завантаженості по майстрах» може оцінюватися в 8 SP, якщо команда розуміє, що це потребує суттєво більше роботи (складний SQL, візуалізація, фільтри). Для шкали зазвичай використовують модифікований ряд Фібоначчі (1, 2, 3, 5, 8, 13, …), який добре відображає нелінійне зростання складності [8; 10].

Оцінка проводиться спільно (Planning Poker), коли всі розробники одночасно показують значення, а розбіжності обговорюються. Atlassian і Scrum Alliance підкреслюють, що спільна оцінка стимулює обговорення ризиків і прихованих допущень [11].

Velocity – це середня кількість Story Points, яку команда фактично завершує за спринт. Velocity обчислюється постфактум: після спринту сумуються тільки ті історії, які повністю відповідають Definition of Done [11; 12].

Для Barbercheck можна змоделювати:

- Спринт 1: завершено 18 SP,
- Спринт 2: 22 SP,
- Спринт 3: 20 SP.

Середня Velocity = (18 + 22 + 20) / 3 = 60 / 3 = 20 SP/спринт.

Якщо в Product Backlog'у залишилося 80 SP до MVP, то можна приблизно очікувати: 80 SP / 20 SP/спринт = 4 спринти до готовності MVP (за стабільної Velocity). Це не жорсткий дедлайн, а робочий прогноз для команди й стейкхолдерів [9; 14].

Сучасна література застерігає від використання Velocity як інструменту тиску на команду або порівняння різних команд; воно має бути внутрішньою метрикою для планування й самооцінки [15; 16]. Для «Barbercheck» Velocity застосовується для:

- планування обсягу робіт на спринт;
- перевірки реалістичності дорожньої карти;
- оцінки впливу змін у команді (нові люди, відпустки) на пропускну здатність.

Для оцінювання трудомісткості завдань і планування спринтів використовується продуктова дошка в Jira, де кожен елемент беклогу описаний у форматі User Story, має пріоритет, оцінку в сторі-поінтах та поточний статус. На рисунку 4.1 наведено фрагмент продуктового беклогу проєкту «Barbercheck» з основними історіями для реалізації функціоналу онлайн-запису та панелі адміністратора.

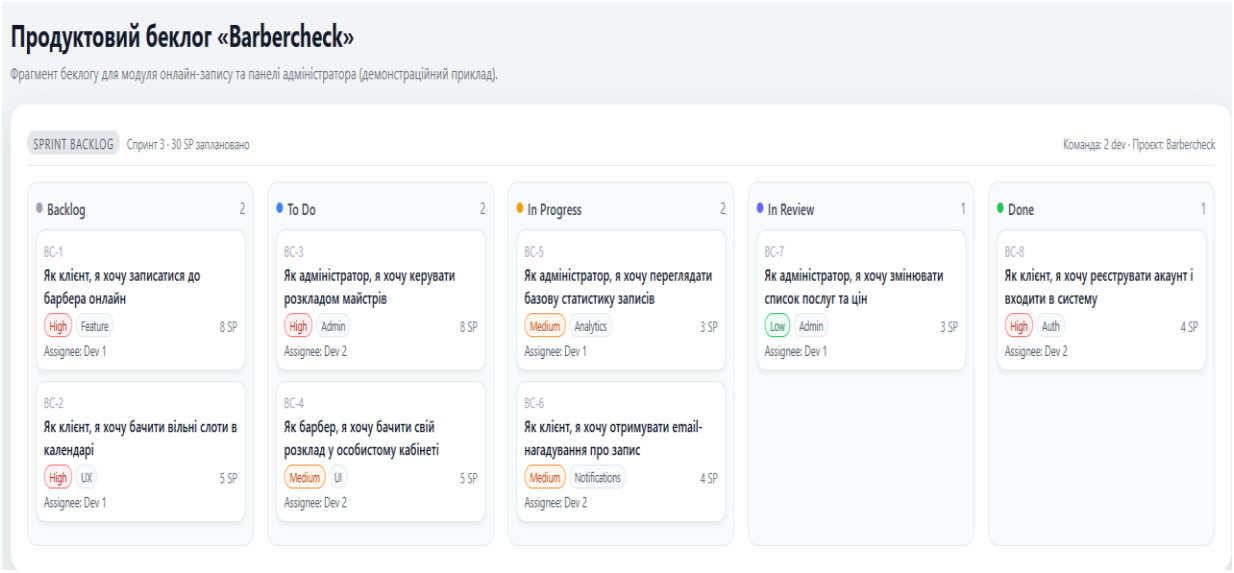

Рис 4.1 – Фрагмент продуктового беклогу проєкту «Barbercheck»

Джерело: розроблено автором

Як видно з рис. 4.2, кожна історія має свій ключ, текст формулювання, пріоритет, оцінку в сторі-поінтах і статус. Саме з цього беклогу під час Sprint Planning формується Sprint Backlog із відібраних для спринту історій.

