Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ МОБІЛЬНОГО IOS ТА ANDROID ЗАСТОСУНКУ ДЛЯ ШВИДКІСНОГО ЧИТАННЯ НА ОСНОВІ МЕТОДУ RAPID SERIAL VISUAL PRESENTATION

DEVELOPMENT AND RESEARCH OF A MOBILE IOS AND ANDROID APPLICATION FOR SPEED READING BASED ON THE RAPID SERIAL VISUAL PRESENTATION METHOD

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-11
Шидловський Б. В.

Керівник:
Бойко Л. С.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Шидловському Богдану Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *«Розробка та дослідження мобільного iOS та Android застосунку для швидкісного читання на основі методу Rapid Serial Visual Presentation».*
Керівник роботи: *к. т. н., доцент Бойко Л. С.*
затверджені наказом закладу вищої освіти від *«29» березня 2025 р.*
№ *190/01-02.*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи
«4» грудня 2025 р.
3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *огляд теоретичних основ читання, когнітивних обмежень і методу RSVP; аналіз підходів до мобільної розробки та обґрунтування вибору Flutter/Dart; формулювання вимог до мобільного застосунку Rapid Read; розробка UML-моделей системи; проєктування архітектури й основних модулів; реалізація застосунку Rapid Read засобами Flutter/Dart; тестування та пілотне експериментальне дослідження з аналізом результатів.*
5. Перелік графічного матеріалу: *16 рисунків, 25 лістингів*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Бойко Л. С.*		
Теоретичне дослідження та практична реалізація	*Бойко Л. С.*		
Експериментальне дослідження системи	*Бойко Л. С.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	***%***		
Академічна доброчесність	*Бойко Л. С.*		

7. Дата видачі завдання *«02» квітня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну схему роботи програмного продукту	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Богдан ШИДЛОВСЬКИЙ

Керівник кваліфікаційної роботи _____ Лев БОЙКО

АНОТАЦІЯ

Шидловський Б. В. Розробка та дослідження мобільного iOS та Android застосунку для швидкісного читання на основі методу Rapid Serial Visual Presentation. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається з вступу, 3 розділів, висновків, списку використаних джерел та додатків.

У даній роботі досліджено предметну область швидкісного читання та когнітивні особливості сприйняття тексту; проаналізовано метод Rapid Serial Visual Presentation (RSVP) та сучасні підходи до розробки мобільних застосунків; розглянуто методи та засоби реалізації кросплатформного мобільного застосунку швидкісного читання на базі Flutter/Dart; сформульовано завдання на кваліфікаційну роботу магістра; обґрунтовано вибір архітектурних рішень, алгоритмів і технологій для реалізації режиму RSVP; здійснено практичну розробку мобільного застосунку Rapid Read для платформ Android та iOS; описано методику експериментального дослідження роботи застосунку; проведено обробку та аналіз отриманих результатів щодо швидкості читання, зручності інтерфейсу та стабільності роботи системи.

Ключові слова: швидкісне читання, RSVP, мобільний застосунок, Flutter, Dart, кросплатформна розробка, UX, продуктивність інтерфейсу.

ABSTRACT

Shydlovskyi B. Development and Research of a Mobile iOS and Android Application for Speed Reading Based on the Rapid Serial Visual Presentation Method. Manuscript.

Master's qualification thesis in the Educational Program "Software Engineering" of the specialty 121 "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The master's qualification thesis consists of an introduction, 3 chapters, conclusions, a list of references, and appendices.

This work examines the subject area of speed reading and the cognitive aspects of text perception; analyses the Rapid Serial Visual Presentation (RSVP) method and modern approaches to mobile application development; considers methods and tools for implementing a cross-platform speed reading mobile application based on Flutter/Dart; formulates the tasks of the master's qualification thesis; substantiates the choice of architectural solutions, algorithms and technologies for implementing the RSVP mode; presents the practical development of the Rapid Read mobile application for Android and iOS platforms; describes the methodology of experimental research of the application; performs processing and analysis of the obtained results regarding reading speed, interface usability and system stability.

Keywords: speed reading, RSVP, mobile application, Flutter, Dart, cross-platform development, UX, interface performance.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасна людина щодня працює з великими обсягами текстової інформації – від новин і навчальних матеріалів до робочих документів і повідомлень. Значна частина цього контенту читається зі смартфона, що зручно, але має обмеження: невеликий екран, підвищене навантаження на зір, часті відволікання та брак часу для уважного читання. Тому зростає інтерес до технологій, які дозволяють читати швидше без критичної втрати розуміння.

Однією з таких технологій є Rapid Serial Visual Presentation (RSVP) – спосіб подачі тексту, за якого слова послідовно з'являються в одній фіксованій точці екрана. Це зменшує кількість саккадичних рухів очей, робить темп читання стабільнішим і частково компенсує обмеження малого екрана. Однак ефективність RSVP залежить від правильного проєктування інтерфейсу та алгоритмів: надто високий темп, слабкий контроль або складність повернення до попереднього фрагмента можуть знижувати якість сприйняття тексту.

Актуальність даної роботи зумовлена поєднанням двох тенденцій: зростання ролі мобільного читання та потреби у зручних інструментах для тренування швидкісного читання. На ринку є окремі рішення, але більшість із них або не адаптовані під україномовний контент, або обмежені за функціональністю, або не враховують достатньою мірою когнітивні особливості сприйняття тексту. Це створює підстави для розробки власного мобільного застосунку, орієнтованого на режим RSVP, з урахуванням реальних сценаріїв використання та можливістю подальшого дослідження його ефективності.

Метою кваліфікаційної роботи є розробка та експериментальна перевірка мобільного застосунку Rapid Read для швидкісного читання текстів на основі методу Rapid Serial Visual Presentation (RSVP) на платформах Android та iOS.

Для досягнення мети роботи необхідно розв'язати такі завдання:

– виконати огляд теоретичних основ процесу читання, когнітивних обмежень та методу RSVP;

– проаналізувати підходи до розробки мобільних застосунків і обґрунтувати вибір стеку Flutter/Dart;

– сформулювати функціональні та нефункціональні вимоги до мобільного застосунку Rapid Read;

– розробити функціональну та інформаційну моделі системи із використанням UML-діаграм;

– спроєктувати архітектуру застосунку та визначити основні програмні модулі;

– реалізувати мобільний застосунок Rapid Read засобами Flutter/Dart;

– провести тестування та пілотне експериментальне дослідження роботи застосунку і проаналізувати отримані результати.

Об'єктом дослідження є процес швидкісного читання текстів на мобільних пристроях.

Предметом дослідження є мобільний застосунок Rapid Read, що реалізує метод Rapid Serial Visual Presentation для подання тексту та забезпечує керування параметрами читання й аналіз базових показників роботи користувача з текстом.

Практична цінність роботи полягає в створенні робочого прототипу мобільного застосунку Rapid Read, який може використовуватися як інструмент для тренування навичок швидкісного читання та подальших досліджень у цій сфері. Розроблений застосунок може бути розширений за рахунок нових режимів роботи з текстом, інтеграції з іншими джерелами контенту або використаний як основа для освітніх і мовних застосунків, де важливі як швидкість, так і розуміння прочитаного.

Апробація роботи. Результати кваліфікаційної роботи заслуховувались на засіданні кафедри інженерії програмного забезпечення та відображені у публікації в науковому збірнику «Студентський науковий вісник» (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Читання – це не просто розпізнавання букв чи слів, а складний когнітивний процес, у якому одночасно працюють зір, пам'ять, увага та мовні механізми. Людина сприймає текст очима, мозок обробляє зоровий сигнал і намагається зрозуміти зміст, часто за долі секунди. Тому читання вважають однією з найскладніших навичок, яку людина поступово опановує впродовж навчання.

На базовому рівні читання починається із зорового сприймання символів. Очі сканують рядок, пересуваючись короткими стрибками – саккадами – і роблячи короткі зупинки (фіксації). Саме під час фіксацій мозок отримує інформацію про побачені літери, залучає попередній досвід, знання мови й контекст, а далі формує зміст речення. Тобто читання відбувається не як безперервний потік, а як серія коротких зупинок, у межах яких виконується основна робота з обробки інформації [1].

Після зорової обробки інформація надходить до мовних центрів мозку, де літери й слова отримують значення. Людина співвідносить побачене зі словниковим запасом, граматичними правилами та вже наявними знаннями – це називають лінгвістичною обробкою. Саме вона дозволяє не просто «декодувати» символи, а розуміти текст. Водночас читання є інтерактивним процесом: мозок постійно порівнює нові слова з уже прочитаними, будує логічні зв'язки, прогнозує продовження та коригує очікування. Розуміння змісту формується поступово, у міру надходження нової інформації.

Психологічні дослідження показують, що читання має кілька взаємопов'язаних рівнів – від розпізнавання літер до побудови цілісного смислу тексту. На ефективність впливають досвід читача, рівень уваги, швидкість

розпізнавання слів, мотивація. Тому різні люди читають з різною швидкістю, навіть працюючи з однаковим текстом. У контексті швидкісного читання (зокрема RSVP) це важливо: технології намагаються оптимізувати окремі ланки процесу – зменшити кількість рухів очей, пришвидшити подачу слів – але не можуть обійти базові фізіологічні обмеження [1]. Щоб розуміти межі таких покращень, потрібно мати уявлення про те, як влаштований сам механізм читання.

Зорове сприйняття є його основою. Людина не бачить весь рядок однаково чітко: максимальна деталізація забезпечується в центральній частині поля зору (фовеа), тоді як периферійні зони дають лише загальне уявлення про форму та розташування слів. Фактично читач «бачить» повністю лише невелику ділянку тексту, і саме вона визначає реальний темп та комфорт читання. Цю ділянку називають перцептивним вікном (рис. 1.1).

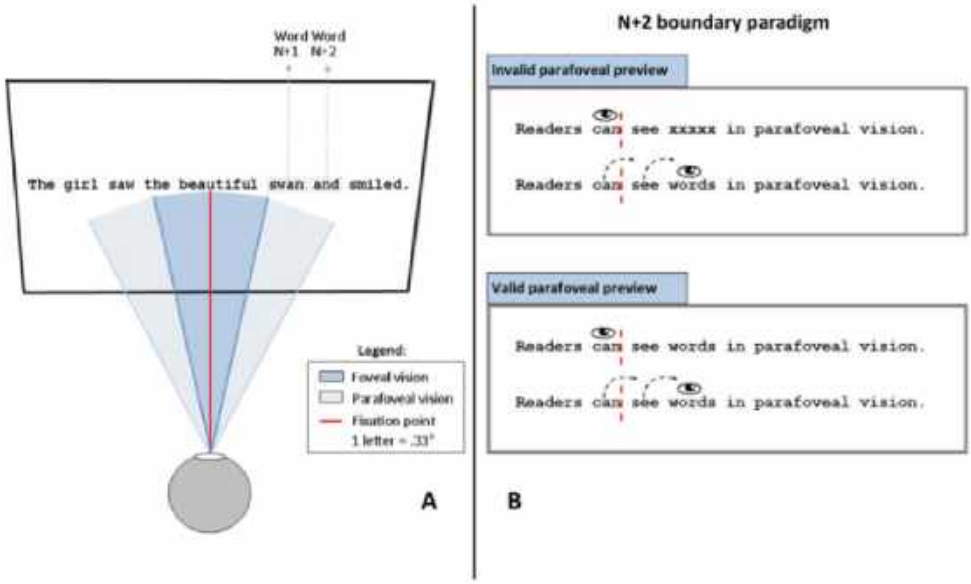

Рисунок 1.1 – Перцептивне вікно під час читання: фовеальна, парафовеальна та периферійна області [2]

У межах однієї фіксації читач чітко сприймає лише кілька символів біля точки погляду, тоді як решта рядка залишається на рівні загальних контурів. Ці

дані мозок використовує, щоб планувати наступні фіксації, обирати напрямок руху очей і приблизно прогнозувати наступні слова [3].

На швидкість і комфорт читання помітно впливають умови візуального сприйняття: розмір шрифту, контраст, інтервал між рядками, тип пристрою. Дрібний шрифт або низький контраст звужують ефективну область зору, змушують робити більше фіксацій і в підсумку сповільнюють читання. На мобільних пристроях це особливо відчутно: невеликий екран зменшує доступний фрагмент тексту, тому зростає кількість мікрорухів очей і загальне зорове навантаження.

Увага є одним із ключових механізмів ефективного читання. Людина не просто сканує текст – вона постійно вирішує, на які слова зосередитися, де зробити фіксацію і яку інформацію утримати в пам'яті. Під час читання увага тісно пов'язана з рухами очей: погляд рухається не плавно, а короткими стрибками (саккадами) і зупиняється у точках фіксації, де й відбувається обробка зорової інформації. Чим складніше слово або чим нижча концентрація, тим довшою стає фіксація [4].

Увага також впливає на вибір наступної точки погляду: за очікуваної складності чи важливості фрагмента очі рухаються повільніше, а при простому тексті – швидше перескакують між фіксаціями. Якщо увага розподілена неефективно (через втому, відволікання або надто високий темп), зростає кількість повернень до вже прочитаного – регресій, що суттєво уповільнює процес. Через це при складних текстах швидкість читання може помітно змінюватися навіть у досвідчених читачів. Типову схему рухів очей із чергуванням фіксацій і саккад наведено на рисунку 1.2.

Рисунок 1.2 – Траєкторія руху очей під час читання: фіксації та саккади [5]

Увага виконує і внутрішню когнітивну функцію: допомагає утримувати зв'язки між реченнями та відсіювати зайві думки й зовнішні подразники. Коли рівень уваги падає, текст втрачає цілісність, і читач змушений повертатися до попередніх фрагментів, щоб відновити контекст.

Отже, ефективність уваги безпосередньо впливає на якість читання: від концентрації залежать і швидкість, і глибина розуміння. Тому будь-які технології, що змінюють спосіб подачі тексту, мають враховувати, як саме увага розподіляється під час обробки інформації.

Робоча пам'ять відіграє важливу роль у читанні, оскільки відповідає за тимчасове збереження й обробку інформації під час руху очей по тексту. На відміну від довготривалої, вона працює «тут і зараз»: утримує окремі слова, структуру речення, логічні зв'язки й допомагає сформувати цілісний зміст.

Однією з найвідоміших моделей робочої пам'яті є модель А. Бадделі [6], яка виділяє три основні компоненти:

Ця модель виділяє три основні компоненти:

– фонологічну петлю, що утримує звукову інформацію й дозволяє запам'ятовувати щойно прочитані слова та проговорювати їх подумки;

– візуально-просторовий нарис, відповідальний за візуальні образи та просторову організацію тексту;

– центрального виконавця, який контролює розподіл уваги, перемикання між завданнями та об'єднання інформації в цілісне розуміння.

У процесі читання всі три компоненти робочої пам'яті працюють разом: фонологічна петля утримує окремі слова, візуально-просторовий нарис допомагає орієнтуватися в структурі тексту, а центральний виконавець стежить за логікою й переходами між думками. Коли робоча пам'ять перевантажена (складний текст, нова тема), стає важче утримувати зв'язки між реченнями, зростає ризик поверхневого читання, втрати сенсу та частих повернень до попередніх фрагментів.

При прискоренні читання (зокрема в технологіях швидкісного подання, як-от RSVP) навантаження на робочу пам'ять зростає: очі рухаються швидше

або текст подається автоматично, і в читача менше часу, щоб об'єднати інформацію в цілісний зміст. Тому саме ємність і ефективність робочої пам'яті обмежують максимальну результативну швидкість читання [6].

Розуміння тексту – це не просто прочитування слів, а здатність поєднувати їх у більші смислові структури: речення, абзаци, загальну ідею. Цей процес називається семантичною інтеграцією: мозок має не лише «бачити» слова, а й вибудовувати між ними логічні зв'язки [7]. Під час читання одночасно обробляються значення слів, їхня граматична роль і зв'язки між частинами речення, а на наступному рівні зміст окремих речень об'єднується в цілісну думку. Це потребує участі пам'яті, уваги та попередніх знань, тому різні читачі можуть по-різному розуміти той самий текст навіть за однакової швидкості.

Контекст відіграє вирішальну роль: мозок використовує попередню інформацію, щоб передбачати наступні слова, перевіряти їх відповідність очікуванням і підтримувати логічну лінію. Якщо тема знайома, інтеграція відбувається швидше; якщо нова чи складна – уповільнюється, і читач витрачає більше зусиль, щоб зібрати цілісну картину.

Семантична інтеграція залежить від кількох чинників: досвіду читача в темі, словникового запасу, складності синтаксису, рівня концентрації та обсягу інформації, яку потрібно утримати в пам'яті. Якщо хоча б один із цих елементів «просідає», текст сприймається як важкий, фрагментарний або суперечливий. Аналогічно, надто високий темп читання може призвести до того, що слова розпізнаються, але зв'язки між ними не встигають сформуватися.

Семантична інтеграція працює в двох напрямках:

– знизу догори – від окремих слів до загального змісту;

– згори донизу – коли очікування, досвід і вже сформований контекст впливають на тлумачення нової інформації.

Завдяки цьому читання є активним процесом: мозок постійно уточнює й переформатовує смисл. Саме це пояснює ситуації, коли ми «пробігаємо» очима текст і майже нічого не розуміємо: слова були прочитані, але не інтегровані в

значення, тож доводиться повертатися назад і повторно вибудовувати смислову структуру.

Отже, розуміння тексту – результат взаємодії сприйняття, пам'яті, уваги та попередніх знань. Це потрібно враховувати при оцінці методів швидкісного читання: саме семантична інтеграція найчастіше страждає, коли темп перевищує індивідуальні когнітивні можливості читача.

Традиційне читання має природні фізіологічні та когнітивні обмеження, які визначають максимально можливу швидкість і якість засвоєння тексту. Досвід і тренування можуть покращити навички, але не змінюють базові межі, пов'язані з роботою зорової системи та мозку.

Одним із головних обмежень є саккади – швидкі стрибкоподібні рухи очей. Погляд постійно переміщується по рядку, кожен рух і наступна фіксація займають час. Чим складніший текст, тим довшими стають фіксації, тож середня швидкість читання завжди обмежена фізичними властивостями зору.

