МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО
«ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
“МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ”»

ФАКУЛЬТЕТ КОМП’ЮТЕРНО-ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка мобільного застосунку для візуалізації робочих процесів на основі методології Kanban»

Шифр групи: ІК-9-22-Б1ІПЗ(4.0д) Бурдим Андрій Олександрович

Спеціальність: Інженерія програмного забезпечення

Робота на здобуття освітньо-кваліфікаційного рівня бакалавр

Науковий керівник _____ Яременко Д. М.
(підпис)

Допущено до захисту ДЕК

Завідувач кафедри _____ професор, док. екон. наук
(підпис) Кавун С.В.

Київ 2026

РЕФЕРАТ

Дипломна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел, додатків, загальним обсягом робота складає 88 сторінок, має 18 рисунків, 3 таблиці, 1 сторінка додатків. Список використаних джерел містить 28 найменувань і займає 4 сторінки.

Метою дипломної роботи є розробка рішення, яке відповідатиме цілям та завдан-ням кінцевого користувача, а саме, допоможе в організації робочого процесу, де основним елементом є рішення, час створення та/або впровадження якого не є сталим показником, а радше робочим процесом, що вимагає креативності, аналі-зу, інновативності та включає інші елементи невизначеності.

У дипломній роботі розглянуті питання адаптації класичних командних канонів методології Kanban до специфіки індивідуальної діяльності людини. Проналізовано популярні ринкові аналоги, зокрема Jira, Trello, ClickUp та Notion. Потрібно сказати, що під час аналізу виявлено їхню значну функціональну надлишковість, яка часом лише заважає зосередженню, та жорстку залежність від мережі Інтернет. Програма працює абсолютно автономно. Спроєктовано та практично реалізовано мобільний застосунок, де надійне збереження об'єктних моделей базується на принципі offline-first через локальну серіалізацію у формат JSON. Сказане дає змогу зробити висновок про високу швидкість відгуку інтерфейсу додатка.

Ключові слова: методологія Kanban, мобільний застосунок, концепція «Ма», заспокійливі технології, ліміти незавершеної роботи, гнучке налаштування, індивідуальна ефективність, офлайн-розробка.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

СДУГ – Синдром дефіциту уваги та гіперактивності

ТС – Test Case (тестовий сценарій / тест-кейс);

JIT – Just in Time – точно-в-термін

WIP – Work in Process – робота в виконанні

UI – User Interface

UX – User Experience

JSON – JavaScript Object Notation (текстовий формат обміну даними, що використовується для локального збереження);

ВСТУП

Будь-яка діяльність без чіклого розуміння її внутрішньої архітектури зазвичай приречена на хаос. Автор переконаний, що виконавчий процес стає абсолютно неефективним, якщо людина втрачає орієнтири і чітко не усвідомлює, що і якими інструментами потрібно робити в конкретний проміжок часу. Сформулювати первинний план – це лише половина справи. Але що відбувається прямо зараз? Чи рухається процес у правильному руслі? Коли саме потрібно коригувати траєкторію? На практиці людина постійно зіштовхується з цими питаннями. Саме тут виникає гостра потреба в гнучкій системі, яка б дозволяла не просто фіксувати завдання, а й відстежувати їхній реальний стан на кожному життєвому етапі виконання. Логічним виходом із цього становища є застосування методології Kanban, яка першочергово будується на принципах наочності.

З розвитком цифрових технологій та широким поширенням мобільних пристроїв постала необхідність в адаптації існуючих методологій до нових умов роботи. Сьогодні мобільні застосунки стали невід'ємною частиною повсякденного життя, а їх використання в управлінні проектами набирає популярності. Зазвичай великі комерційні платформи на кшталт Jira, Trello орієнтовані на командну роботу, постійну синхронізацію даних між колегами та оперативний обмін інформацією в мережі. Проте, на думку автора, для індивідуального фахівця такий надлишковий функціонал часто перетворюється на додаткове джерело відволікання.

Зважаючи на це, метою кваліфікаційної роботи є розробка мобільного програмного рішення, яке максимально точно відповідатиме цілям та потребам кінцевого користувача в межах його особистої ефективності. Мова йде про допомогу в організації складного робочого процесу. Особливо там, де час створення чи впровадження кінцевого продукту не є фіксованим показником, а радше являє собою тривалий інтелектуальний шлях. Такий процес вимагає глибокого аналізу, креативності та інноватівності, а отже – апріорі містить у собі чималу частку невизначеності.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Ефективне проєктування мобільного інструменту візуалізації персональних робочих процесів неможливе без попереднього вивчення специфіки предметної області та декомпозиції існуючих методологічних рішень. У межах цього розділу здійснюється системний аналіз концепції Agile-планування з фокусом на застосування інструментарію Kanban для управління індивідуальними завданнями. Оцінка ринку сучасного програмного забезпечення та аналіз обмежень популярних комерційних таск-менеджерів дозволяють чітко окреслити проблемне поле, обґрунтувати актуальність створення автономного локального рішення та закласти стійкий теоретичний фундамент для його подальшої інженерної реалізації.