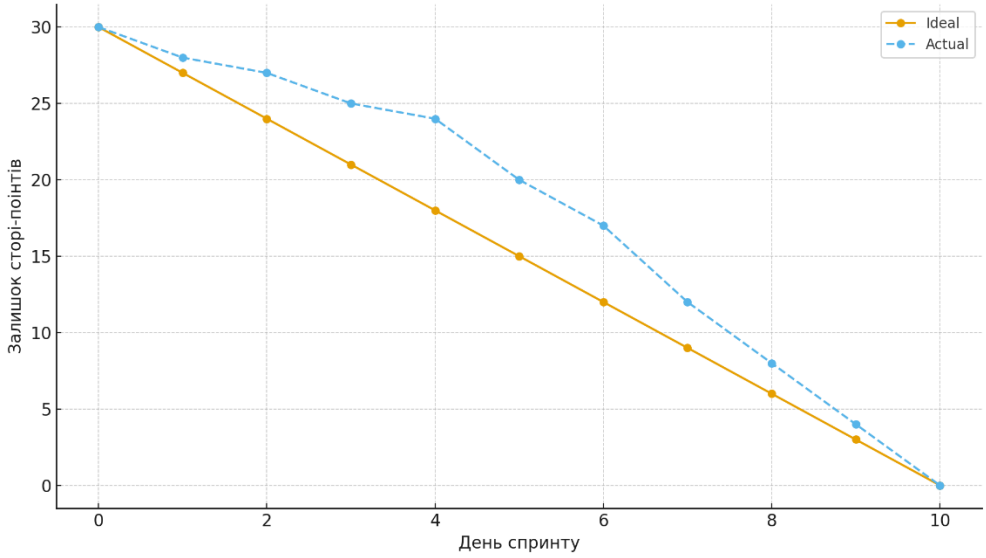

Рис. 4.2 – Burndown chart спринту проєкту «Barbercheck»

Джерело: розроблено автором

Для візуального контролю темпу виконання завдань у спринті використовується діаграма згоряння задач (burndown chart). На рис. 4.3 показано приклад burndown chart для третього спринту, де суцільна лінія відображає ідеальну траєкторію зменшення обсягу робіт, а пунктирна – фактичне виконання сторі-поінтів командою протягом 10 днів спринту. Видно, що в середині спринту відставання від ідеальної лінії було незначним, а наприкінці команда повністю «спалила» запланований обсяг робіт.