До цього додається обмеження перцептивного вікна: людина чітко бачить лише невелику частину тексту, тому мозок змушений працювати з інформацією фрагментами. Неможливо охопити поглядом цілий абзац, тож читання відбувається покроково. У середньому за одну фіксацію ефективно сприймається лише 1-2 слова, що також стримує швидкість.

Суттєвими є й когнітивні обмеження. Робоча пам'ять має обмежену ємність: якщо текст перенасичений складними конструкціями чи незнайомими термінами, зростає навантаження, і темп читання падає. Увага теж не може залишатися стабільною нескінченно довго: втома, зовнішні подразники й власні думки уповільнюють читання, змушують втрачати контекст і повертатися до вже прочитаних фрагментів.

Свою роль відіграють і технічні чинники: формат тексту, розмір шрифту, якість екрана, освітлення. На смартфонах через невелику площу екрана читач зазвичай робить більше фіксацій, що також знижує швидкість [8].

Отже, традиційне читання має чіткі фізіологічні та когнітивні межі. Саме вони стимулювали появу альтернативних способів подачі тексту, зокрема

методів швидкісного читання та RSVP, які намагаються зменшити кількість саккад і раціональніше розподілити навантаження на увагу [2, 4].

Швидкість і розуміння – два основні показники ефективності читання, між якими існує природний компроміс: зі збільшенням темпу якість розуміння зазвичай знижується. Це стосується і традиційного читання, і технологій швидкісного подання тексту.

За високих швидкостей у мозку менше часу на обробку кожного слова, побудову зв'язків між реченнями та формування цілісної смислової картини. На невеликому темпі читач ще може робити паузи, перевіряти себе, повертатися до складних фрагментів; на високому – ці механізми майже зникають. Дослідження показують, що навіть досвідчені читачі мають межу: після певного порогу збільшення швидкості рівень розуміння падає, оскільки обсяг інформації зростає, а можливості робочої пам'яті та уваги залишаються сталими.

Типова крива «speed-comprehension» виглядає так: до певного моменту зростання швидкості майже не впливає на якість розуміння, але після порогу (орієнтовно 300-400 слів на хвилину, залежно від читача) розуміння різко погіршується. Саме тому фахівці з читання наголошують: надмірне підвищення темпу часто призводить до поверхневого опрацювання тексту.

Ця залежність характерна і для звичайного читання, і для методів швидкісного подання тексту. Навіть якщо технологія зменшує кількість рухів очей або подає слова в зручному форматі, когнітивні обмеження не зникають: робоча пам'ять не може обробляти необмежений потік інформації, а увага має природні межі концентрації.

Тому спроби підвищити швидкість читання без втрати якості мають враховувати цей компроміс. Його частково можна пом'якшити практикою, зміною формату тексту, введенням пауз чи тренуванням уваги, але повністю усунути конфлікт між швидкістю та розумінням неможливо [3, 7].

Метод Rapid Serial Visual Presentation (RSVP) – один із найпоширеніших підходів до швидкісного подання тексту, створений для зменшення

навантаження на зорову систему. На відміну від традиційного читання, де очі постійно рухаються рядком, у RSVP слова з'являються по одному в центрі екрана, що практично усуває потребу в саккадах, які займають значну частину часу.

Класично RSVP працює так:

– слово з'являється у фіксованій точці;

– протягом короткого проміжку часу читач його сприймає;

– слово зникає, а на тому самому місці з'являється наступне;

– очі залишаються нерухомими, а мозок зосереджується лише на змісті.

У звичайному читанні рухи очей можуть займати до 30-40 % часу, витраченого на опрацювання тексту, тоді як RSVP усуває цю складову й дозволяє досягати значно вищих швидкостей – до 500-700 слів на хвилину і більше для підготовлених користувачів. Візуальну схему подачі тексту методом RSVP наведено на рисунку 1.3.

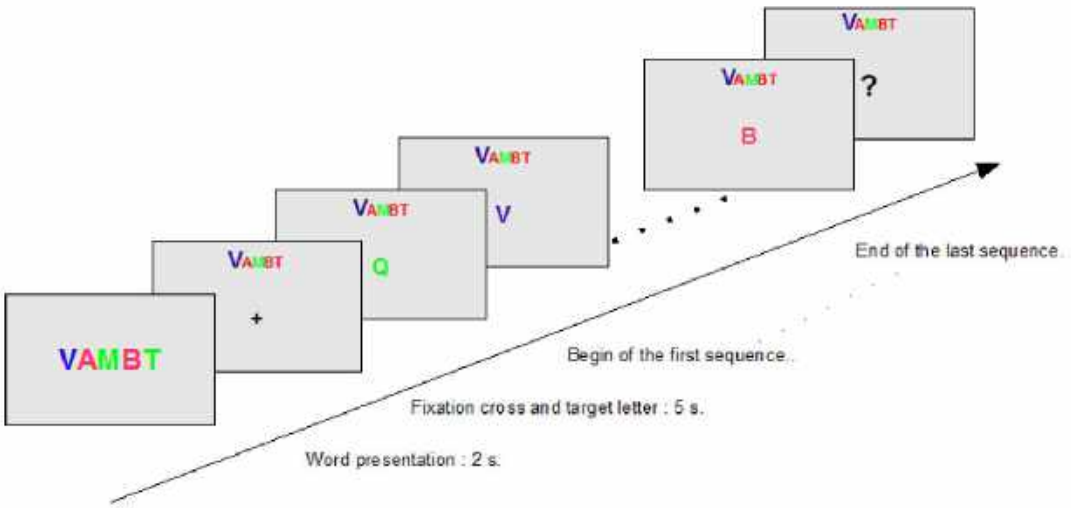

Рисунок 1.3 – Принцип подачі тексту методом RSVP: кожне слово з'являється по центру екрана та замінюється наступним [9]

Переваги методу RSVP полягають насамперед у зміні способу подачі тексту. Очам не потрібно рухатися між словами – усе відбувається в одній точці, що знімає потребу в саккадах і зменшує втому. Подача слів відбувається з

фіксованим темпом (200, 300, 400+ слів/хв), який користувач може точно контролювати. Формат добре підходить для мобільних пристроїв: не потрібні великі текстові блоки, немає залежності від верстки, шрифту чи інтерліньяжу – важливим стає лише сам зміст.

Водночас метод має низку обмежень. У стандартній реалізації складно миттєво повернутися до попередніх слів, що збільшує навантаження на робочу пам'ять. Через відсутність видимого контексту навколо поточного слова на високих швидкостях може погіршуватися розуміння: складні або довгі слова, а також важкі речення важче сприймати без можливості затримати погляд чи перечитати. Окремою проблемою є одноманітність подачі – відсутність абзаців, виділень чи структурних підказок.

Саме тому останніми роками RSVP активно адаптують. Запроваджуються:

– виділення оптимальної точки розпізнавання слова (ORP) [10];

– динамічна зміна швидкості (уповільнення на складних фрагментах, пришвидшення на простих);

– довші паузи після розділових знаків;

– мікроанімації для природнішого переходу між словами;

– подача не лише окремих слів, а й коротких стійких фраз.

Ці удосконалення роблять метод комфортнішим і ближчим до звичайного читання.

У мобільному середовищі RSVP має особливі переваги. Невеликий екран традиційно змушує масштабувати текст, прокручувати сторінки, робити більше фіксацій і стежити за рядками. При RSVP текст подається в одній точці, користувачу не потрібно взаємодіяти з інтерфейсом – достатньо обрати швидкість і зосередитися на змісті. Такий формат забезпечує стабільний ритм читання, не залежить від верстки, не перевантажує зорову систему й добре підходить для коротких сесій «на ходу».

У підсумку читання базується на взаємодії зору, уваги, робочої пам'яті та семантичної обробки, і кожен із цих механізмів має свої межі. RSVP не скасовує

цих обмежень, але пропонує інший спосіб подачі тексту, який краще відповідає умовам мобільного використання: покадровий показ слів допомагає утримувати концентрацію, підтримувати рівний темп і максимально ефективно використовувати екран. Тому цей метод є логічною основою для мобільного застосунку швидкісного читання, який поєднує природні особливості зорового сприйняття з можливостями сучасних технологій.

1.2 Огляд і аналіз методів та засобів розробки мобільного застосунку для швидкісного читання на основі RSVP для вирішення проблеми дослідження

Розробка мобільних застосунків сьогодні охоплює декілька технологічних напрямів, кожен із яких має свої можливості, обмеження та сфери використання. Для системи швидкісного читання, де важливо забезпечити плавність анімації, точність таймінгу, стабільний рендеринг тексту та низьку затримку оновлення інтерфейсу, вибір технології має вирішальне значення. Саме тому перед початком створення застосунку необхідно проаналізувати основні підходи до мобільної розробки та оцінити, наскільки вони відповідають вимогам RSVP.

Першим і найбільш класичним підходом є нативна розробка, яка передбачає створення окремих версій застосунку для iOS та Android. Для iOS використовують мову Swift та інструменти Apple – SwiftUI і UIKit, які забезпечують максимальний рівень продуктивності та найкращу інтеграцію з операційною системою. Офіційна документація Apple описує Swift як мову, оптимізовану для високої швидкості та стабільності роботи [11]. Для Android нативною мовою є Kotlin, яку Google позиціонує як основний інструмент для сучасної Android-розробки [12]. Нативний підхід забезпечує найкращу продуктивність, оскільки код виконується без додаткових проміжних шарів. Це дозволяє досягти стабільної кадрової частоти, точно керувати таймерами та забезпечити плавний інтерфейс навіть на слабких пристроях. Проте головний

недолік нативної розробки – необхідність підтримувати дві незалежні кодові бази, що суттєво збільшує час і вартість створення застосунку. Для академічного чи індивідуального проєкту це може бути надто обтяжливо.

Паралельно з нативним підходом існують гібридні технології, такі як Ionic або Apache Cordova. У таких фреймворках застосунок фактично працює всередині веббраузера, а інтерфейс будується за допомогою HTML, CSS і JavaScript. Це робить розробку простою й доступною для веброзробників, а також дозволяє створювати одну кодову базу для всіх платформ одразу. Однак гібридні рішення мають суттєві обмеження щодо продуктивності, оскільки рендеринг UI відбувається у WebView, який не забезпечує стабільних 60 FPS на всіх пристроях. Це підтверджують офіційні рекомендації Ionic, де зазначено, що складні анімації можуть працювати нестабільно на слабких девайсах [13]. Для RSVP, де текст оновлюється десятки разів на секунду, будь-які мікропригальмовування помітні, а затримки таймерів можуть спотворювати темп читання. Тому гібридний підхід не є оптимальним із точки зору точності та плавності.

Компромісом між нативністю та універсальністю стали кросплатформні фреймворки, серед яких сьогодні найбільш популярні React Native, .NET MAUI та Flutter. React Native працює завдяки тому, що логіка застосунку виконується мовою JavaScript, а інтерфейс рендериться через нативні компоненти платформи. Однак взаємодія між JS-логікою та нативною частиною здійснюється через спеціальний «міст» (JS Bridge). У документації React Native зазначено, що цей механізм може спричиняти затримки під час виконання частих оновлень інтерфейсу [14]. Для застосунку, який змінює слово на екрані 5-10 разів на секунду, цей міст може стати джерелом мікролагів. Досвід розробників також свідчить, що на слабких Android-пристроях React Native може втрачати стабільність частоти кадрів, що є критичним недоліком для швидкісного відображення тексту.

Схожа ситуація спостерігається у .NET MAUI – сучасного фреймворку від Microsoft. Хоча MAUI і пропонує єдину кодову базу, а рендеринг частково

використовує нативні компоненти, екосистема фреймворку ще відносно молода. Як зазначено у документації Microsoft [15], MAUI все ще розвивається, а деякі аспекти продуктивності можуть бути нестабільними, особливо при роботі з анімаціями та точними таймерами.

На цьому тлі особливо вирізняється Flutter – кросплатформний фреймворк від Google, який працює за зовсім іншою моделлю, ніж більшість конкурентів. Flutter не використовує нативні UI-компоненти і не покладається на JavaScript-міст, як це робить React Native. Натомість він рендерить увесь інтерфейс самостійно завдяки власному високопродуктивному графічному рушію Skia. Саме Skia використовується у Google Chrome, Android, ChromeOS, Flutter Web і в багатьох інших системах, що вимагають стабільних 60-120 FPS [16].

Завдяки цьому Flutter працює як повноцінний ігровий рушій для інтерфейсу: кожен елемент UI перерисовується напряму на GPU, без жодних проміжних шарів. Це усуває затримки, характерні для JavaScript bridge у React Native або WebView в гібридних рішеннях (рис. 1.4).

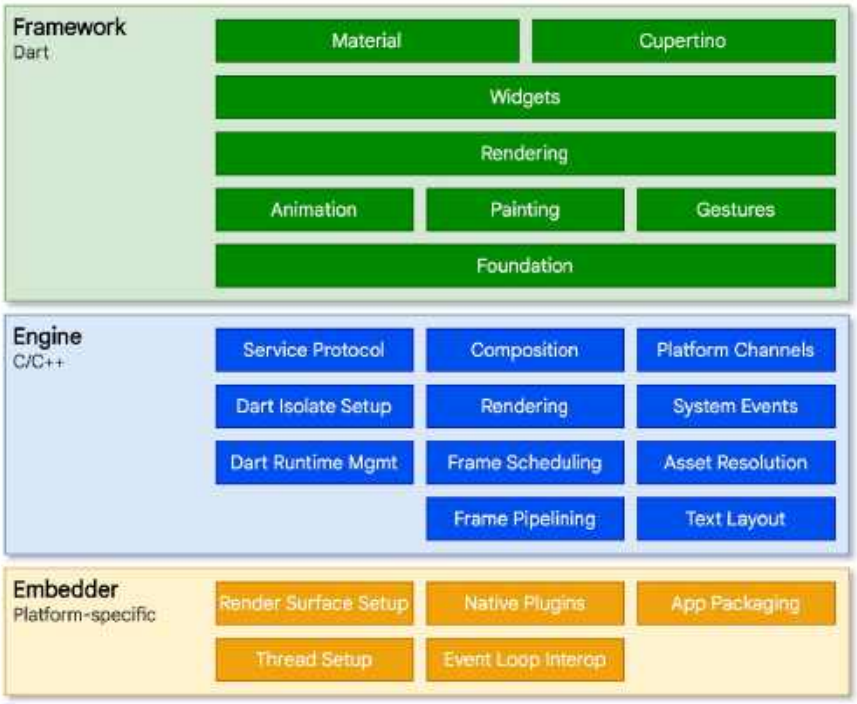

Рисунок 1.4 – Архітектурна схема Flutter [17]

На рисунку 1.6 показано архітектуру Flutter, де добре видно ключову особливість фреймворку – повністю власний графічний конвеєр на основі Skia, який напряму відтворює інтерфейс на GPU без JavaScript-містів чи залежності від нативних UI-компонентів.

Для розробки мобільного застосунку швидкісного читання на основі RSVP це має принципове значення. RSVP вимагає максимально точного контролю часу показу слова: інтерфейс повинен оновлюватися десятки разів за секунду, і навіть мікрозатримки можуть порушувати ритм читання. Flutter забезпечує стабільний FPS та мінімізує затримки саме тому, що не залежить від комунікації між різними потоками чи мовами.

Важливо й те, що Flutter пропонує один із найгнучкіших механізмів роботи з текстом.

Він рендериться через TextPainter, що дозволяє:

– точно визначати позицію кожного слова на екрані;

– програмно виділяти оптимальну точку розпізнавання слова (ORP);

– створювати власні анімації появи та зникнення;

– контролювати інтервали між словами з мілісекундною точністю;

– застосовувати плавні переходи між станами;

– відображати великі обсяги тексту без втрати продуктивності.

Жоден інший мобільний фреймворк сьогодні не дає такої комбінації продуктивності та гнучкості у роботі з текстом.

Крім того, Flutter має AnimationController – механізм, який дозволяє керувати таймерами та анімаціями з дуже високою точністю. Це робить його практично ідеальним для систем швидкісного подання тексту, де потрібно з високою стабільністю підтримувати обраний користувачем темп читання (наприклад, 300 чи 600 слів на хвилину).

Популярність Flutter у світі зростає вже кілька років поспіль, і це добре помітно в глобальних опитуваннях і аналітичних звітах. На відміну від багатьох технологічних «трендів», які швидко зникають, Flutter демонструє стабільно високу динаміку використання – як серед індивідуальних розробників, так і

серед великих компаній. Це зумовлено поєднанням продуктивності, простоти розробки та можливістю створювати інтерфейси, які працюють однаково передбачувано на iOS та Android.

Одним із найбільш авторитетних джерел для оцінки популярності технологій залишається щорічне опитування StackOverflow Developer Survey. У виданні 2023 та 2024 років Flutter стабільно входить у трійку найулюбленіших інструментів серед мобільних розробників. Більше того, відсоток тих, хто хоче почати вивчати Flutter, продовжує зростати, що свідчить про довгострокову перспективність фреймворку [18].

У свою чергу, аналітичні платформи, такі як Statista, демонструють ще більш переконливу картину. За даними глобального дослідження мобільної розробки, Flutter став найпопулярнішим кросплатформним фреймворком у світі, охопивши близько 46 % усіх розробників, які працюють з кросплатформними мобільними технологіями. Для порівняння, React Native у тому ж дослідженні має 32 % (рис. 1.5). Такий розрив пояснюється вищою продуктивністю Flutter та відсутністю проблем, пов'язаних із JavaScript bridge – фактор, який особливо відчутний у проектах з високою частотою оновлення UI, таких як RSVP.

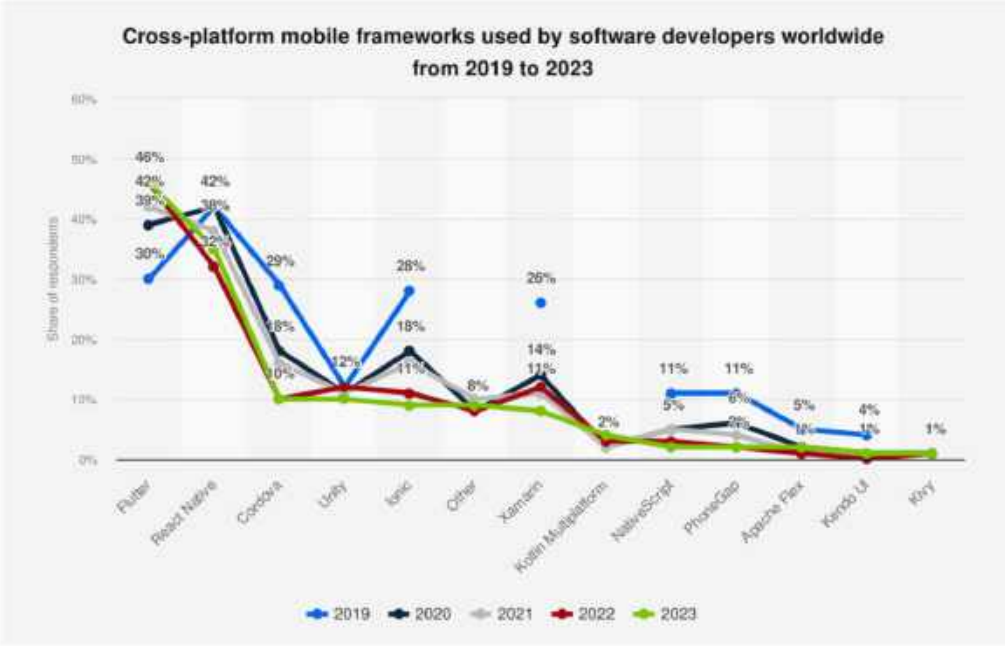

Рисунок 1.5 – Графік використання кросплатформних фреймворків [19]

Такий рівень присутності Flutter у світовій індустрії важливий не лише як показник моди чи короткострокового тренду. Популярність фреймворку підтверджується реальним фактом: ним користуються великі компанії, він має потужну підтримку з боку Google, а спільнота розробників продовжує активно зростати. Це означає, що Flutter не просто зручний технічно – він стабільний, перевірений практикою й має довгострокову траєкторію розвитку.

Для мобільного застосунку швидкісного читання це має пряме значення. Технологія, яку ми обираємо, повинна забезпечувати прогнозовану роботу з текстом, точне відтворення анімацій, стабільний рендер і можливість масштабування в майбутньому. Flutter поєднує всі ці властивості: він працює на GPU, гарантує високу плавність інтерфейсу, дозволяє точно контролювати таймінги показу слів і не залежить від нативних UI-компонентів, які можуть відрізнятися між платформами.

Саме тому вибір Flutter – це не просто технологічне рішення, а стратегічно обґрунтований крок. Він забезпечує і технічну якість, і надійність екосистеми, що є критично важливим для застосунку, який має працювати швидко, стабільно та однаково добре на всіх сучасних смартфонах.

Окремо варто згадати про мову програмування Dart, оскільки вона є невід'ємною частиною Flutter і безпосередньо впливає на технічні можливості платформи. Dart розроблена Google спеціально для створення високопродуктивних інтерфейсів, і саме тому Flutter використовує її як основну мову. Її особливості мають ключове значення для таких задач, як точне відтворення таймінгів у методі RSVP та обробка тексту в реальному часі.

Однією з причин ефективності Dart є поєднання двох режимів компіляції – JIT (Just-in-Time) для швидкої розробки та AOT (Ahead-of-Time) для оптимізованого виконання. У релізних збірках Dart-компілятор генерує нативний ARM-код, що дозволяє Flutter-додаткам працювати з продуктивністю, близькою до нативної. Для RSVP це критично, адже метод вимагає відображення слів із точністю до мілісекунд, без затримок та фризів.

Ще одна важлива особливість – ізоляти (Isolates), механізм конкурентності, який забезпечує паралельну обробку даних без блокування основного UI-потоку. Це дає можливість обробляти великі текстові файли, розбивати текст на слова, підсвічувати ORP або рахувати паузи для пунктуації на окремих ізолятах, не впливаючи на плавність анімації.

Dart також має null safety, сучасну типізовану систему, потужні можливості асинхронності (async/await) та чистий синтаксис, що спрощує розробку складної бізнес-логіки та робить код більш надійним. На практиці це означає менше помилок при роботі з текстовими даними та стабільнішу поведінку застосунку під час швидкісного читання.

З огляду на ці властивості Dart є не просто мовою для Flutter, а важливим технологічним фактором, який дозволяє реалізувати RSVP на мобільних пристроях із високою точністю, плавністю та стабільністю.

Проведений аналіз показує, що серед усіх сучасних підходів до мобільної розробки саме Flutter найповніше відповідає вимогам до системи швидкісного читання на основі RSVP. Нативні рішення забезпечують максимальну продуктивність, але вимагають подвійної розробки; гібридні фреймворки поступаються в плавності та точності рендерингу, що критично для частих оновлень тексту; інші кросплатформні системи мають архітектурні обмеження або недостатній рівень стабільності. Flutter вирізняється власним графічним рушієм Skia, відсутністю проміжних шарів, високою продуктивністю на GPU та гнучкими інструментами для роботи з анімаціями й текстом, що робить його особливо придатним для задач, де важлива мілісекундна точність.

У поєднанні з можливостями мови Dart – зокрема AOT-компіляцією, ізолятами та асинхронною моделлю – Flutter забезпечує платформу, яка дозволяє не лише стабільно відтворювати текст у швидкому темпі, а й масштабувати застосунок, підтримувати високу якість інтерфейсу та розширювати функціональність у майбутньому. Саме тому вибір Flutter є технічно обґрунтованим і стратегічно правильним кроком у межах розробки мобільного застосунку для швидкісного читання.

1.3 Постановка завдання на кваліфікаційну роботу магістра

Метою кваліфікаційної роботи є розробка та експериментальна перевірка мобільного застосунку Rapid Read для швидкісного читання текстів на основі методу Rapid Serial Visual Presentation (RSVP) на платформах Android та iOS.

Для досягнення поставленої мети у роботі необхідно розв'язати такі завдання:

– виконати огляд і аналіз теоретичних підходів до процесу читання, когнітивних обмежень (увага, робоча пам'ять, семантична інтеграція) та сучасних технологій швидкісного подання тексту, зокрема методу RSVP;

– проаналізувати існуючі підходи до розробки мобільних застосунків (нативні, гібридні, кросплатформні рішення) та обґрунтувати вибір стеку Flutter/Dart для реалізації системи швидкісного читання;

– сформулювати функціональні та нефункціональні вимоги до мобільного застосунку Rapid Read (режим RSVP-читання, робота з бібліотекою файлів, налаштування, статистика, PIN-захист, офлайн-режим);

– розробити функціональну та інформаційну моделі системи із використанням UML-діаграм, визначити основні сутності, їхні атрибути та зв'язки;

– спроєктувати архітектуру застосунку та реалізувати основні програмні модулі: модуль роботи з файлами, модуль підготовки й подання тексту в режимі RSVP, модуль збереження прогресу та статистики, модуль керування налаштуваннями й PIN-кодом;

– виконати практичну реалізацію мобільного застосунку Rapid Read на базі Flutter/Dart із урахуванням вимог до продуктивності, плавності анімацій та зручності інтерфейсу;

– провести тестування розробленої системи, а також експериментальне дослідження роботи застосунку (оцінка швидкості читання, стабільності інтерфейсу, зручності використання) та проаналізувати отримані результати.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ШВИДКІСНОГО ЧИТАННЯ НА ОСНОВІ RSVP

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Розроблюваний мобільний застосунок Rapid Read орієнтований на швидкісне читання текстів у режимі Rapid Serial Visual Presentation (RSVP) на платформах iOS та Android. На відміну від класичних «читалок», де користувач сам прокручує сторінки, у нашому випадку система сама подає слова одне за одним у заданому темпі. Це накладає специфічні вимоги як до функціональності програми, так і до її продуктивності та архітектури.

З погляду функціоналу застосунок має покривати повний життєвий цикл роботи з текстом – від завантаження файлу до завершення сеансу читання. До основних функціональних вимог належать:

– підтримка імпорту текстових файлів (наприклад, у форматі .txt) з пам'яті пристрою та додавання їх до внутрішньої бібліотеки;

– відображення списку доступних документів із можливістю обрати файл для читання, переглянути його статус (новий, у процесі, прочитаний) та за потреби видалити;

– збереження прогресу читання для кожного файлу (індекс останнього прочитаного слова або відсоток проходження), щоб користувач міг продовжити з того місця, де зупинився.

Ключовим режимом роботи системи є режим читання RSVP. Тут застосунок повинен відображати текст у вигляді послідовного показу слів у центрі екрана, забезпечуючи користувачеві базовий контроль над потоком. Серед вимог до цього режиму:

– можливість запустити сеанс читання, поставити його на паузу та відновити;

– зміна швидкості подання слів у певному діапазоні (наприклад, від повільного тренувального режиму до високих значень для досвідчених користувачів);

– перехід до наступного або попереднього слова, а також швидке повернення на початок тексту;

– автоматичне відновлення сеансу після перезапуску застосунку з останньої збереженої позиції.

Окремий блок становлять налаштування інтерфейсу та читання, які мають бути доступні без зміни коду. Користувач очікує, що застосунок дозволить:

– обирати мову інтерфейсу (наприклад, українська / англійська);

– змінювати розмір і тип шрифту в зоні читання;

– перемикати тему оформлення (світла/темна) залежно від умов освітлення та особистих уподобань;

– зберігати всі налаштування локально і застосовувати їх автоматично при наступних запусках.

Важливо, щоб зміни параметрів одразу відображалися в UI, а сам інтерфейс оновлювався реактивно через механізми керування станом.

У застосунку також передбачені вимоги до безпеки доступу: користувач може встановити PIN-код, який захищає вхід у програму. Для цього система повинна підтримувати:

– початкове налаштування PIN-коду (або свідомий відмову від нього);

– зміну коду в налаштуваннях;

– видалення PIN-коду за бажанням користувача.

Це робить застосунок більш персоналізованим і підходящим для сценаріїв, де користувач не хоче, щоб хтось інший мав легкий доступ до його бібліотеки чи статистики читання.

Важливою частиною функціоналу є збір та відображення статистики. Система має фіксувати базові показники активності користувача:

– загальну кількість прочитаних слів;

– сумарний час, проведений у режимі читання;

– розподіл читання за днями чи окремими сесіями (наприклад, у вигляді простої діаграми або списку сеансів).

Ця інформація показує прогрес користувача і дає змогу оцінити ефективність тренування навичок швидкісного читання.

Окремо формулюються нефункціональні вимоги, пов'язані з продуктивністю та зручністю користування. Для RSVP-клієнта критичною є плавність змін слів: інтерфейс має оновлюватися без помітних затримок та ривків навіть на пристроях із середніми характеристиками. Іншими словами, система повинна підтримувати стабільний ритм показу, щоб користувач сприймав текст як рівномірний потік. Так само важливо, щоб реакція інтерфейсу на дії (натискання кнопок паузи, зміни швидкості, переходу до іншого файлу) була максимально швидкою і не відволікала від читання.

З точки зору зберігання даних застосунок має працювати локально і не залежати від наявності інтернет-з'єднання. До вимог до надійності даних належать:

– локальне збереження бібліотеки файлів, прогресу читання, статистики та налаштувань;

– використання стійкого механізму зберігання (наприклад, локальної бази даних для складних структур і схеми «ключ-значення» для простих параметрів);

– коректне відновлення стану після перезапуску програми.

З погляду зручності інтерфейсу (UX) застосунок має залишатися максимально простим і не перевантаженим. Основний фокус користувача – зона відображення слова та елементи керування темпом. Отже, інтерфейс повинен забезпечувати:

– достатній розмір шрифту та контраст між текстом і фоном;

– мінімальну кількість відволікаючих елементів біля зони читання;

– доступ до ключових дій (play/pause, зміна швидкості, переключення файлів) у один-два торкання.

Окремою групою є архітектурні вимоги. Система повинна бути побудована модульно, з чітким розділенням відповідальностей. Це означає наявність окремих модулів для:

– роботи з файлами (імпорт, зберігання, стан кожного документа);

– підготовки та подання тексту в режимі читання (ядро RSVP-логіки);

– зберігання та обробки статистики;

– роботи з налаштуваннями (мова, тема, шрифт, PIN-код).

Модульність полегшує супровід та розвиток системи, дозволяє вносити зміни в один блок без критичного впливу на інші. Керування станом інтерфейсу має бути організовано так, щоб UI автоматично реагував на зміну даних, а архітектура дозволяла в майбутньому розширювати функціональність (наприклад, додавати нові режими читання або розширену аналітику) без повної переробки основи.

Сукупність цих вимог формує цілісне уявлення про те, якою має бути система швидкісного читання з точки зору можливостей, стабільності та зручності. На їх основі в наступних підпунктах будуть побудовані функціональні та інформаційні моделі, а також обґрунтовані конкретні алгоритми й засоби реалізації.

У процесі проєктування програмних систем важливо описувати не лише набір функцій, а й те, як саме користувач взаємодіє із системою та які сценарії роботи є основними. Для цього використовують функціональні моделі та діаграми, що дозволяють на наочному рівні показати поведінку застосунку, його реакцію на дії користувача та послідовність кроків у типових сценаріях. У сучасній практиці проєктування поширеним підходом є використання UML (Unified Modeling Language), зокрема діаграм варіантів використання та діаграм діяльності, які допомагають формалізувати вимоги ще до етапу безпосередньої реалізації.

Для мобільного застосунку швидкісного читання на основі RSVP функціональне моделювання має особливе значення, оскільки система включає декілька логічних підсистем: роботу з файлами, підготовку тексту, режим

читання, модуль налаштувань, статистику, керування доступом тощо. Використання діаграм дозволяє чітко виділити основні варіанти використання (читання тексту, зміна швидкості, керування бібліотекою, перегляд статистики) та описати їх у вигляді зрозумілих сценаріїв. Це спрощує узгодження вимог, полегшує подальший перехід до розробки архітектури та дає можливість наочно показати логіку роботи застосунку в рамках кваліфікаційної роботи.

На діаграмі варіантів використання (рис. 2.1) зображено основні сценарії взаємодії користувача з мобільним застосунком швидкісного читання на основі методу RSVP. Єдиним актором виступає користувач, який працює із системою через кілька ключових груп функцій.

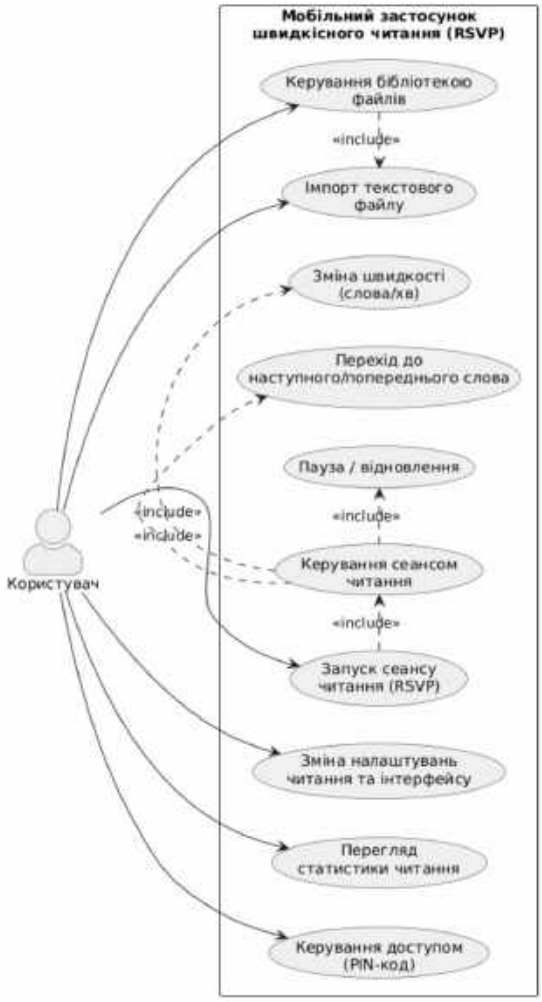

Рисунок 2.1 – Діаграма варіантів використання мобільного застосунку швидкісного читання

По-перше, користувач може керувати бібліотекою файлів: переглядати список наявних документів, обирати текст для читання, а також імпортувати нові текстові файли з пам'яті пристрою. Операція імпорту виділена окремим варіантом використання, який включається до загального сценарію керування бібліотекою.

По-друге, основним сценарієм є запуск сеансу читання в режимі RSVP. Після його запуску користувач отримує доступ до керування сеансом: може ставити читання на паузу та відновлювати його, змінювати швидкість подання слів (у словах за хвилину), а також переходити до наступного чи попереднього слова. Ці дії подані як окремі варіанти використання, що включаються до загального сценарію «Керування сеансом читання», оскільки всі вони реалізуються в межах активного сеансу.

Крім того, користувач має можливість змінювати налаштування системи: параметри читання (швидкість за замовчуванням, шрифт, тема оформлення) та загальні налаштування інтерфейсу. Окремий варіант використання відведено для перегляду статистики читання, де відображаються узагальнені показники активності. Також передбачено керування доступом до застосунку за допомогою PIN-коду: встановлення, зміна або видалення PIN, що підвищує приватність роботи з особистою бібліотекою.

Таким чином, діаграма відображає повний набір основних дій, які може виконувати користувач у системі, та показує логічні зв'язки між базовими сценаріями (керування бібліотекою, читання, налаштування, статистика, безпека). Це створює основу для подальшої деталізації процесів на рівні діаграм діяльності й структурних моделей системи.

На спрощеній діаграмі діяльності (рис. 2.2) показано основні етапи роботи мобільного застосунку під час сеансу читання в режимі RSVP без зайвих технічних деталей. Процес починається з того, що користувач обирає текстовий файл у бібліотеці.