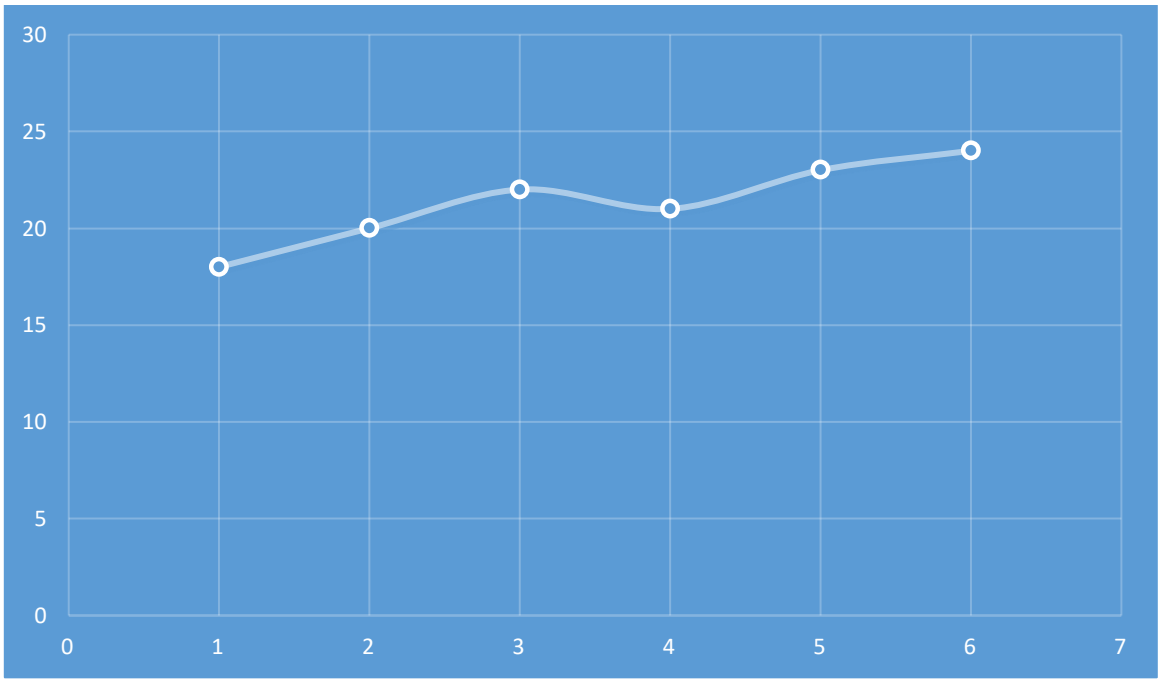

Рис 4.3 - Кількість завершених сторі-поінтів

Джерело: розроблено автором

Стабільність та прогнозованість роботи команди відображає velocity-діаграма, на якій по осі абсцис відкладаються спринти, а по осі ординат – кількість завершених сторі-поінтів. На рис. 4.3 наведено умовний приклад velocity-діаграми для шести спринтів проєкту «Barbercheck». З рисунка видно, що після перших двох спринтів команда вийшла на відносно стабільну пропускну

здатність у межах 20–24 сторі-поінтів за ітерацію, що дозволяє використовувати середню velocity для прогнозування термінів реалізації подальших релізів.

Таким чином, Story Points і Velocity створюють гнучкий інструмент управління трудомісткістю, що краще відповідає природі дослідницько-розробницької роботи, ніж традиційні часові оцінки.

4.3. Розробка системи показників (метрик) для моніторингу прогресу та якості розробки

Щоб гнучкий процес розробки не перетворився на «хаос під гаслом свободи», Scrum-команди використовують систему аналітичних метрик, які дозволяють відстежувати як швидкість, так і якість розробки. Огляди Agile-метрик рекомендують поєднувати метрики потоку (lead time, cycle time), продуктивності (velocity, частота релізів) та якості (defect density, відсоток автоматизованих тестів) [17–19].

Для проєкту «Barbercheck» доцільно виділити кілька груп метрик.

1. Метрики прогресу та потоку роботи

- Velocity – кількість SP, завершених за спринт (див. підрозд. 4.2).
- Lead time – час від появи задачі в беклозі до її завершення;
- Cycle time – час від початку активної роботи (статус «In Progress») до завершення [17; 20].

Lead time показує, скільки в середньому чекає стейкхолдер від ідеї до готової фічі; cycle time – ефективність внутрішнього процесу розробки. Скорочення цих показників за умови стабільної якості є ознакою зрілості процесу.

- Defect density – кількість дефектів на одиницю розміру (наприклад, на 1000 рядків коду або на 10 SP);
- Defect escape rate – частка дефектів, виявлених після релізу (у продуктивному середовищі);

- Code coverage – відсоток коду, покритий автоматизованими тестами.

Дослідження вказують, що дефектна щільність у поєднанні з lead/cycle time дозволяє збалансувати продуктивність та якість: надто швидке зменшення cycle time без контролю дефектів призводить до «швидко, але неякісно» [19; 23].

Для вебзастосунку «Barbercheck» важливі:

- частота релізів (deploy frequency);
- Mean Time To Resolution (MTTR) – середній час від виявлення критичної помилки до її виправлення;
- availability – відсоток часу, коли система доступна користувачам [17; 23].