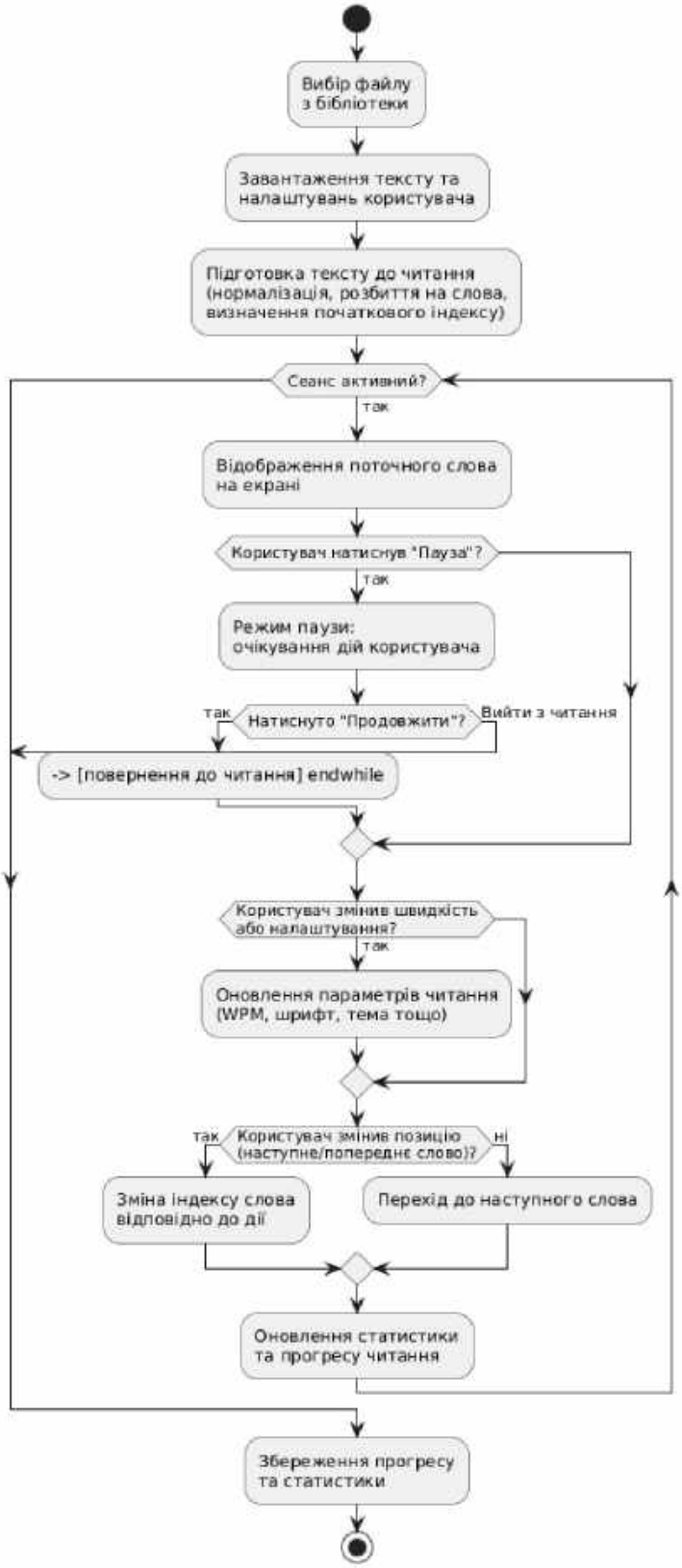

Рисунок 2.2 – Спрощена activity-діаграма процесу читання тексту в режимі RSVP

Після вибору система завантажує вміст файлу разом із збереженими налаштуваннями користувача (швидкість читання, тема, шрифт, мова інтерфейсу) і виконує початкову підготовку тексту: нормалізує його, розбиває на послідовність слів та визначає початковий індекс, з якого слід розпочати читання. Якщо раніше було збережено прогрес, система може відновити позицію, але цей крок уже приховано всередині етапу підготовки, щоб не ускладнювати діаграму.

Основний цикл відображає сам сеанс читання. На кожній ітерації система показує поточне слово на екрані в зоні читання. Далі можливі кілька варіантів розвитку подій, залежно від дій користувача. Якщо натиснуто кнопку «Пауза», застосунок переходить у режим очікування: оновлення слова зупиняється, і система чекає, поки користувач не вирішить, що робити далі. Якщо обрано «Продовжити», сеанс повертається до звичайного циклу читання з того самого слова; якщо ж користувач вирішує вийти з режиму читання, сеанс переривається, і система переходить до фінального етапу збереження прогресу.

Крім паузи, користувач може в будь-який момент змінити параметри читання – наприклад, швидкість у словах за хвилину або деякі візуальні налаштування. У такому випадку система оновлює відповідні параметри та продовжує сеанс уже з новими значеннями, не вимагаючи перезапуску. Окремо обробляються дії, пов'язані зі зміною позиції в тексті: перехід до наступного чи попереднього слова. Якщо користувач явно задає такі дії, застосунок змінює індекс поточного слова відповідно до команди; якщо ні – індекс збільшується автоматично, і система переходить до наступного слова в послідовності.

На кожній ітерації циклу, після оновлення слова, застосунок актуалізує статистику читання та прогрес: фіксує, скільки слів прочитано, як змінюється тривалість сеансу, на якій позиції у файлі зараз перебуває користувач. Коли сеанс більше не активний (наприклад, користувач вийшов із режиму читання або досягнуто кінця тексту), цикл завершується. На фінальному етапі система зберігає останній прогрес та накопичену статистику, щоб ці дані були доступні при наступному запуску застосунку.

У такому вигляді діаграма залишається компактною, але водночас відображає всі ключові кроки: вибір файлу, підготовку тексту, відображення слів, керування паузою, зміною швидкості та позиції, а також збереження результатів. Це дозволяє зрозуміло представити логіку роботи RSVP-модуля без перенасичення схемою надмірними деталями.

Щоб реалізувати мобільний застосунок швидкісного читання, недостатньо лише описати сценарії взаємодії користувача із системою. Важливо також визначити, які саме дані зберігає і обробляє застосунок, як ці дані структуровані та які сутності між собою пов'язані. Такий опис прийнято оформлювати у вигляді інформаційної або структурної моделі системи, де виділяються основні об'єкти предметної області, їхні атрибути та зв'язки.

У випадку RSVP-застосунку до ключових сутностей належать текстові файли, параметри читання, поточний стан сеансу, статистика та налаштування користувача. На основі цих сутностей формується модель, яка лягає в основу реалізації класів у коді (моделі, провайдери, сервіси зберігання). Для наочності така модель зазвичай подається у вигляді діаграми класів UML, що дозволяє показати структуру даних та їхню взаємодію в компактній та зрозумілій формі. У цьому підпункті буде описано основні об'єкти системи та наведено їх структурні зв'язки.

На діаграмі класів (рис. 2.3) подано спрощену інформаційну модель мобільного застосунку швидкісного читання, яка відображає основні сутності системи та зв'язки між ними. Модель орієнтується на реальну структуру даних у застосунку Rapid Read, але подана у більш узагальненому вигляді, без прив'язки до конкретної реалізації у коді.

У центрі моделі знаходиться клас Library, який представляє собою внутрішню бібліотеку користувача. Він містить колекцію об'єктів типу FileDocument. Кожен FileDocument відповідає окремому текстовому файлу, доступному для читання. Для нього зберігаються ідентифікатор, назва, шлях до файлу, загальна кількість слів, індекс поточного прогресу (останнє прочитане слово), мова тексту та дата додавання. Це дозволяє застосунку не лише

відкривати файл, а й відновлювати читання з потрібного місця та відображати статус документа в списку (новий, у процесі, прочитаний).

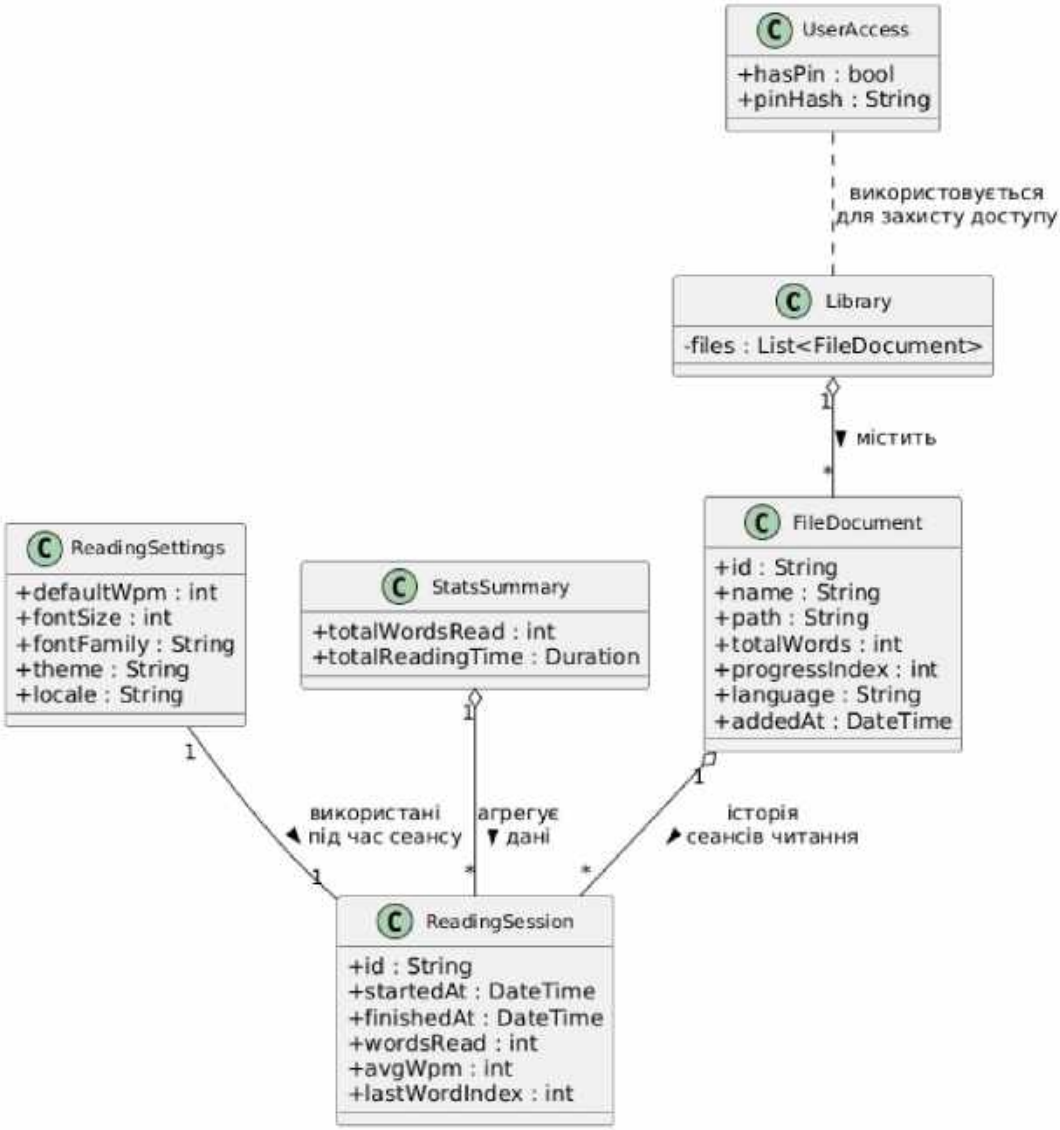

Рисунок 2.3 – Діаграма класів інформаційної моделі мобільного застосунку швидкісного читання

З кожним файлом пов'язана історія читання у вигляді об'єктів класу ReadingSession. Один FileDocument може мати багато ReadingSession, що відображено зв'язком «один до багатьох». Кожен сеанс містить інформацію про час початку та завершення, кількість прочитаних слів, середню швидкість (WPM) та індекс останнього прочитаного слова в рамках саме цього сеансу.

Така структура дозволяє аналізувати, як користувач працює з конкретним текстом, та будувати більш деталізовану статистику.

Клас ReadingSettings описує налаштування, які впливають на режим читання: базову швидкість (defaultWpm), розмір шрифту, обраний шрифт, тему оформлення (світла або темна) та мову інтерфейсу. На діаграмі показано зв'язок між ReadingSettings і ReadingSession: кожний сеанс читання відбувається з певним набором налаштувань. У реальній реалізації це може бути як посилання на поточні глобальні налаштування, так і збереження їхнього стану на момент сеансу.

Для узагальнення даних про активність користувача використовується клас StatsSummary, який агрегує окремі сеанси. У ньому зберігається загальна кількість прочитаних слів та сумарний час читання. Зв'язок «один до багатьох» між StatsSummary і ReadingSession показує, що статистика формується на основі масиву сеансів, а не обчислюється ізольовано.

Окремо виділено клас UserAccess, що відповідає за налаштування доступу до застосунку. У найпростішому варіанті зберігається інформація про те, чи увімкнено PIN-захист (hasPin) та хеш самого PIN-коду (pinHash). На діаграмі показано асоціативний зв'язок між UserAccess і Library, який ілюструє, що налаштування доступу застосовуються до всієї бібліотеки та пов'язаних даних користувача. Це підкреслює, що захист стосується не окремих файлів, а роботи з застосунком загалом.

Наведена інформаційна модель охоплює основні дані, з якими працює система: текстові файли, параметри читання, окремі сеанси, агреговану статистику та налаштування доступу.

Алгоритмічну частину мобільного застосунку можна умовно поділити на три послідовні етапи: підготовка тексту, організація сеансу читання та оновлення прогресу зі статистикою. Кожен із цих етапів має свою логіку, але всі вони працюють разом, щоб користувач просто бачив слова по центру екрана в потрібному темпі.

Першим кроком є підготовка тексту. Після того, як користувач обирає файл у бібліотеці, застосунок зчитує його вміст і приводить до більш зручного для обробки вигляду. З тексту прибираються зайві пропуски, некоректні символи або «подвійні» переноси рядків. Потім цей очищений текст розбивається на послідовність слів і розділових знаків. На цьому етапі фактично формується список елементів, які згодом будуть по одному показуватися користувачу в режимі RSVP. Якщо для цього файлу раніше вже був збережений прогрес, система одразу визначає, з якого місця саме потрібно продовжити – тобто початковий індекс поточного слова.

Другий етап стосується організації самого сеансу читання. Коли користувач натискає кнопку «Старт», застосунок переходить у стан активного читання: обирається поточне слово із сформованого списку, воно відображається в центрі екрана, а далі запускається механізм регулярного оновлення. Частота зміни слів залежить від обраної швидкості в словах за хвилину – чим вища швидкість, тим менше часу слово залишається на екрані. Між показом окремих слів система робить короткі паузи; після кожної паузи індекс слова змінюється, і на екран виводиться наступний елемент послідовності. У будь-який момент користувач може поставити читання на паузу, змінити швидкість або перейти до наступного чи попереднього слова – застосунок реагує на ці дії, коригуючи поточний стан, але не перериваючи загальний алгоритм.

Третій важливий аспект – це оновлення прогресу та статистики. Під час сеансу система поступово накопичує інформацію: на якому слові зараз зупинився користувач, скільки слів було показано, скільки часу загалом тривало читання. Коли користувач виходить із режиму читання або доходить до кінця тексту, ці дані зберігаються: оновлюється прогрес для конкретного файлу, а також загальна статистика – наприклад, сумарна кількість прочитаних слів та тривалість читання за день чи за всі сеанси. Надалі ці показники використовуються для відображення аналізу читання на окремому екрані й для

того, щоб при повторному відкритті файлу система могла одразу запропонувати продовжити з потрібного місця.

У такому вигляді алгоритмічні підходи залишаються відносно простими: текст готується один раз, далі працює цикл показу слів із можливістю керування з боку користувача, а результати фіксуються й зберігаються. Цього достатньо, щоб забезпечити передбачувану і стабільну поведінку застосунку в режимі RSVP та створити основу для подальших розширень логіки без повної переробки вже наявних механізмів.

Отже, було сформовано цілісне бачення того, як саме має працювати мобільний застосунок швидкісного читання Rapid Read на рівні вимог, моделей і алгоритмів. Визначено функціональні та нефункціональні вимоги, окреслено структурні особливості системи та побудовано UML-діаграми, що відображають основні сценарії її використання, логіку роботи та модель даних. У межах алгоритмічної частини описано ключові етапи обробки тексту, організацію сеансу читання та механізми фіксації статистики, які забезпечують стабільність і передбачуваність роботи RSVP-модуля.

2.2 Практична реалізація об'єкта проектування

Мобільний застосунок Rapid Read реалізовано засобами фреймворку Flutter з використанням мови програмування Dart. Архітектура застосунку побудована за принципом розділення відповідальностей між окремими модулями, що спрощує супровід, тестування та подальший розвиток системи. Логіка роботи з даними, керування станом інтерфейсу, доступ до локального сховища та візуальна частина виділені в окремі блоки, які взаємодіють між собою через чітко визначені інтерфейси.

Основна структура каталогу lib має модульний характер. У папці blocks зосереджено бізнес-логіку, розділену за доменами: робота з файлами (blocks/file), режим читання (blocks/reading), статистика (blocks/stats), тема оформлення (blocks/theme), шрифти (blocks/font) та локалізація інтерфейсу

(blocks/locale). Кожен блок містить модель даних, провайдер або контролер стану, а також супровідні класи (репозиторії, хелпери тощо). Папка services містить сервіси нижчого рівня – зокрема, DBService для роботи з локальною базою даних (Hive) та FileService для читання текстових файлів із файлової системи пристрою. Графічний інтерфейс згруповано в каталозі ui, де виділено окремі екрани (ui/widgets/screens) та багаторазово використовувані компоненти. Спільні константи, типи та допоміжні класи винесено до каталогу common.

Точка входу в застосунок реалізована у файлі main.dart. На етапі старту виконуються початкова ініціалізація Flutter (WidgetsFlutterBinding.ensureInitialized()), налаштування локальної бази даних Hive, реєстрація адаптерів для моделей файлів, а також ініціалізація сервісу DBService, який відповідає за збереження бібліотеки документів, прогресу читання та частини налаштувань користувача. Після цього створюються та ініціалізуються провайдери стану, що реалізують централізоване керування даними на основі патерну ChangeNotifier і пакету provider. На рівні кореня застосунку підключаються провайдери для локалі інтерфейсу, бібліотеки файлів, режиму читання, статистики, шрифту та теми оформлення.

У лістингу 2.1 наведено фрагмент функції main(), яка демонструє послідовність стартової ініціалізації застосунку Rapid Read та підключення основних провайдерів стану.

Лістинг 2.1 – Функція main() з ініціалізацією сервісів та провайдерів стану

```
void main() async {

 WidgetsFlutterBinding.ensureInitialized();

 await Hive.initFlutter();

 Hive.registerAdapter(FileModelAdapter());

 await DBService().init();

 final localeProvider = LocaleProvider();
 await localeProvider.loadSavedLocale();
```

```
 runApp(
   MultiProvider(
     providers: [
       ChangeNotifierProvider(create: (_) => localeProvider),
                          ChangeNotifierProvider(create:    (_)    =>
FileProvider()..loadFiles()),
       ChangeNotifierProvider(create: (_) => ReadingProvider()),
       ChangeNotifierProvider(create: (_) => StatsProvider()),
                          ChangeNotifierProvider(create:    (_)    =>
FontProvider()..loadFont()),
       ChangeNotifierProvider(create: (_) => ThemeProvider()),
     ],
     child: const MyApp(),
   ),
 );
}
```

Кінець лістингу 2.1

У цьому фрагменті видно, що перед запуском інтерфейсу виконуються всі необхідні підготовчі кроки: ініціалізація локальної бази даних, реєстрація моделей, завантаження збережених налаштувань локалі, а також створення кореневого контейнера MultiProvider, який ін'єктує в дерево віджетів основні блоки логіки. Зокрема, FileProvider відповідає за завантаження бібліотеки файлів, ReadingProvider – за організацію сеансів читання в режимі RSVP, StatsProvider – за збір та обробку статистики, FontProvider – за параметри шрифту, а ThemeProvider – за вибір теми оформлення.

Графічна оболонка застосунку визначається у класі MyApp, який формує кореневий MaterialApp із підключенням тем, локалізації та маршрутизації. У лістингу 2.2 наведено відповідний фрагмент коду.

Лістинг 2.2 – Клас MyApp із налаштуванням теми, локалізації та навігації

```
class MyApp extends StatelessWidget {
 const MyApp({super.key});

 @override
 Widget build(BuildContext context) {
   final localeProvider = Provider.of<LocaleProvider>(context);
   final themeProvider = Provider.of<ThemeProvider>(context);

   return MaterialApp(
     title: Config.appTitle,
```

```
      theme: DefaultTheme.lightTheme,
      darkTheme: DefaultTheme.darkTheme,
          themeMode: themeProvider.isDarkMode ? ThemeMode.dark :
ThemeMode.light,
      supportedLocales: S.delegate.supportedLocales,
      localizationsDelegates: const [
        S.delegate,
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      locale: localeProvider.locale,
      initialRoute: '/',
      onGenerateRoute: generateRoute,
      debugShowCheckedModeBanner: false,
      home: const SplashRouter(),
    );
  }
}
```

Кінець лістингу 2.2

У цьому класі використовується значення з LocaleProvider для вибору поточної мови інтерфейсу та стан з ThemeProvider для перемикання між світлою та темною темами. Підключено згенерований делегат локалізації S, який забезпечує підтримку кількох мов, а також стандартні делегати Flutter для матеріальних, купертіновських та віджетів інтерфейсу. Поле onGenerateRoute використовує окремо визначений маршрутизатор generateRoute, що спрощує навігацію між основними екранами застосунку (головний екран бібліотеки, екран читання, налаштування, статистика тощо). Початковим екраном є SplashRouter, який виконує стартові перевірки та перенаправляє користувача до потрібного екрану залежно від стану системи (наявність файлів, налаштувань, PIN-коду тощо).

У типовому сценарії після splash-екрана (та, за потреби, введення PIN-коду) користувач потрапляє на головний екран застосунку (рис. 2.4). Тут відображається зведена статистика читання (загальна кількість прочитаних слів, середні показники за тиждень), кнопка «Продовжити читати» для швидкого повернення до останнього тексту, а внизу – панель навігації з переходом до

бібліотеки та інших розділів. Така структура дає змогу одразу побачити власну активність і за один дотик відновити сеанс читання.

Рисунок 2.4 – Головний екран мобільного застосунку Rapid Read із відображенням основної статистики та кнопкою продовження читання

Таким чином, загальна структура проєкту Rapid Read базується на розділенні інтерфейсу та логіки між окремими модулями, централізованому керуванні станом через провайдери та використанні сервісів для доступу до локального сховища й файлової системи. Це створює основу для подальшого розгляду практичної реалізації ключових підсистем: ядра RSVP-читання, роботи з файлами, налаштувань, статистики та механізмів безпеки.

Центральним елементом застосунку Rapid Read є модуль, який відповідає за показ тексту в режимі Rapid Serial Visual Presentation. Саме тут реалізовано логіку послідовного відображення слів, керування швидкістю, паузами та

збереженням прогресу. З погляду архітектури ця функціональність зосереджена в блоці blocks/reading, де розміщено модель стану та провайдер, що керує процесом читання, а також у відповідному екрані інтерфейсу.

2.2.1 Модель стану сеансу читання

Для зручної роботи з даними використовується окрема модель, яка описує, що відбувається під час сеансу RSVP. У спрощеному вигляді вона містить список слів, індекс поточного слова, обрану швидкість та службову інформацію для статистики (ліст. 2.3).

Лістинг 2.3 – Спрощена модель стану сеансу читання

```
class ReadingModel {
  final List<String> words;
  int currentIndex;
  int wpm;
  bool isPlaying;

  ReadingModel({
    required this.words,
    this.currentIndex = 0,
    this.wpm = 300,
    this.isPlaying = false,
  });
}
```

Кінець лістингу 2.3

Також до цієї моделі також належать дані про файл, час початку сеансу, поточний прогрес і додаткові параметри, але для розуміння логіки достатньо продемонструвати базові поля. Саме на цій структурі надалі працює провайдер читання.

2.2.2 Провайдер ReadingProvider та керування потоком слів

Логіка режиму читання реалізована в класі ReadingProvider, який успадковується від ChangeNotifier і виступає єдиною «точкою правди» для всіх віджетів, пов'язаних із RSVP. Провайдер зберігає поточну модель, керує таймером, змінює індекс слова та повідомляє інтерфейс про зміни (ліст. 2.4).

Лістинг 2.4 – Фрагмент провайдера ReadingProvider з методами запуску та паузи

```
class ReadingProvider extends ChangeNotifier {
  ReadingModel? _model;
  Timer? _timer;

  ReadingModel? get model => _model;

  void start(ReadingModel model) {
    _model = model;
    _model!.isPlaying = true;
    _startTimer();
    notifyListeners();
  }

  void pause() {
    _model?.isPlaying = false;
    _timer?.cancel();
    notifyListeners();
  }
```

Кінець лістингу 2.4

Метод start отримує готову модель (список слів, початкову позицію, швидкість), позначає стан як активний (isPlaying = true) та запускає внутрішній таймер. Метод pause зупиняє таймер і переводить провайдер у стан паузи. У роботі ці методи викликаються з екрану читання при натисканні відповідних кнопок.

Основна логіка послідовного показу слів реалізована у приватному методі, який створює періодичний таймер. Інтервал таймера залежить від обраної швидкості в словах за хвилину (ліст. 2.5).

Лістинг 2.5 – Логіка таймера та переходу до наступного слова

```
void _startTimer() {
  _timer?.cancel();
  final intervalMs = (60000 / _model!.wpm).round();

  _timer = Timer.periodic(
    Duration(milliseconds: intervalMs),
    (_) => _nextWord(),
  );
}

void _nextWord() {
```

```
  if (_model == null || !_model!.isPlaying) return;

  if (_model!.currentIndex < _model!.words.length - 1) {
    _model!.currentIndex++;
  } else {
    _finishSession();
  }
  notifyListeners();
}
```

Кінець лістингу 2.5

Кожен тік таймера викликає _nextWord(): якщо слова ще залишилися, індекс збільшується, і провайдер генерує оновлення для інтерфейсу через notifyListeners(). Якщо досягнуто кінця списку, викликається метод завершення сеансу, де фіксується прогрес і оновлюється статистика. За потреби користувач може змінити швидкість читання прямо під час сеансу; у цьому разі провайдер перезапускає таймер з новим інтервалом.

2.2.3 Екран читання та прив'язка до стану

Візуальне відображення режиму RSVP реалізоване на окремому екрані, який підписується на ReadingProvider і реагує на зміни поточного слова, швидкості та статусу відтворення. Центральним елементом є зона показу слова, що розміщена по центру екрану з акцентом на розмірі шрифту та контрасті (ліст. 2.6).

Лістинг 2.6 – Фрагмент екрану читання з відображенням поточного слова

```
class ReadingScreen extends StatelessWidget {
  const ReadingScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Consumer<ReadingProvider>(
      builder: (context, reading, _) {
        final model = reading.model;
        final word = model == null
            ? ''
            : model.words[model.currentIndex];

        return Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
```

```
            Text(
              word,
              textAlign: TextAlign.center,
              style: Theme.of(context).textTheme.displayMedium,
            ),
            const SizedBox(height: 32),
            const _Controls(),
          ],
        );
      },
    );
  }
}
```

Кінець лістингу 2.6

У цьому фрагменті використовується Consumer<ReadingProvider>, який отримує поточну модель з провайдера та відображає слово з індексом currentIndex. Зміна індексу в провайдері автоматично призводить до оновлення віджета, тому користувач бачить нове слово без додаткової логіки у UI.

Блок керування _Controls містить елементи взаємодії: кнопку старт/пауза, регулятор швидкості та, за потреби, кнопки переходу до наступного або попереднього слова. Логіка обробки дій користувача зводиться до виклику методів провайдера (ліст. 2.7).

Лістинг 2.7 – Керування сеансом читання (кнопка «старт/пауза» та зміна швидкості)

```
class _Controls extends StatelessWidget {
  const _Controls();

  @override
  Widget build(BuildContext context) {
    final reading = Provider.of<ReadingProvider>(context);
    final isPlaying = reading.model?.isPlaying ?? false;

    return Column(
      children: [
        IconButton(
          icon: Icon(isPlaying ? Icons.pause : Icons.play_arrow),
          onPressed: () {
            isPlaying ? reading.pause() : reading.resume();
          },
        ),
        Slider(
```

```
            min: 100,
            max: 800,
            value: (reading.model?.wpm ?? 300).toDouble(),
            onChanged: (value) => reading.setSpeed(value.toInt()),
          ),
        ],
      );
    }
}
```

Кінець лістингу 2.7

У результаті інтерфейс для користувача залишається максимально простим: одна зона для сприйняття тексту й мінімальний набір елементів керування темпом (рис. 2.5). Вся «важка» логіка – обчислення інтервалів, зміна індексу, збереження прогресу – інкапсульована у провайдері.

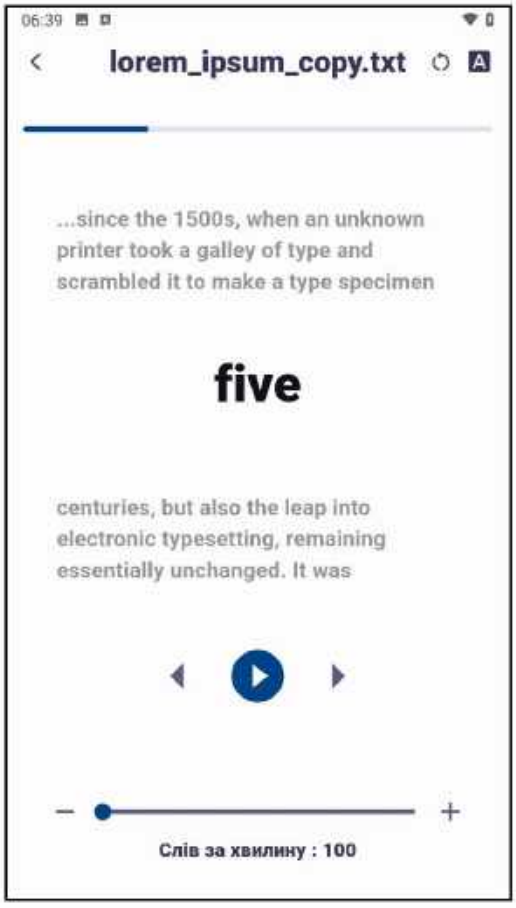

Рисунок 2.5 – Екран читання в режимі RSVP (стан паузи) у мобільному застосунку Rapid Read

Таким чином, ядро RSVP-читання в Rapid Read побудоване на зв'язці «модель стану + провайдер + реактивний інтерфейс». Провайдер відповідає за таймінги, керування поточним словом і оновлення статистики, а екран читання лише відображає актуальний стан та передає дії користувача назад у логіку. Такий підхід спрощує підтримку коду й дозволяє розширювати функціональність (наприклад, додавати паузи за пунктуацією або підсвічування оптимальної літери) без змін базової архітектури.

Бібліотека документів є однією з базових підсистем застосунку Rapid Read: саме через неї користувач додає нові тексти, обирає файл для читання та бачить загальний стан кожного документа (новий, у процесі, прочитаний). На рівні реалізації робота з файлами побудована навколо трьох основних компонентів: моделі файлу (FileModel), сервісу доступу до локального сховища (DBService) та провайдера стану бібліотеки (FileProvider), який пов'язує зберігання даних з інтерфейсом.

2.2.4 Модель файлу та зберігання в локальній БД

Для представлення текстового документа використовується окрема модель, що зберігається в локальній базі даних Hive. Модель містить базові атрибути: ідентифікатор, назву, шлях до файлу на пристрої, загальну кількість слів, поточний прогрес та дату додавання.

Лістинг 2.8 – Спрощена модель FileModel для зберігання даних про документ

```
@HiveType(typeId: 0)
class FileModel extends HiveObject {
  @HiveField(0)
  String id;

  @HiveField(1)
  String name;

  @HiveField(2)
  String path;

  @HiveField(3)
  int totalWords;

  @HiveField(4)
  int currentIndex;
```

```
  @HiveField(5)
  DateTime addedAt;

  FileModel({
    required this.id,
    required this.name,
    required this.path,
    this.totalWords = 0,
    this.currentIndex = 0,
    required this.addedAt,
  });
}
```

Кінець лістингу 2.8

Клас успадковує HiveObject, що дозволяє напряму взаємодіяти з Hive (зберігати, оновлювати, видаляти записи). Поле currentIndex використовується для фіксації прогресу читання: під час сеансу цей індекс оновлюється, а при повторному відкритті файлу застосунок може продовжити читання з потрібного місця.

Ініціалізація сховища та відкриття коробки з файлами виконується в сервісі DBService, що викликається ще на етапі старту застосунку. У спрощеному вигляді доступ до боксу з файлами виглядає так (ліст. 2.9).

Лістинг 2.9 – Фрагмент сервісу DBService для роботи з колекцією файлів

```
class DBService {
  static const String filesBoxName = 'files';

  late Box<FileModel> filesBox;

  Future<void> init() async {
    filesBox = await Hive.openBox<FileModel>(filesBoxName);
  }

  Box<FileModel> getFilesBox() => filesBox;
}
```

Кінець лістингу 2.9

Таким чином, усі дані бібліотеки зберігаються локально, без необхідності підключення до мережі, а доступ до них організовано через єдиний сервіс.

2.2.5 Провайдер бібліотеки FileProvider

Безпосередню взаємодію між локальним сховищем та інтерфейсом реалізує провайдер FileProvider. Він завантажує список документів із бази даних, надає їх у вигляді колекції для UI, а також відповідає за додавання, оновлення та видалення файлів (ліст. 2.10).

Лістинг 2.10 – Фрагмент провайдера FileProvider з завантаженням і видаленням файлів

```
class FileProvider extends ChangeNotifier {
  final DBService _dbService = DBService();
  late Box<FileModel> _filesBox;

  List<FileModel> _files = [];

  List<FileModel> get files => _files;

  Future<void> loadFiles() async {
    _filesBox = _dbService.getFilesBox();
    _files = _filesBox.values.toList();
    notifyListeners();
  }

  Future<void> deleteFile(FileModel file) async {
    await file.delete();
    _files.remove(file);
    notifyListeners();
  }
}
```

Кінець лістингу 2.10

Метод loadFiles() викликається під час ініціалізації провайдера, після чого список _files стає доступним для віджетів, які відображають бібліотеку. При будь-яких змінах (додавання, видалення, оновлення прогресу) провайдер викликає notifyListeners(), і інтерфейс автоматично оновлюється.

Додавання нового файлу відбувається у кілька кроків: користувач обирає файл через системний файловий пікер, застосунок зчитує його вміст, виконує попередній підрахунок слів, створює новий екземпляр FileModel та зберігає його в Hive (ліст. 2.11).

Лістинг 2.11 – Додавання нового документа до бібліотеки в FileProvider

```
Future<void> addFile(String path, String name, int totalWords)
async {
  final file = FileModel(
    id: const Uuid().v4(),
    name: name,
    path: path,
    totalWords: totalWords,
    addedAt: DateTime.now(),
  );

  await _filesBox.add(file);
  _files.add(file);
  notifyListeners();
}
```

Кінець лістингу 2.11

Підрахунок totalWords виконується окремим хелпером після зчитування тексту з файлу.

2.2.6 Відображення бібліотеки у користувацькому інтерфейсі

На рівні інтерфейсу бібліотека реалізована окремим екраном, який підписується на FileProvider та відображає список файлів (ліст. 2.12).