Таблиця 4.3 – Ключові метрики моніторингу процесу розробки «Barbercheck»

Метрика	Визначення	Джерело даних	Інтерпретація для Barbercheck
Velocity	SP, завершені за спринт	Jira	Планування обсягу спринтів, прогноз релізів
Lead time	Від створення історії до її завершення	Jira (стани задач)	Швидкість реалізації ідей
Cycle time	Від «In Progress» до «Done»	Jira/борда	Ефективність розробки
Defect density	Дефекти / 10 SP або / 1 тис. рядків коду	Issue tracker + Git	Контроль якості коду
Defect escape rate	% дефектів, виявлених у проді	Prod-лог + issue tracker	Якість тестування до релізу
Deployment freqency	Кількість релізів на місяць	CI/CD-лог	Гнучкість у виведенні оновлень
MTTR	Середній час закриття критичних багів	Issue tracker	Оперативність реагування

Важливий принцип – не перетворювати метрики на інструмент покарання. У сучасних гайдах наголошується: метрики мають служити для безперервного покращення процесу, а не для індивідуального «рейтингу» розробників [19; 23].

4.4. Економічне обґрунтування та оцінка ефективності впровадження гнучкого управління на проєкті «Barbercheck».

Для невеликого продукту «Barbercheck» економічний ефект упровадження гнучких підходів доцільно показати на максимально реалістичних для українського ринку припущеннях. Ідеться не про велику аутсорсингову компанію, а радше про невеликий стартап або студентську команду з двох розробників, де вартість їхньої праці рахується як умовна «собівартість» участі в проєкті. Нехай сумарна вартість роботи двох фахівців (наприклад, одного більш досвідченого розробника і одного джуніора) становить орієнтовно 40 000 грн на місяць разом. Один двотижневий спринт за тривалістю дорівнює половині місяця, отже вартість одного спринту для такої команди можна оцінити приблизно у 20 000 грн (40 000 × 0,5). Якщо припустити, що в гнучкій моделі MVP «Barbercheck» вдається підготувати за шість спринтів, тобто приблизно за три місяці активної роботи, загальна вартість створення мінімально життєздатного продукту становитиме близько 120 000 грн (6 × 20 000). У традиційній каскадній (Waterfall) схемі, коли значна частина вимог деталізується наприкінці і виникає багато переробок, той самий обсяг робіт легко розтягується до еквівалента дев’яти спринтів (чотири з половиною місяці), що дає орієнтовну загальну вартість близько 180 000 грн (9 × 20 000). Таким чином, різниця між підходами становить приблизно 60 000 грн, тобто близько третини загального бюджету Waterfall-варіанту. Ця різниця добре вкладається в емпіричні оцінки дослідників ROI Agile, які фіксують помітне зниження вартості й скорочення термінів постачання в гнучких проєктах порівняно з традиційними [24; 27]. Для наочності ці розрахунки можна подати у вигляді порівняльної таблиці.

З таблиці видно, що навіть за доволі скромної для українського контексту оцінки вартості праці двох розробників гнучкий підхід дозволяє зекономити близько 60 000 грн ще на етапі створення першої версії продукту. Водночас більш

ранній вихід MVP на ринок дає змогу швидше почати тестування бізнес-моделі «Barbercheck», запропонувати сервіс реальним барбершопам, протестувати структуру тарифів і сценарії використання. У літературі саме прискорення time-to-market і рання можливість монетизації розглядаються як ключові фінансові чинники, які виправдовують перехід до гнучких методологій [24; 27].

Таблиця 4.4 – Порівняння витрат на створення MVP «Barbercheck» за Agile та Waterfall

Підхід	**Кількість спринтів**	**Вартість одного спринту, грн**	**Загальна вартість MVP, грн**
Agile (Scrum)	6	20 000	120 000
Waterfall	9	20 000	180 000

Другий важливий аспект економічного ефекту Agile пов’язаний зі вартістю змін. Для системи на кшталт «Barbercheck» типовими прикладами таких змін є додавання підтримки мережі барбершопів (кілька локацій в одному акаунті), розширення аналітики чи інтеграція з іншими CRM. У Waterfall-процесі, коли основна частина архітектури, моделі даних і користувацьких інтерфейсів уже зафіксована наприкінці, подібні вимоги можуть вимагати фактично окремого «міні-проєкту» на рівні ще одного повноцінного спринту тієї ж команди. За наведених вище припущень це означає додаткові 20 000 грн витрат лише на те, щоб адаптувати систему до потреб, які не були своєчасно враховані. У гнучкому процесі з регулярними Sprint Review замовник і стейкхолдери бачать робочий прототип уже після кількох перших спринтів і можуть одразу сформулювати побажання до сценарію запису, вигляду календаря, структури звітів. Завдяки цьому вимоги, які в каскадній моделі перетворилися б на дорогі пізні change requests, у Scrum оформлюються як звичайні User Stories і реалізуються інкрементально в наступних спринтах, коли система ще є достатньо «пластичною» [2; 28]. У підсумку сумарна вартість змін розподіляється

рівномірніше в часі й зменшується, а ризик великих непередбачуваних витрат наприкінці життєвого циклу проєкту істотно падає.

Третя площина економічного ефекту стосується якості та витрат на підтримку. Дослідження показують, що команди, які послідовно застосовують гнучкі практики – короткі ітерації, безперервну інтеграцію, автоматизоване тестування, поступове погашення технічного боргу, – досягають помітного зниження дефектності продукту й пов'язаних із нею витрат на супровід [17; 19; 21]. Для невеликого вебзастосунку на кшталт «Barbercheck» це означає меншу кількість критичних збоїв у пікові години роботи барбершопів, менше аварійних виправлень у неробочий час і більш прогнозований час відновлення сервісу завдяки налагодженому CI/CD та моніторингу. Якщо для такої ж команди з двох розробників орієнтовні середньомісячні витрати на підтримку і «гасіння пожеж» без системного застосування Agile-підходів оцінити хоча б у 10 000 грн, то зниження дефектності та скорочення часу на виправлення на 30 % дає щомісячну економію близько 3 000 грн. За рік це становитиме приблизно 36 000 грн, які не потрібно витрачати на постійні латання помилок і непередбачувані понаднормові.

Сумарний економічний ефект для «Barbercheck» у такому наближенні складається з двох основних компонентів. По-перше, йдеться про близько 60 000 грн економії на етапі створення MVP завдяки скороченню тривалості розробки та уникненню масових переробок наприкінці. По-друге, це орієнтовно 36 000 грн щорічної економії на підтримці за рахунок зменшення дефектності та упорядкування процесу релізів. Разом це дає близько 96 000 грн потенційної економії за перший рік життєвого циклу продукту за умов дуже стриманих, «приземлених» для українського ринку припущень. Таким чином, Agile для «Barbercheck» виглядає виправданим не лише з точки зору теорії управління проєктами та організації командної роботи, а й з погляду прямої та непрямої економічної доцільності навіть для невеликої локальної команди

Висновки до розділу 4

У четвертому розділі розглянуто організацію гнучкого процесу розробки вебзастосунку із застосуванням методології Scrum. Визначено ролі в команді, описано процеси планування, проведення мітингів та взаємодії учасників проєкту.

Здійснено оцінку трудомісткості завдань, використано метрики (зокрема Story Points та Velocity) для моніторингу ефективності розробки. Також проведено економічне обґрунтування доцільності впровадження гнучкого підходу.

У результаті доведено, що застосування Agile-методологій підвищує ефективність управління розробкою, покращує якість продукту та сприяє досягненню бізнес-цілей проєкту.

ВИСНОВКИ

Узагальнюючи результати виконаної кваліфікаційної роботи, можна стверджувати, що поставлена мета – теоретичне обґрунтування управління (на основі Scrum) процесом створення вебзастосунку для онлайн-запису до барбершопу «Barbercheck», що забезпечить ефективне досягнення бізнес-цілей проєкту–досягнута повною мірою. Усі основні завдання, визначені у вступі, логічно реалізовані в чотирьох розділах роботи, які взаємно доповнюють одне одного та утворюють цілісну концепцію продукту й процесу його розробки.

У першому розділі було систематизовано теоретичні засади гнучких методологій управління проєктами та сучасної веб-розробки. На основі аналізу наукових джерел і практичних гайдів розкрито еволюцію Agile-підходів, ключові принципи гнучких методологій і детально розглянуто методологію Scrum як домінуючу практику в індустрії розробки вебзастосунків. Окрему увагу приділено порівнянню Scrum і класичної каскадної моделі (Waterfall) у контексті створення веб-продуктів: показано, що в умовах динамічних вимог, потреби швидких релізів і постійного зворотного зв’язку з користувачами Scrum забезпечує вищу гнучкість, менші ризики невідповідності очікуванням замовника та кращі передумови для еволюції продукту. Також було проаналізовано сучасні технологічні стеки й архітектурні підходи (JavaScript-орієнтовані стекі, n-tier, модульний моноліт, мікросервіси, REST/GraphQL), що створило методологічний фундамент для подальшого проєктування системи «Barbercheck».