Лістинг 2.12 – Відображення списку документів на екрані бібліотеки

```
class LibraryScreen extends StatelessWidget {
  const LibraryScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Consumer<FileProvider>(
      builder: (context, filesProvider, _) {
        final files = filesProvider.files;

        return ListView.builder(
          itemCount: files.length,
          itemBuilder: (context, index) {
            final file = files[index];

            return ListTile(
              title: Text(file.name),
              subtitle: Text(
                '${file.currentIndex}/${file.totalWords} слів',
              ),
              onTap: () {
                // Open file in reading mode
```

```
                },
                trailing: IconButton(
                  icon: const Icon(Icons.delete),
                  onPressed: () => filesProvider.deleteFile(file),
                ),
              );
            },
          );
        },
      );
    }
}
```

Кінець лістингу 2.12

Кожен елемент списку показує назву документа, прогрес читання та, за потреби, додаткову інформацію (наприклад, відсоток прочитаного) (рис. 2.6).

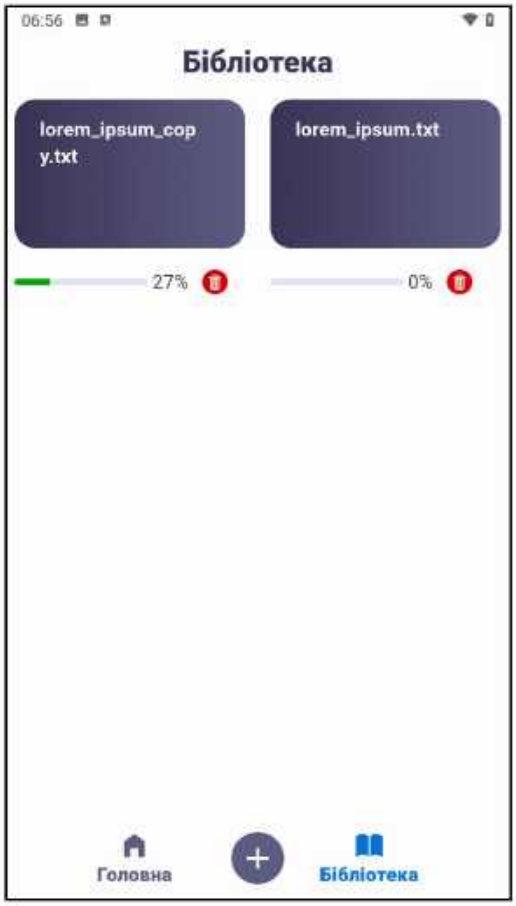

Рисунок 2.6 – Екран бібліотеки документів у мобільному застосунку Rapid Read

Коли користувач натискає на певний документ, система відкриває екран читання й передає до ReadingProvider інформацію про вибраний файл та поточний прогрес. При натисканні на кнопку видалення запис видаляється з Hive, а провайдер оновлює список у пам'яті.

Таким чином, підсистема роботи з файлами забезпечує повний цикл роботи з текстами: імпорт, збереження метаданих, відображення бібліотеки, оновлення прогресу та видалення документів. Локальне зберігання на основі Hive та використання провайдера FileProvider дозволяють реалізувати цю логіку компактно та забезпечити реактивне оновлення інтерфейсу без зайвого дублювання коду.

Окремий блок функціональності в Rapid Read відповідає за персоналізацію інтерфейсу: вибір теми (світла/темна), параметрів шрифту в зоні читання та мови інтерфейсу. Ці налаштування безпосередньо впливають на комфорт читання, особливо на мобільних пристроях, де умови освітлення, розмір екрана та індивідуальні особливості зору користувача відіграють важливу роль. На рівні реалізації налаштування винесені в окремі провайдери (ThemeProvider, FontProvider, LocaleProvider) і зберігаються у локальному сховищі, щоб застосовуватися автоматично при наступних запусках застосунку.

2.2.7 Керування темою оформлення (ThemeProvider)

Вибір теми реалізовано через окремий провайдер, який зберігає поточний режим (світлий або темний) і надає метод для його перемикання. Значення теми зберігається у локальному сховищі (наприклад, через SharedPreferences або окремий Hive-box), а при старті застосунку – відновлюється (ліст. 2.13).

Лістинг 2.13 – Спрощений провайдер ThemeProvider для керування темою оформлення

```
class ThemeProvider extends ChangeNotifier {
  bool _isDarkMode = false;

  bool get isDarkMode => _isDarkMode;

  Future<void> loadTheme() async {
    // Load theme from local storage
```

```
    final stored = await _loadFromStorage();
    _isDarkMode = stored ?? false;
    notifyListeners();
  }

  Future<void> toggleTheme() async {
    _isDarkMode = !_isDarkMode;
    await _saveToStorage(_isDarkMode);
    notifyListeners();
  }

  Future<bool?> _loadFromStorage() async {
    // Implementation of reading persisted theme value
  }

  Future<void> _saveToStorage(bool value) async {
    // Implementation of saving theme value
  }
}
```

Кінець лістингу 2.13

У кореневому MaterialApp значення isDarkMode використовується для вибору ThemeMode, а самі теми визначені у класі DefaultTheme. Такий підхід дозволяє змінювати тему «на льоту»: після виклику toggleTheme() всі віджети, що залежать від теми, автоматично оновлюються.

2.2.8 Налаштування шрифту для зони читання (FontProvider)

Для швидкісного читання важливими є не лише розмір шрифту, а й його тип – від цього залежить читабельність тексту на різних екранах. У Rapid Read параметри шрифту зібрані в окремому провайдері, який зберігає обраний розмір та, за потреби, конкретне сімейство шрифту (ліст. 2.14).

Лістинг 2.14 – Провайдер FontProvider для збереження та зміни розміру шрифту

```
class FontProvider extends ChangeNotifier {
  double _fontSize = 32.0;

  double get fontSize => _fontSize;

  Future<void> loadFont() async {
    // Load font size from local storage
    final stored = await _loadFontSize();
    if (stored != null) {
      _fontSize = stored;
      notifyListeners();
```

```
    }
  }

  Future<void> setFontSize(double size) async {
    _fontSize = size;
    await _saveFontSize(size);
    notifyListeners();
  }

  Future<double?> _loadFontSize() async {
    // Implementation of reading font size
  }

  Future<void> _saveFontSize(double size) async {
    // Implementation of saving font size
  }
}
```

Кінець лістингу 2.14

На екрані читання цей провайдер використовується для формування стилю тексту. Наприклад, замість фіксованого displayMedium можна застосувати розмір, що прийшов із FontProvider (ліст. 2.15).

Лістинг 2.15 – Використання розміру шрифту з FontProvider на екрані читання

```
final fontProvider = Provider.of<FontProvider>(context);

Text(
  word,
  textAlign: TextAlign.center,
  style: Theme.of(context).textTheme.displayMedium?.copyWith(
        fontSize: fontProvider.fontSize,
      ),
);
```

Кінець лістингу 2.15

Таким чином, користувач може збільшити або зменшити розмір шрифту в налаштуваннях, і ці зміни будуть застосовані до зони читання без перезапуску застосунку.

2.2.9 Локалізація інтерфейсу (LocaleProvider)

Застосунок підтримує кілька мов інтерфейсу (зокрема, українську та англійську), що реалізовано на основі стандартного механізму локалізації Flutter

та згенерованого класу S. Вибір мови винесено в окремий провайдер, який зберігає поточну локаль і дозволяє змінювати її в налаштуваннях (ліст. 2.16).

Лістинг 2.16 – Провайдер LocaleProvider для вибору мови інтерфейсу

```
class LocaleProvider extends ChangeNotifier {
  Locale _locale = const Locale('uk');

  Locale get locale => _locale;

  Future<void> loadSavedLocale() async {
    final code = await _loadFromStorage();
    if (code != null) {
      _locale = Locale(code);
      notifyListeners();
    }
  }

  Future<void> setLocale(Locale locale) async {
    _locale = locale;
    await _saveToStorage(locale.languageCode);
    notifyListeners();
  }

  Future<String?> _loadFromStorage() async {
    // Implementation of reading saved language code
  }

  Future<void> _saveToStorage(String code) async {
    // Implementation of saving language code
  }
}
```

Кінець лістингу 2.16

У класі MyApp значення localeProvider.locale передається в параметр locale MaterialApp, а supportedLocales та localizationsDelegates визначені на основі згенерованого делегата S. Це забезпечує автоматичне оновлення текстів інтерфейсу при зміні мови.

2.2.10 Екран налаштувань та взаємодія з користувачем

Усі описані параметри зібрані на окремому екрані налаштувань, де користувач може змінити тему, шрифт, мову інтерфейсу, а також (у межах цього ж розділу) налаштувати PIN-код доступу. Екран підписується на відповідні провайдери і через елементи керування змінює їхній стан (рис. 2.7).

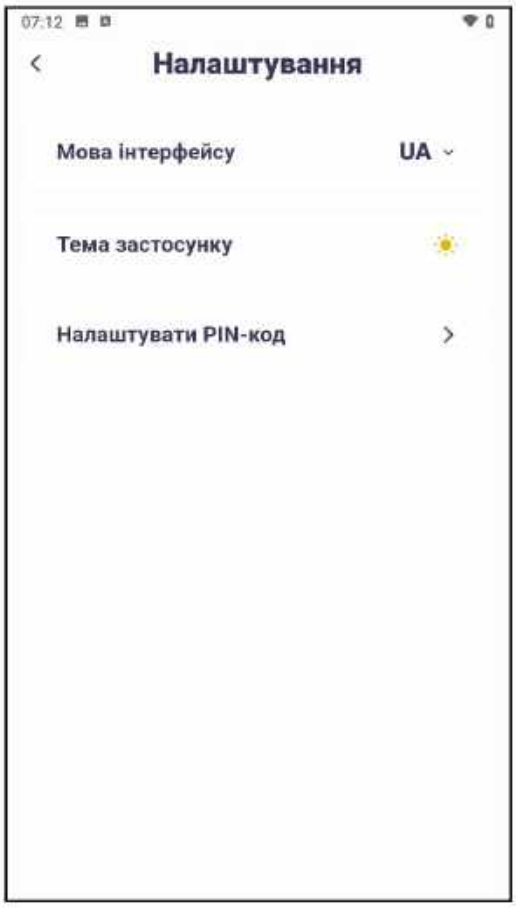

Рисунок 2.7 – Екран налаштувань мобільного застосунку Rapid Read (тема, шрифт, мова інтерфейсу)

У результаті налаштування теми, шрифту та мови реалізовані як окремі, слабо зв'язані модулі, які зберігають свій стан у локальному сховищі й надають його через провайдери всьому застосунку. Це дозволяє легко адаптувати Rapid Read під індивідуальні потреби користувача та забезпечує стабільну поведінку налаштувань при повторних запусках програми.

У мобільному застосунку Rapid Read підтримується простий, але зручний механізм захисту доступу: користувач може встановити PIN-код, яким буде захищено вхід до програми та доступ до його особистої бібліотеки й статистики читання. Такий підхід особливо актуальний, якщо застосунок використовується на спільному пристрої або зберігає особисті матеріали (нотатки, навчальні тексти тощо).

Архітектурно PIN-функціональність складається з трьох рівнів:

– сховище налаштувань у DBService, де зберігаються сам PIN-код та прапорці стану (чи встановлений PIN, чи був він пропущений тощо);

– маршрутизація після splash-екрана, яка вирішує, чи показувати екран введення PIN, чи відразу переходити до головного екрана;

– екран роботи з PIN-кодом, де користувач може створити, ввести, змінити або видалити PIN.

2.2.11 Перевірка PIN-коду під час запуску застосунку

Одразу після splash-екрана застосунок повинен вирішити, що показати користувачу: екран введення PIN-коду чи головний екран бібліотеки. Для цього використовується окремий віджет SplashRouter, який звертається до DBService і перевіряє, чи є встановлений PIN та чи не було його свідомо пропущено (ліст. 2.17).

Лістинг 2.17 – Фрагмент SplashRouter з перевіркою наявності PIN-коду

```
class SplashRouter extends StatelessWidget {
  const SplashRouter({super.key});

  Future<bool> _checkPin() async {
    final db = DBService();
    return db.isPinSet() && !db.isPinSkipped();
  }

  @override
  Widget build(BuildContext context) {
    return FutureBuilder<bool>(
      future: _checkPin(),
      builder: (context, snapshot) {
        final hasPin = snapshot.data ?? false;

        if (!hasPin) return const HomeScreen();

        return PinScreen(
          isFirstTime: !hasPin,
           onSuccess: () => Navigator.pushReplacementNamed(context,
'/home'),
          isFromDrawer: false,
        );
      },
    );
  }
}
```

Кінець лістингу 2.17

Тут використовується два важливі прапорці, що зберігаються в локальному сховищі:

– isPinSet() – чи збережено PIN у налаштуваннях;

– isPinSkipped() – чи вирішив користувач тимчасово пропустити налаштування PIN.

Якщо PIN не встановлено або користувач його свідомо пропустив, застосунок відразу відкриває головний екран. Якщо ж PIN є і його не пропущено – показується екран PinScreen, на якому користувач повинен ввести код.

2.2.12 Зберігання та перевірка PIN-коду в DBService

Логіка зберігання PIN-коду зосереджена в сервісі DBService, який працює зі спеціальним Hive-box налаштувань. У ньому зберігаються сам PIN та допоміжні прапорці (ліст. 2.18).

Лістинг 2.18 – Методи роботи з PIN-кодом у сервісі DBService

```
class DBService {
  static const String _pinCodeKey = 'pinCode';
  static const String _isPinSkippedKey = 'isPinSkipped';

  bool isPinSet() {
    try {
      return _settingsBox.containsKey(_pinCodeKey);
    } catch (e) {
      return false;
    }
  }

  Future<void> savePinCode(String pin) async {
    await _settingsBox.put(_pinCodeKey, pin);
  }

  String? getPinCode() {
    try {
      return _settingsBox.get(_pinCodeKey);
    } catch (e) {
      return null;
    }
  }

  bool verifyPin(String enteredPin) {
    try {
```

```
      final savedPin = getPinCode();
      return savedPin == enteredPin;
    } catch (e) {
      return false;
    }
  }

  Future<void> deletePinCode() async {
    await _settingsBox.delete(_pinCodeKey);
  }

  Future<void> setPinSkipped(bool skipped) async {
    await _settingsBox.put(_isPinSkippedKey, skipped);
  }

  bool isPinSkipped() {
    try {
          return _settingsBox.get(_isPinSkippedKey, defaultValue:
false);
    } catch (e) {
      return false;
    }
  }
}
```

Кінець лістингу 2.18

У такому вигляді DBService відповідає за всі базові операції:

– збереження нового PIN (savePinCode);

– перевірка правильності введеного коду (verifyPin);

– видалення PIN (deletePinCode);

– встановлення/перевірка прапорця пропуску PIN (setPinSkipped / isPinSkipped).

Це дозволяє не дублювати логіку в UI-компонентах та централізовано керувати станом безпеки застосунку.

2.2.13 Екран введення та керування PIN-кодом

Екран введення PIN-коду реалізований у вигляді комбінації PinScreen та PinBody. PinScreen відповідає лише за обгортку Scaffold, тоді як основна логіка міститься у стані _PinBodyState. Тут обробляється введення коду, вибір режиму (створення нового PIN чи перевірка наявного), а також зміна та видалення PIN.

Нижче наведено ключовий фрагмент валідації PIN-коду (ліст. 2.19).

Лістинг 2.19 – Логіка створення та перевірки PIN-коду у стані PinBody

```
class _PinBodyState extends State<PinBody> {
  final _pinController = TextEditingController();
  final DBService db = DBService();

  String? _errorText;

  void _validatePin() {
    final pin = _pinController.text.trim();
    if (pin.length != 4) {
      setState(() => _errorText = 'PIN must be 4 digits');
      return;
    }

    if (widget.isFirstTime || !db.isPinSet()) {
      db.savePinCode(pin);
      widget.onSuccess();
    } else {
      final isValid = db.verifyPin(pin);
      if (isValid) {
        widget.onSuccess();
      } else {
        setState(() => _errorText = 'Invalid PIN');
      }
    }
  }
}
```

Кінець лістингу 2.19

Алгоритм роботи простий:

– якщо користувач вперше налаштовує PIN (або він ще не встановлений), введене значення зберігається через savePinCode, після чого викликається onSuccess() і користувача перекидає на головний екран;

– якщо PIN уже існує, застосунок порівнює введене значення з тим, що збережено в базі (verifyPin); у разі успіху викликається onSuccess(), у разі помилки – виводиться повідомлення про неправильний код.

У самому інтерфейсі екран доповнений (рис. 2.8):

– індикаторами введення (крапки, що показують кількість введених цифр);

– цифровою клавіатурою (PinPad) для введення PIN;

– кнопками «Skip» та «Change PIN» (через SkipAndChangeButtons), які дозволяють пропустити налаштування чи викликати діалог зміни коду;

– іконкою видалення PIN (через DeletePinDialog), яка доступна, якщо код уже встановлений.

Рисунок 2.8 – Екран введення PIN-коду в мобільному застосунку Rapid Read

У підсумку механізм PIN-коду в Rapid Read реалізовано як легку, локальну систему захисту доступу: усі дані зберігаються в налаштуваннях через DBService, а логіка взаємодії винесена в окремий екран із чітким поділом на створення, перевірку, зміну та видалення PIN. Це підвищує приватність роботи із застосунком без ускладнення загального користувацького сценарію.

Щоб застосунок швидкісного читання був корисним не лише «тут і зараз», а й у довгостроковій перспективі, він має вміти запам’ятовувати, де

користувач зупинився, та показувати загальну динаміку читання. У Rapid Read це реалізовано через два взаємопов'язані механізми:

– прогрес читання для кожного файлу (індекс останнього прочитаного слова);

– агреговану статистику (загальна кількість прочитаних слів, сумарний час читання тощо).

На практиці за це відповідають ReadingProvider (який оновлює прогрес та «сирі» показники) та окремий StatsProvider, що зберігає і надає зведені дані для екрану статистики.

2.2.14 Оновлення прогресу під час сеансу читання

Під час роботи в режимі RSVP застосунок повинен не тільки показувати наступні слова, а й фіксувати, на якому слові користувач зупинився. У спрощеному вигляді це реалізовано в ReadingProvider: після кожного кроку (або при завершенні/паузі) провайдер оновлює поле currentIndex відповідного FileModel та зберігає зміни в локальній базі даних (ліст. 2.20).