Другий розділ був присвячений аналізу предметної області й формалізації вимог до майбутньої системи. На основі огляду ринку барбершопів і спеціалізованих сервісів онлайн-запису виокремлено ключові очікування бізнесу та клієнтів: доступність сервісу 24/7, простий сценарій запису, автоматичні нагадування, гнучке управління розкладом майстрів та послугами, інтеграція з

маркетинговими інструментами й базова аналітика. Ці висновки були трансформовані у систему функціональних вимог, змодельованих у форматі User Stories з чіткими Acceptance Criteria для основних ролей – клієнта, барбера та адміністратора. Паралельно було окреслено нефункціональні вимоги до архітектури системи: вимоги до надійності, масштабованості, безпеки та інтеграційних можливостей. Окремо обґрунтовано вибір інструментів підтримки гнучкого управління (Jira/Trello, Git) для організації процесу розробки й координації роботи команди.

У третьому розділі реалізовано власне проєктну частину, у якій теоретичні й аналітичні напрацювання були трансформовані в архітектурні рішення та konkretні технічні підходи. Було обґрунтовано вибір технологічного стеку для «Barbercheck»: React + TypeScript для клієнтської частини, NestJS (Node.js) для серверної логіки та PostgreSQL як основної СУБД. Запропоновано логічну архітектуру застосунку у вигляді багатошарової системи з чітким розділенням презентаційного шару, шару бізнес-логіки, API-шару та шару доступу до даних, а також побудовано концептуальну ER-модель основних сутностей (користувачі, барбершопи, майстри, послуги, записи, розклад, сповіщення). Розроблено логіку інтерфейсу користувача для клієнтів (сценарії запису, перегляд календаря, особистий кабінет) і панелі управління для барберів та адміністратора (календар майстра, довідник послуг, базова статистика). Окремим результатом стало проєктування механізмів обробки сповіщень та синхронізації даних у реальному часі на основі черг подій та WebSocket-каналу, що відповідає вимогам сучасних інтерактивних веб-платформ.

Четвертий розділ зосереджено на організації гнучкого процесу розробки та економічній оцінці ефективності обраного підходу. Було змодельовано структуру Scrum-команди для проєкту «Barbercheck», визначено ролі Product Owner’a, Scrum Master’a та кросфункціональної групи розробників, а також сформовано комунікаційну матрицю та регламенти проведення ключових Scrum-подій (Sprint

Planning, Daily Scrum, Review, Retrospective). Розглянуто механізми планування й оцінки трудомісткості через Story Points і Velocity, показано, як на основі емпіричної Velocity можна прогнозувати строки виведення MVP на ринок і керувати очікуваннями стейкхолдерів. Запропоновано систему метрик для моніторингу прогресу й якості розробки (Velocity, lead/cycle time, дефектна щільність, частота релізів, MTTR), орієнтовану на безперервне покращення процесу, а не на формальне «вимірювання» людей. Економічне обґрунтування продемонструвало, що впровадження гнучкого управління дозволяє скоротити часові й фінансові витрати на створення MVP, зменшити вартість пізніх змін і знизити витрати на підтримку за рахунок вищої якості й передбачуваності процесів.