Лістинг 2.20 – Оновлення прогресу читання файлу в ReadingProvider

```
class ReadingProvider extends ChangeNotifier {
  final DBService _db = DBService();
  FileModel? _currentFile;
  ReadingModel? _model;
  Timer? _timer;

    void startReading(FileModel file, List<String> words, int
startIndex) {
    _currentFile = file;
    _model = ReadingModel(
      words: words,
      currentIndex: startIndex,
      wpm: file.defaultWpm ?? 300,
    );
    _model!.isPlaying = true;
    _startTimer();
    notifyListeners();
  }

  void _nextWord() {
    if (_model == null || !_model!.isPlaying) return;

    if (_model!.currentIndex < _model!.words.length - 1) {
```

```
      _model!.currentIndex++;
      _updateFileProgress();
    } else {
      _finishSession();
    }

    notifyListeners();
  }

  Future<void> _updateFileProgress() async {
    if (_currentFile == null || _model == null) return;

    _currentFile!.currentIndex = _model!.currentIndex;
    await _db.updateFile(_currentFile!);
  }
}
```

Кінець лістингу 2.20

Метод _updateFileProgress() викликається при зміні індексу слова. Завдяки цьому в FileModel завжди зберігається актуальна позиція, і при повторному відкритті документа Rapid Read може продовжити читання з того місця, де користувач зупинився раніше. Логіка оновлення може виконуватися не на кожному слові (щоб не перевантажувати сховище), а, наприклад, з певним інтервалом, однак загальний принцип залишається тим самим.

2.2.15 Агрегована статистика у StatsProvider

Для відстеження загальної активності використовується окремий провайдер StatsProvider. Він накопичує базові показники: скільки слів загалом прочитано, скільки часу користувач провів у режимі читання, а також, за потреби, проміжні значення (наприклад, статистику за день або за файл) (ліст. 2.21).

Лістинг 2.21 – Провайдер StatsProvider для збереження загальної статистики читання

```
class StatsProvider extends ChangeNotifier {
  final DBService _db = DBService();

  int _totalWords = 0;
  Duration _totalReadingTime = Duration.zero;
```

```
  int get totalWords => _totalWords;
  Duration get totalReadingTime => _totalReadingTime;

  Future<void> loadStats() async {
    final stats = await _db.loadStats();
    _totalWords = stats.totalWords;
    _totalReadingTime = stats.totalReadingTime;
    notifyListeners();
  }

  Future<void> addSessionStats(int words, Duration duration) async
{
    _totalWords += words;
    _totalReadingTime += duration;

    await _db.saveStats(_totalWords, _totalReadingTime);
    notifyListeners();
  }
}
```

Кінець лістингу 2.21

Метод loadStats() викликається при старті застосунку або при відкритті екрану статистики. Метод addSessionStats() використовується для оновлення показників після завершення кожного сеансу.

2.2.16 Передача статистики з ReadingProvider до StatsProvider

Щоб підрахунок статистики був узгоджений із процесом читання, ReadingProvider фіксує початок сеансу (час старту та початковий індекс слова), а при завершенні передає в StatsProvider кількість прочитаних слів та тривалість сеансу (ліст. 2.22).

Лістинг 2.22 – Оновлення статистики сеансу після завершення читання

```
class ReadingProvider extends ChangeNotifier {
  final StatsProvider _statsProvider;
  DateTime? _sessionStart;
  int _startIndex = 0;

  ReadingProvider(this._statsProvider);

    void startReading(FileModel file, List<String> words, int
startIndex) {
    _startIndex = startIndex;
    _sessionStart = DateTime.now();
    // ... ініціалізація _model та запуск таймера
```

```
  }

  Future<void> _finishSession() async {
    _timer?.cancel();
    _model?.isPlaying = false;

    if (_sessionStart != null && _model != null) {
      final duration = DateTime.now().difference(_sessionStart!);
      final wordsRead = _model!.currentIndex - _startIndex + 1;

      await _statsProvider.addSessionStats(wordsRead, duration);
    }

    notifyListeners();
  }
}
```

Кінець лістингу 2.22

Такий підхід дозволяє розділити відповідальність: ReadingProvider знає про деталі конкретного сеансу, а StatsProvider відповідає за довгострокове накопичення даних.

2.2.17 Зберігання статистики в локальному сховищі

Як і інші дані, статистика зберігається локально за допомогою DBService. Залежно від складності моделі це може бути окремий Hive-box або запис із ключем у таблиці налаштувань. У спрощеному варіанті використовується окремий клас StatsModel (ліст. 2.23).

Лістинг 2.23 – Модель StatsModel для зберігання сукупної статистики

```
@HiveType(typeId: 1)
class StatsModel extends HiveObject {
  @HiveField(0)
  int totalWords;

  @HiveField(1)
  int totalReadingSeconds;

  StatsModel({
    required this.totalWords,
    required this.totalReadingSeconds,
  });
}
```

Кінець лістингу 2.23

Відповідні методи в DBService будуть виглядати так (ліст. 2.24).

Лістинг 2.24 – Методи читання та збереження статистики в DBService

```
class DBService {
  static const String statsBoxName = 'stats';

  late Box<StatsModel> _statsBox;

  Future<void> initStatsBox() async {
    _statsBox = await Hive.openBox<StatsModel>(statsBoxName);
  }

  Future<StatsModel> loadStats() async {
    if (_statsBox.isEmpty) {
       final stats = StatsModel(totalWords: 0, totalReadingSeconds:
0);
      await _statsBox.add(stats);
      return stats;
    }
    return _statsBox.values.first;
  }

  Future<void> saveStats(int totalWords, Duration totalTime) async
{
    final stats = _statsBox.values.first;
    stats
      ..totalWords = totalWords
      ..totalReadingSeconds = totalTime.inSeconds;
    await stats.save();
  }
}
```

Кінець лістингу 2.24

Це забезпечує збереження статистики між запусками застосунку та дає можливість поступово накопичувати дані.

2.2.18 Відображення статистики в інтерфейсі

Для користувача статистика подається у зручному, стиснутому вигляді: загальна кількість прочитаних слів, сумарний час читання, можливо – середня швидкість за сеанси чи простий розподіл по днях (рис. 2.4). На окремому екрані StatsScreen провайдер StatsProvider підключається через Consumer і передає актуальні значення у віджети (ліст. 2.25).

Лістинг 2.25 – Екран статистики читання з використанням StatsProvider

```
class StatsScreen extends StatelessWidget {
  const StatsScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Consumer<StatsProvider>(
      builder: (context, stats, _) {
        final minutes = stats.totalReadingTime.inMinutes;
        final seconds = stats.totalReadingTime.inSeconds % 60;

        return Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              'Загалом прочитано слів: ${stats.totalWords}',
              style: Theme.of(context).textTheme.titleMedium,
            ),
            const SizedBox(height: 8),
            Text(
              'Сумарний час читання: $minutes хв $seconds с',
              style: Theme.of(context).textTheme.titleMedium,
            ),
              // За потреби: тут можна додати просту діаграму або
список сеансів
          ],
        );
      },
    );
  }
}
```

Кінець лістингу 2.25

У підсумку механізм прогресу та статистики в Rapid Read забезпечує:

– персоналізацію (повернення до того самого місця в тексті);

– мотиваційний ефект (користувач бачить власний прогрес);

– основу для аналізу (можна оцінювати ефективність тренувань у швидкісному читанні).

Усе це реалізовано локально, без зовнішнього сервера, за рахунок поєднання ReadingProvider, StatsProvider та сервісу DBService, що спрощує супровід і робить застосунок автономним.

Після реалізації основних модулів мобільного застосунку Rapid Read було проведено серію перевірок, спрямованих не лише на виявлення помилок, а й на

перевірку того, наскільки система відповідає вимогам, сформульованим вище. Оскільки застосунок орієнтований на реальних користувачів і щоденне використання, основний акцент зроблено на функціональному тестуванні, перевірці стабільності RSVP-режиму, збереженні даних та зручності роботи інтерфейсу.

2.2.19 Функціональне тестування основних сценаріїв

Перший етап тестування був зосереджений на перевірці ключових користувацьких сценаріїв, які складають «кістяк» роботи застосунку. Тестування проводилося вручну, за підготовленими сценаріями, на реальному пристрої Android та в емуляторі. Основні перевірені сценарії узагальнені в таблиці 2.1.

Таблиця 2.1 – Основні сценарії тестування

Сценарій	Очікуваний результат
Імпорт нового .txt-файлу з пам'яті пристрою	Файл з'являється в бібліотеці з правильною назвою та підрахованою кількістю слів
Відкриття файлу з бібліотеки й запуск сеансу читання	Відкривається екран читання, з'являється перше слово, таймер запускається
Пауза та продовження сеансу	Після «Пауза» слова перестають змінюватися; після «Старт» читання триває з того ж слова
Зміна швидкості (WPM) під час активного читання	Інтервал між словами змінюється, читання залишається плавним
Вихід із режиму читання та повторне відкриття того ж файлу	Читання продовжується з того слова, на якому користувач зупинився
Видалення файлу з бібліотеки	Файл зникає зі списку, спроба його відкрити більше неможлива
Перемикання теми (світла/темна) у налаштуваннях	Тема інтерфейсу змінюється без перезапуску, у всіх основних екранах
Зміна розміру шрифту в зоні читання	Розмір тексту на екрані читання змінюється відповідно до обраного значення
Зміна мови інтерфейсу	Тексти кнопок, заголовків та повідомлень відображаються обраною мовою
Встановлення PIN-коду та повторний запуск застосунку	Після перезапуску показується екран введення PIN, вхід без коду неможливий

Продовження таблиці 2.1

Сценарій	Очікуваний результат
Пропуск налаштування PIN-коду (Skip)	Застосунок відкривається без PIN, повторні запуски також не вимагають коду
Перегляд статистики після кількох сеансів	Відображаються оновлені значення загальної кількості прочитаних слів та часу

За результатами цього етапу всі основні сценарії працюють відповідно до очікувань: користувач може додавати файли, читати їх у режимі RSVP, повернутися до потрібного місця, змінювати оформлення та мову, а також переглядати базову статистику.

2.2.20 Перевірка стабільності та плавності RSVP-режиму

Окрему увагу приділено саме режиму швидкісного читання, оскільки він є головною особливістю Rapid Read. Під час тестування перевірялися:

– плавність оновлення слів при різних значеннях швидкості (від тренувальних до максимальних, передбачених застосунком);

– відсутність «ривків» та випадкових пауз у зміні слів;

– час реакції на натискання кнопок «Пауза», «Старт», зміну швидкості та перемикання між словами.

Практичні спостереження показали, що на реальному пристрої при стандартних швидкостях читання інтерфейс оновлюється рівномірно, зміна слів сприймається як стабільний потік, а натискання кнопок обробляються без помітних затримок. Підвищення швидкості наближає читання до природних меж сприйняття користувача, але з технічної точки зору анімація залишається плавною.

Також перевірялися сценарії:

– постановка на паузу й миттєве продовження;

– швидка зміна швидкості під час активного сеансу;

– довші сесії (по кілька хвилин без зупинки).

У цих сценаріях читання зберігало стабільність, без збоїв таймера чи некоректних переходів між словами.

2.2.21 Перевірка збереження даних та поведінки при перезапуску

Для застосунку, який працює локально і не використовує сервер, важливо, щоб усі ключові дані коректно зберігалися в пам'яті пристрою. Тому окремий блок тестування був присвячений:

– прогресу читання (вихід з режиму читання та повторне відкриття того ж файлу; перезапуск застосунку (повне закриття та повторний запуск));

– налаштуванням (зміна теми, мови, розміру шрифту, встановлення/зняття PIN-коду; перевірка, чи зберігаються ці параметри після перезапуску);

– статистиці (виконання кількох сеансів читання з різною тривалістю; перевірка, чи коректно оновлюється загальна кількість прочитаних слів та сумарний час).

У результаті тестування підтверджено, що:

– індекс останнього прочитаного слова для кожного файлу зберігається й використовується при повторному відкритті;

– вибрані налаштування інтерфейсу (тема, шрифт, мова) зберігаються та відновлюються при наступних запусках;

– статистика читання накопичується коректно, без «обнулення» при виході із застосунку.

2.2.22 Перевірка PIN-коду та сценаріїв доступу

Оскільки PIN-код впливає на вхід до застосунку, для нього окремо перевірялися:

– встановлення PIN при першому налаштуванні;

– успішний вхід із правильним PIN та відмова при неправильному;

– можливість пропустити налаштування PIN та подальша робота без нього;

– зміна існуючого PIN на новий;

– видалення PIN-коду й повернення до «відкритого» режиму доступу.

У всіх сценаріях логіка поводилася передбачувано: при правильному PIN доступ надавався, при неправильному – відображалося повідомлення про

помилку, а дані бібліотеки при цьому залишались недоступними. Після видалення PIN-коду екран його введення більше не показувався.

Проведене тестування показало, що мобільний застосунок Rapid Read:

– відповідає основним функціональним вимогам, сформульованим на етапі постановки задачі (імпорт файлів, режим RSVP, налаштування, статистика, PIN);

– забезпечує стабільний режим читання без помітних технічних затримок та збоїв навіть при зміні швидкості «на ходу»;

– коректно зберігає прогрес та налаштування, що робить його зручним для щоденного використання;

– має передбачувану поведінку в основних та граничних сценаріях (перезапуск, видалення файлів, неправильний PIN).

Отримані результати дозволяють зробити висновок, що реалізований програмний продукт досягає поставленої мети: забезпечує комфортне швидкісне читання текстів у режимі RSVP на мобільних пристроях і може слугувати основою для подальшого розширення функціональності та проведення експериментальних досліджень у наступних розділах роботи.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ШВИДКІСНОГО ЧИТАННЯ НА ОСНОВІ RSVP

3.1 Методика проведення дослідження

Мета дослідження – перевірити, як використання мобільного застосунку Rapid Read з методом RSVP впливає на швидкість і розуміння тексту, а також оцінити зручність самого застосунку як інструменту для читання.

Планується простий експеримент із повторними вимірюваннями: кожен учасник читає тексти у двох режимах – традиційному (звичайний скрол тексту на смартфоні) та в режимі RSVP через Rapid Read. Такий дизайн дозволяє порівнювати режими в межах однієї й тієї ж людини, зменшуючи вплив індивідуальних відмінностей.

У дослідженні беруть участь 15-25 респондентів, які регулярно читають з екрана смартфона. Спеціальної професійної підготовки не потрібно, важливо лише, щоб українська була для них звичною мовою читання, а зір – у межах норми (з урахуванням корекції). Усі тести проводяться в спокійному приміщенні з постійним освітленням, на подібних за характеристиками смартфонах, щоб мінімізувати вплив зовнішніх факторів.

Матеріалами дослідження є кілька текстів нон-фікшн середньої складності (близько 800-1000 слів), підготовлені запитання на розуміння до кожного тексту та короткі анкети. Для традиційного режиму читання використовується стандартний перегляд тексту на смартфоні, для швидкісного – застосунок Rapid Read. Після кожного прочитаного тексту учасник проходить тест на розуміння, а в кінці – заповнює анкету із загальною оцінкою зручності та комфортності застосунку.

Процедура для одного учасника виглядає так: короткий вступ та інструктаж, читання першого тексту у вибраному режимі, тест на розуміння, невелика перерва, читання другого тексту в іншому режимі, знову тест, а

потім – анкета щодо вражень від Rapid Read. Перед основною частиною проводиться коротка демонстрація застосунку та тренувальна сесія, щоб учасник звик до формату «слово по слову» і налаштував комфортну швидкість.

У ході експерименту фіксуються три групи показників:

– швидкість читання (час, кількість слів, розрахунок слів за хвилину);

– розуміння тексту (відсоток правильних відповідей на запитання);

– суб'єктивні оцінки (зручність інтерфейсу, комфорт читання, готовність користуватися застосунком надалі).

Таке поєднання об'єктивних метрик і опитувань відповідає сучасним підходам до юзабіліті-тестування мобільних застосунків, де аналізують і продуктивність, і сприйняття інтерфейсу користувачами.

Подальша обробка даних будується на описовій статистиці: порівнюються середні (або медіанні) значення швидкості та розуміння для традиційного читання та RSVP, аналізуються анкети й типові коментарі. Таке дослідження має пілотний характер: його завдання – не дати «остаточну істину» про всі можливі сценарії, а показати, чи є підхід із Rapid Read практично корисним, де він дає найбільший виграш і які моменти в роботі застосунку потребують подальшого вдосконалення.

3.2 Обробка та аналіз отриманих результатів

У пілотному дослідженні взяли участь 20 респондентів, які виконували завдання читання у двох режимах: традиційному (звичайний перегляд тексту на смартфоні зі скролом) та в режимі швидкісного подання тексту через мобільний застосунок Rapid Read (RSVP). Для кожного учасника фіксувалися показники швидкості читання, рівень розуміння тексту та суб'єктивна оцінка зручності використання застосунку.

За результатами вимірювань середня швидкість традиційного читання склала близько 230 слів/хв, тоді як у режимі RSVP – приблизно 360 слів/хв. Тобто в середньому учасники читали на ≈ 55-60 % швидше, коли

використовували Rapid Read. При цьому в індивідуальних випадках приріст швидкості коливався від +80 до +200 слів/хв: у частини респондентів, які й до цього читали досить швидко, ефект був помірним, тоді як для повільніших читачів перехід на RSVP давав відчутно більший приріст.

Для наочності доцільно подати порівняння середніх значень у вигляді стовпчикового графіка (рис. 3.1), де по осі абсцис відображено два режими («традиційний», «RSVP»), а по осі ординат – середня швидкість читання в словах за хвилину.

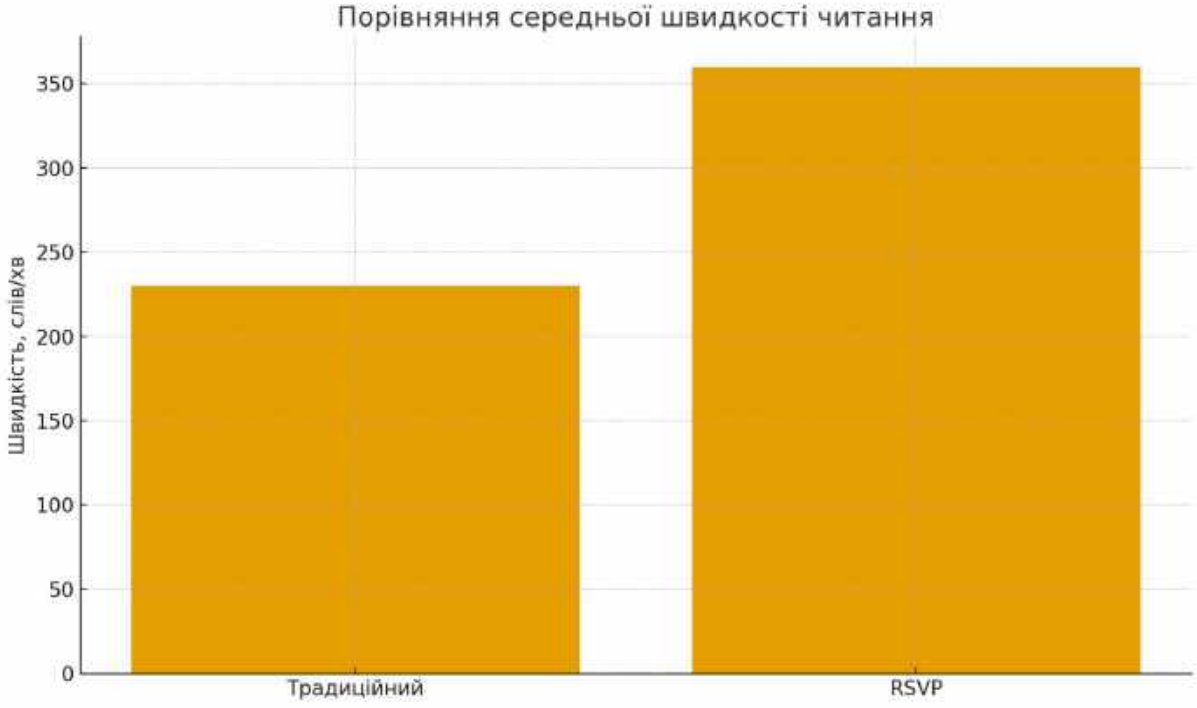

Рисунок 3.1 – Порівняння середньої швидкості читання в традиційному режимі та в режимі RSVP

Розуміння тексту оцінювалося за відсотком правильних відповідей у тестах після кожного прочитаного фрагмента. У традиційному режимі середній результат склав близько 82 % правильних відповідей, у режимі RSVP – 78 %. Тобто спостерігається невелике, але помітне зниження показника розуміння при переході на швидкісну подачу тексту.

Детальніший аналіз показав, що для помірних швидкостей у RSVP (до 300-350 слів/хв) рівень розуміння майже не відрізнявся від традиційного читання (різниця в межах 2-3 відсоткових пунктів). Більш відчутне падіння показників спостерігалося тоді, коли учасники свідомо підвищували швидкість до 400 слів/хв і вище: у цій групі середній результат опускався до 70-72 %. Це узгоджується з теоретичними уявленнями про компроміс між швидкістю та глибиною опрацювання тексту.

Порівняння середніх відсотків правильних відповідей зручно подати у вигляді ще одного стовпчикового графіка (рис. 3.2), де буде видно, що різниця між режимами помірна, але не нульова.

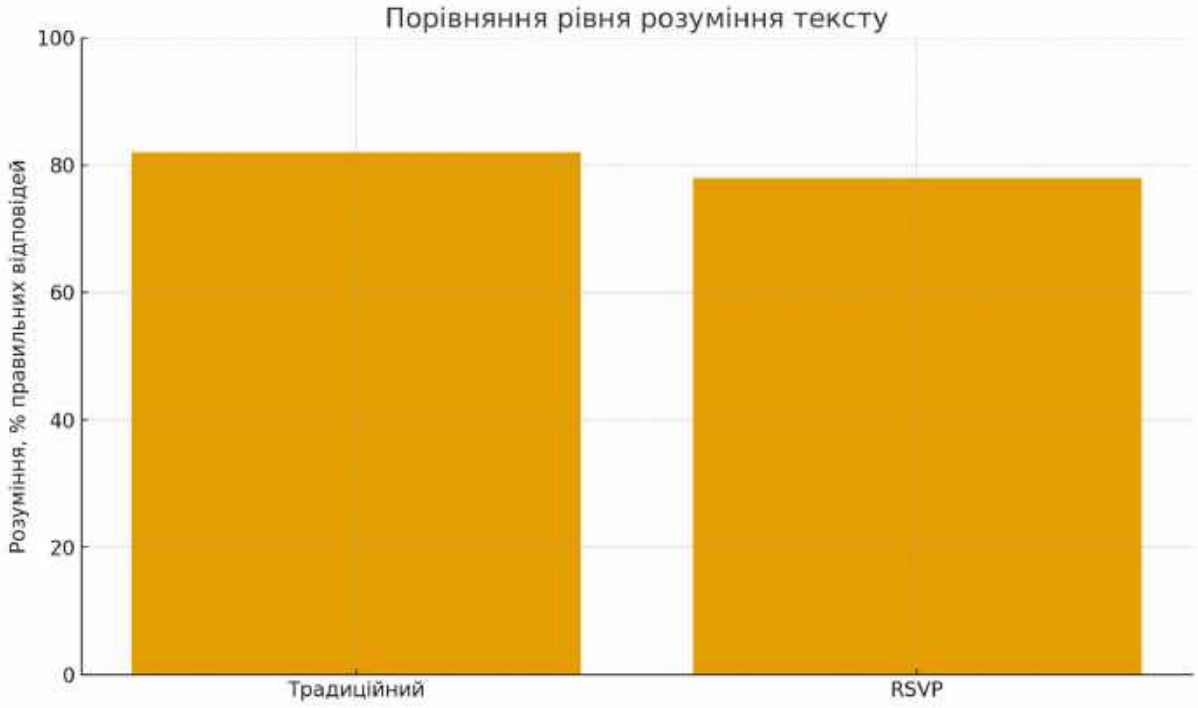

Рисунок 3.2 – Порівняння середнього рівня розуміння тексту при традиційному читанні та в режимі RSVP

Після проходження обох режимів учасники оцінювали зручність Rapid Read за кількома критеріями за п’ятибальною шкалою. У середньому було отримано такі значення:

– зрозумілість інтерфейсу – 4,4 / 5;

– комфортність читання (навантаження на очі, зручність сприйняття) – 4,0 / 5;

– зосередженість під час читання в RSVP – 4,2 / 5;

– загальна задоволеність застосунком – 3,9 / 5.

Більшість респондентів відзначили, що формат покадрової подачі тексту допомагає легше утримувати увагу й менше відволікатися на сторонні елементи інтерфейсу. Водночас декілька учасників вказали на типовий недолік RSVP – складність швидко повернутися до попереднього фрагмента, якщо щось було пропущено або неправильно зрозумілося. Також окремі коментарі стосувалися бажання мати «плавніший» контроль швидкості та можливість швидкого перемикання між кількома пресетами темпу.

Середні значення оцінок зручно зобразити на комбінованому графіку (рис. 3.3) у вигляді стовпчикової діаграми з підписаними шкалами для кожного з критеріїв.

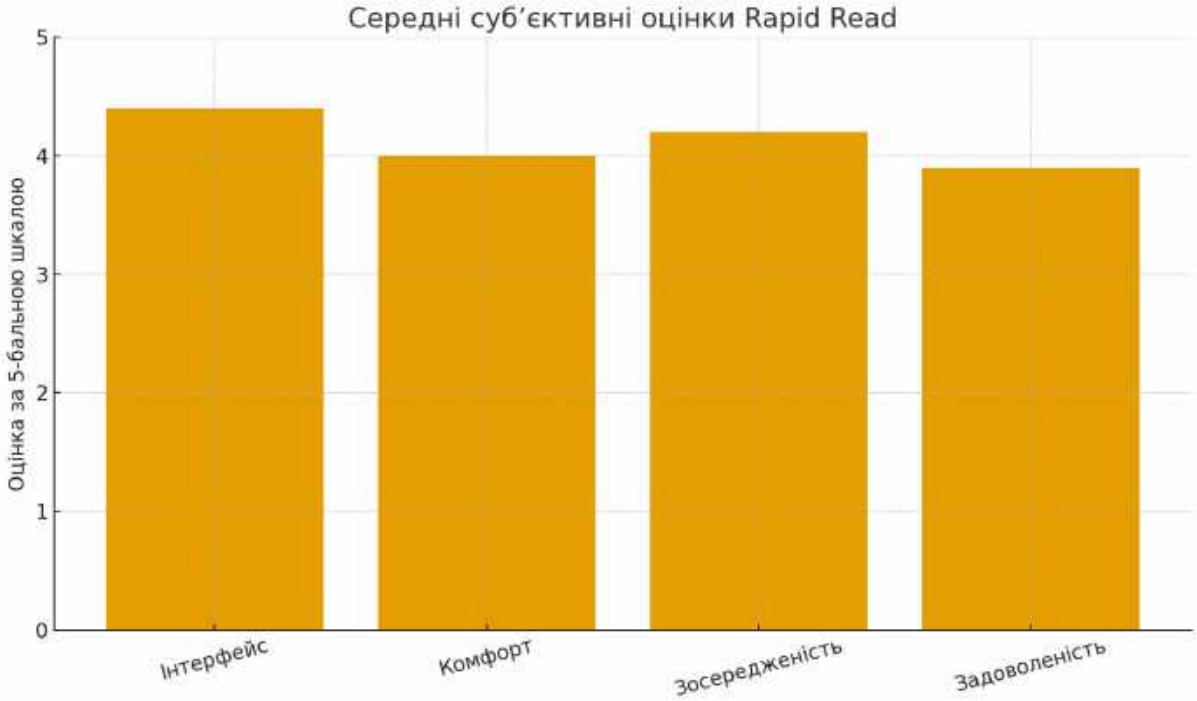

Рисунок 3.3 – Середні суб'єктивні оцінки зручності та комфортності використання мобільного застосунку Rapid Read

Узагальнюючи результати, можна сказати, що використання Rapid Read у режимі RSVP дозволило істотно підвищити швидкість читання при відносно невеликому зниженні показників розуміння тексту. Для більшості учасників перехід на RSVP означав відчутний приріст темпу (на десятки відсотків), але при цьому рівень comprehension залишився на прийнятному рівні, особливо за умови помірних швидкостей. Суб'єктивні оцінки показали, що інтерфейс сприймається як зрозумілий і зручний, а сам формат подачі тексту – як цілком комфортний для регулярного використання.

Отримані дані підтверджують практичну доцільність використання методу RSVP у мобільному застосунку для швидкісного читання та вказують на основні напрямки подальшого вдосконалення: покращення механізмів повернення до попереднього контенту, тонке налаштування швидкості та подальше підлаштування візуальних параметрів під індивідуальні особливості користувачів.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено та досліджено мобільний застосунок Rapid Read для швидкісного читання текстів на основі методу Rapid Serial Visual Presentation (RSVP) на платформах Android та iOS. Робота поєднує теоретичний аналіз когнітивних особливостей читання з практичною реалізацією програмного продукту та пілотним експериментом. Отримані результати підтверджують, що використання RSVP на мобільних пристроях дає змогу підвищити швидкість читання за умови збереження прийнятного рівня розуміння тексту та забезпечення зручного інтерфейсу. Мета роботи досягнута, усі поставлені завдання виконано.

Виконано аналіз когнітивних механізмів читання (зорове сприйняття, увага, робоча пам'ять, семантична інтеграція) та їхніх природних обмежень. Показано, що саме ці обмеження визначають межі традиційного читання та мотивують використання технологій швидкісного подання тексту. Окремо проаналізовано принципи роботи методу RSVP, його переваги (зменшення саккад, стабільний темп) і недоліки (ускладнене повернення до попереднього контексту).

Розглянуто нативні, гібридні та кросплатформні підходи до розробки мобільних застосунків. Показано, що для задачі RSVP-клієнта критичними є продуктивність, плавність анімацій та точність таймінгів, що робить гібридні рішення менш придатними. На підставі аналізу архітектури та статистики використання обґрунтовано вибір стеку Flutter/Dart як оптимального для кросплатформної реалізації застосунку швидкісного читання.

Сформульовано функціональні вимоги до системи (імпорт і зберігання текстових файлів, режим RSVP-читання, збереження прогресу, статистика, налаштування, PIN-захист) та нефункціональні вимоги (продуктивність, плавність інтерфейсу, офлайн-робота, простота використання). Вимоги узгоджено з реальними сценаріями мобільного читання та особливостями методу RSVP, що створило цілісну специфікацію майбутнього застосунку.

Побудовано UML-діаграму варіантів використання, діаграму діяльності для процесу читання в режимі RSVP та діаграму класів інформаційної моделі системи. Виокремлено основні сутності (документ, сеанс читання, налаштування, статистика, доступ користувача) та визначено зв'язки між ними. Це дозволило формалізувати логіку роботи застосунку й закласти основу для побудови архітектури.

Запропоновано модульну архітектуру Rapid Read з розділенням на підсистеми: робота з файлами, підготовка та подання тексту в режимі RSVP, збереження прогресу та статистики, керування налаштуваннями та доступом. Описано принципи керування станом інтерфейсу та взаємодію між екранами (бібліотека, режим читання, налаштування, статистика, PIN-екран). Такий підхід забезпечив розширюваність та зручність подальшого супроводу системи.

Здійснено реалізацію кросплатформного мобільного застосунку Rapid Read засобами Flutter/Dart. Реалізовано основний функціонал: імпорт текстових файлів, бібліотеку документів, режим RSVP з керуванням швидкістю та паузою, збереження прогресу читання, локальну статистику, налаштування інтерфейсу та PIN-захист. Особливу увагу приділено плавності відображення слів та стабільній роботі інтерфейсу на мобільних пристроях.

Проведено базове функціональне тестування для перевірки коректності роботи основних сценаріїв (імпорт, читання, збереження прогресу, налаштування, доступ). На основі пілотного дослідження отримано попередні кількісні та якісні оцінки: використання Rapid Read у режимі RSVP дозволяє підвищити середню швидкість читання порівняно з традиційним способом при прийнятному рівні розуміння тексту та позитивних суб'єктивних оцінках зручності інтерфейсу. Це підтверджує доцільність обраного підходу та створює основу для подальших, більш масштабних досліджень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Arnold L., A Systematic Literature Review of Eye-Tracking and Machine Learning Methods for Improving Productivity and Reading Abilities. 2025. Vol. 15, no. 6. P. 3308. URL: https://doi.org/10.3390/app15063308 (date of access: 16.09.2025).

2. Illustration of the perceptual span during reading. ResearchGate. URL: https://www.researchgate.net/figure/a-Illustration-of-the-perceptual-span-during-reading-b-Demonstration-of-the_fig11_307477053 (date of access: 16.09.2025).

3. The Science of Reading: A Handbook. URL: https://www.amazon.com/Science-Reading-Blackwell-Developmental-Psychology/dp/1119705096/ (date of access: 16.09.2025).

4. Degno F., Liversedge S. P. Eye Movements and Fixation-Related Potentials in Reading: A Review. Vision. 2020. Vol. 4, no. 1. P. 11. URL: https://doi.org/10.3390/vision4010011 (date of access: 18.09.2025).

5. Conceptual illustration of eye movements while reading. ResearchGate. URL: https://www.researchgate.net/figure/Conceptual-illustration-of-eye-movements-while-reading_fig4_310749284 (date of access: 20.09.2025).

6. Logie R., Camos V., Cowan N. Working Memory: State of the Science. Oxford University Press, 2020. 464 p. URL: https://doi.org/10.1093/oso/9780198842286.001.0001 (date of access: 24.09.2025).

7. Shin J. A meta-analysis of the relationship between working memory and second language reading comprehension: Does task type matter?. Applied Psycholinguistics. 2020. Vol. 41, no. 4. P. 873-900. URL: https://doi.org/10.1017/s0142716420000272 (date of access: 26.09.2025).

8. Voiskounsky A. E., Solodov M. Y. How features of digital text affect reading efficiency and comprehension. Psychology in Education. 2020. Vol. 2, no. 2. P. 134-142. URL: https://doi.org/10.33910/2686-9527-2020-2-2-134-142 (date of access: 26.09.2025).

9. RSVP paradigm (color mode). ResearchGate. URL: https://www.researchgate.net/figure/RSVP-paradigm-color-mode-At-the-beginning-the-word-is-presented-then-the-target_fig2_49627394 (date of access: 01.10.2025).

10. Rapid serial visual presentation in reading: The case of Spritz. The Sixth W. URL: https://www.tsw.it/wp-content/uploads/Rapid-serial-visual-presentation-in-reading-The-case-of-Spritz-1.pdf (date of access: 03.10.2025).

11. Swift. Apple Developer. URL: https://developer.apple.com/swift/ (date of access: 06.10.2025).

12. Kotlin Programming Language. Kotlin. URL: https://kotlinlang.org/ (date of access: 08.10.2025).

13. The Cross-Platform App Development Leader. Ionic Framework. URL: https://ionicframework.com/ (date of access: 11.10.2025).

14. Learn once, write anywhere. React Native. URL: https://reactnative.dev/ (date of access: 13.10.2025).

15. .NET Multi-platform App UI documentation. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-10.0 (date of access: 16.10.2025).

16. Flutter architectural overview. Flutter documentation. URL: https://docs.flutter.dev/resources/architectural-overview (date of access: 20.10.2025).

17. 2024 Stack Overflow Developer Survey. Stack Overflow Insights. Developer Hiring, Marketing, and User Research. URL: https://survey.stackoverflow.co/2024/technology (date of access: 22.10.2025).

18. Cross-platform mobile frameworks used by global developers 2023 Statista. URL: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (date of access: 23.10.2025).

19. About the Unified Modeling Language Specification Version 2.5.1. Object Management Group. URL: https://www.omg.org/spec/UML/2.5.1 (date of access: 03.11.2025).