Практична цінність роботи полягає в тому, що на прикладі конкретного домену – ринку послуг барбершопів – показано цілісну методику переходу від аналізу ринку та користувацьких потреб до архітектури, інтерфейсу й організації гнучкого процесу розробки вебзастосунку. Запропонована модель «Barbercheck» може бути використана як основа для створення реальної SaaS-платформи для салонів краси й барбершопів, а також адаптована для інших сфер, де ключовими є онлайн-запис, розклад ресурсів і комунікація з клієнтами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Швабер, К. Посібник зі Скраму™. Повний навчальний посібник зі Скраму: правила гри [Електронний ресурс] / К. Швабер, Д. Сазерленд. – Листопад 2020. – Режим доступу: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Ukrainian.pdf (дата звернення: 03.12.2025).
2. Сазерленд, Д. Scrum. Навчись робити вдвічі більше за менший час / Д. Сазерленд ; пер. з англ. Я. Лебеденко. – Харків : Клуб сімейного дозвілля, 2022. – 280 с.
3. Оккерман, С. Опанування професійного Scrum. Командна робота та ефективні Scrum-команди / С. Оккерман, Е. Рейндел. – Харків : Фабула, 2023. – 224 с.
4. Cohn, M. Agile Estimating and Planning / M. Cohn. – Upper Saddle River (NJ) : Addison-Wesley, 2005. – 368 p.
5. Pressman, R. S. Software Engineering: A Practitioner’s Approach / R. S. Pressman, B. R. Maxim. – 8th ed. – New York : McGraw-Hill Education, 2014. – 976 p.
6. Sommerville, I. Software Engineering / I. Sommerville. – 10th ed. – Boston : Pearson, 2016. – 816 p.
7. Forsgren, N. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations / N. Forsgren, J. Humble, G. Kim. – Portland : IT Revolution Press, 2018. – 288 p.
8. Humble, J. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation / J. Humble, D. Farley. – Boston : Addison-Wesley, 2011. – 512 p.
9. Rico, D. F. What is the ROI of agile vs. traditional methods? An analysis of extreme programming, test-driven development, pair programming, and scrum

(using real options) / D. F. Rico // TickIT International. – 2008. – Vol. 10, № 4. – P. 9–18.

10. Адаменко, В. О. Веб-сервіс для організації та управління освітнім процесом університету / В. О. Адаменко // Інформаційні технології – 2025 : матеріали наук.-практ. конф. (Київ, 15 травня 2025 р.). – Київ : Київ. ун-т ім. Б. Грінченка, 2025. – С. 2–3.

11. Приймак, В. Гнучкі моделі управління командною роботою в умовах цифрової трансформації / В. Приймак // Вісник Київського нац. ун-ту ім. Т. Шевченка. Серія «Економіка». – 2019. – № 207. – С. 21–27.

12. Pavliuk, T. Agile project management models in the context of digital economy / T. Pavliuk // Menedzhment. – 2024. – № 1(29). – P. 15–24.

13. Scrum Guides. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game [Електронний ресурс] / Scrum Guides. – Scrum.org, 2020. – Режим доступу: https://www.scrumguides.org/scrum-guide.html (дата звернення: 03.12.2025).

14. Scrum Guides. Scrum Guide Revisions & 2020 Updates [Електронний ресурс] / Scrum Guides. – Scrum.org, 2020. – Режим доступу: https://www.scrum.org/resources/blog/scrum-guide-2020-updates (дата звернення: 03.12.2025).

15. Jocham, R. Scrum Guide Expansion Pack [Електронний ресурс] / R. Jocham. – Scrum Expansion, 2025. – Режим доступу: https://scrumexpansion.org/scrum-guide-expansion-pack/ (дата звернення: 03.12.2025).

16. Project Manager Template. Scrum Cheat Sheet: A Quick-Start Guide to Agile [Електронний ресурс]. – 2025. – Режим доступу: https://projectmanagertemplate.com/scrum-cheat-sheet-a-quick-start-guide-to-agile/ (дата звернення: 03.12.2025).

17. Atlassian. Agile project management & collaboration tools [Електронний ресурс]. – Atlassian, 2025. – Режим доступу: https://www.atlassian.com/agile/project-management (дата звернення: 03.12.2025).

18. Wunder. Scrum events and artifacts [Електронний ресурс]. – Wunder.io, 2025. – Режим доступу: https://wunder.io/blog/scrum-events-and-artifacts (дата звернення: 03.12.2025).

19. Kitemetric. Mastering the Scrum Framework: Roles, Events, and Artifacts [Електронний ресурс]. – Kitemetric, 2025. – Режим доступу: https://kitemetric.com/blog/mastering-the-scrum-framework-roles-events-artifacts (дата звернення: 03.12.2025).

20. Atlassian. What are story points in Agile and how do you estimate them? [Електронний ресурс]. – Atlassian, 2025. – Режим доступу: https://www.atlassian.com/agile/project-management/estimation (дата звернення: 03.12.2025).

21. GeeksforGeeks. Story Points and Velocity in Scrum [Електронний ресурс]. – GeeksforGeeks, 2025. – Режим доступу: https://www.geeksforgeeks.org/story-points-and-velocity-in-scrum/ (дата звернення: 03.12.2025).

22. Mountain Goat Software. Estimating with Story Points [Електронний ресурс]. – Mountain Goat Software, 2025. – Режим доступу: https://www.mountaingoatsoftware.com/blog/estimating-with-story-points (дата звернення: 03.12.2025).

23. Scrum Alliance. Story point estimation on an agile team [Електронний ресурс]. – Scrum Alliance, 2025. – Режим доступу: https://resources.scrumalliance.org/Article/story-point-estimation-agile-team (дата звернення: 03.12.2025).

24. Scrum.org. Story points are not the problem, velocity is [Електронний ресурс]. – Scrum.org, 2023. – Режим доступу:

https://www.scrum.org/resources/blog/story-points-are-not-problem-velocity (дата звернення: 03.12.2025).

25. Koster, C. Agility Without Story Points [Електронний ресурс] / C. Koster. – Medium, 2023. – Режим доступу: https://medium.com/ (конкретну URL-адресу статті вкажи за власним посиланням) (дата звернення: 03.12.2025).

26. Talent500. Agile Story Points Estimation Guide [Електронний ресурс]. – Talent500, 2025. – Режим доступу: https://talent500.co/blog/agile-story-points-estimation-guide/ (дата звернення: 03.12.2025).

27. Edvantis. Essential IT Metrics To Track For Project Success [Електронний ресурс]. – Edvantis, 2020. – Режим доступу: https://www.edvantis.com/blog/essential-it-metrics-to-track-for-project-success/ (дата звернення: 03.12.2025).

28. DECODE. 10 essential metrics and KPIs to track in nearshore software development [Електронний ресурс]. – Decode.agency, 2025. – Режим доступу: https://decode.agency/article/nearshore-software-development-metrics/ (дата звернення: 03.12.2025).

29. Axify. 17 Agile Metrics to Track Dev Teams' Performance [Електронний ресурс]. – Axify, 2025. – Режим доступу: https://www.axify.io/en/blog/engineering/17-agile-metrics-to-track-dev-teams-performance (дата звернення: 03.12.2025).

30. Clifford, D. Agile Metrics: Measuring Progress and Success in Agile Projects [Електронний ресурс] / D. Clifford. – Medium, 2023. – Режим доступу: https://medium.com/ (конкретну URL-адресу статті вкажи за власним посиланням) (дата звернення: 03.12.2025).

31. DevCrew I/O. Software Development Metrics: Complete Guide [Електронний ресурс]. – DevCrew I/O, 2025. – Режим доступу: https://devcrew.io/blog/software-development-metrics-complete-guide/ (дата звернення: 03.12.2025).

32. MagicPod. Understanding Key Metrics for Agile Development [Електронний ресурс]. – MagicPod, 2024. – Режим доступу: https://blog.magicpod.com/ (конкретну URL-адресу статті вкажи за власним посиланням) (дата звернення: 03.12.2025).

33. Aha! Labs. Agile Metrics: A Complete Guide for PMs and Engineers [Електронний ресурс]. – Aha!, 2024. – Режим доступу: https://www.aha.io/roadmapping/guide/agile/agile-metrics (дата звернення: 03.12.2025).

34. Startearly. Developer Productivity Metrics You Should Be Measuring [Електронний ресурс]. – Startearly.ai, 2025. – Режим доступу: https://startearly.ai/blog/developer-productivity-metrics-you-should-be-measuring (дата звернення: 03.12.2025).

35. Simform. State of Agile Adoption [Електронний ресурс]. – Simform, 2022. – Режим доступу: https://www.simform.com/blog/state-of-agile-adoption/ (дата звернення: 03.12.2025).

36. Rico, D. F. What is the ROI of Agile vs. Traditional Methods? [Електронний ресурс] / D. F. Rico. – 2008. – Режим доступу: https://www.wdavidfrico.com/rico08b.pdf (дата звернення: 03.12.2025).

37. Abella, D. What is the Return on Investment (ROI) of Agile Methods? [Електронний ресурс] / D. Abella. – Daniel Abella, 2010. – Режим доступу: https://daniel-abella.com/what-is-the-return-on-investment-roi-of-agile-methods/ (дата звернення: 03.12.2025).

38. McKinsey & Company. An agile approach to funding enterprise outcomes [Електронний ресурс]. – McKinsey & Company, 2022. – Режим доступу: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/an-agile-approach-to-funding-enterprise-outcomes (дата звернення: 03.12.2025).