1.1. Огляд методологій управління проєктами

Управління проєктами є комплексним процесом, що включає планування, організацію, координацію та контроль виконання робіт з метою досягнення поставлених цілей у визначені строки та з обмеженими ресурсами. Вибір методології управління проєктом суттєво впливає на ефективність реалізації проєкту, особливо у сфері розробки програмного забезпечення, де вимоги часто змінюються, а результати мають бути отримані у стислі терміни.

На сьогодні існує велика кількість методологій управління проєктами, які умовно поділяються на класичні та гнучкі (Agile) підходи [7].

Чи можна уявити сучасну розробку без гнучких маневрів? Мабуть, важко, але фундаментом всього залишається класика. Мова йде про каскадну модель, або ж Waterfall, яка, на думку автора, є втіленням суворої лінійності. Тут все просто: спочатку аналізуємо вимоги, потім проєктуємо, далі – реалізація, і лише в кінці тестування з впровадженням. Жодного кроку вперед, поки не закрито попередній етап.

Звісно, подібний підхід дає відчуття контролю. Коли кожен документ на своєму місці, а результати можна спрогнозувати на місяці вперед – це

підкуповує. Проте, виникає логічне питання: а що робити, якщо вимоги раптом зміняться посеред шляху? Тут і криється головний камінь спотикання. Waterfall – це не про маневреність. Автор зауважує, що низька адаптивність робить цю модель вкрай незручною для динамічних середовищ, де зміни є єдиною константою. Це як будувати величезний корабель за кресленнями, які неможливо підправити після спуску на воду.

«Kanban – це методологія управління виробництвом, проєктами. Є найбільш гнучким підходом Agile-філософії управління проєктами та розробки програмного забезпечення, яка спрямована на гнучкість, швидкість та відкритість у співпраці між розробниками та замовниками. [7]»

Етимологія поняття Kanban сягає корінням у японську мову, де семантичне ядро складається з двох елементів: «кан» (видимий/візуальний) та «бан» (картка чи дошка) [12]. У ширшому лінгвістичному контексті термін трактується як «бирка» або «знак». Чи є це просто маркуванням? Автор вважає, що ні. У виробничому циклі канбан постає як контрольна картка або наряд-замовлення, що нерозривно супроводжує кожен виріб. Прикріплена до вузла чи деталі, така одиниця інформації чітко сигналізує про генезис компонента та його подальший логістичний шлях.

Фактично, ми маємо справу з розгалуженою інформаційною мережею. Вона цементує завод у цілісний організм, синхронізуючи розрізнені процеси з вектором споживчого попиту [18]. Варто зауважити, що коріння методу лежить у площині машинобудування. Саме фахівці корпорації «Toyota» розробили цей інструментарій у межах концепції «точно-в-термін» [12]. 1962 рік став переломним, тоді на рейки JIT перейшли всі потужності гіганта. Досить сміливий крок для того часу.

Сьогодні управління інноваційно-інвестиційними процесами на підприємствах це справжній квест із багатьма невідомими. Попри популярність таких систем, як Kaizen, Kanban чи Six Sigma, ми все ще спотикаємося об ті самі виклики, які заважають цим практикам «прорости» на нашому ґрунті [16]. Як на мене,

основна проблема полягає не у відсутності знань, а в тому, що наявні дослідження часто існують відірвано від реальності.

Аналізуючи стан справ, автор виділяє кілька критичних прогалин, які потребують втручання:

1. Брак адаптації: Спроби просто скопіювати чужі методи без урахування галузевої специфіки зазвичай приречені на поразку.
2. Культурний бар'єр: Ми досі не маємо чіткого розуміння, як саме організаційна культура українських компаній «перетравлює» ці інновації – чи вона допомагає їм, чи, навпаки, відторгає.
3. Методологічний хаос: Відсутній уніфікований підхід до інтеграції цих інструментів у єдину систему.
4. Дефіцит вітчизняного досвіду: Більшість кейсів – західні, а досліджень, проведених саме в умовах українського бізнес-середовища, критично мало.
5. Відсутність метрик: Ми часто не знаємо, як реально виміряти ефективність впровадження та як ці методи впливають на фінансові результати в довгостроковій перспективі.

Власне рішення бачу в тому, щоб змінити фокус. Хоча наукових праць про окремі методики вистачає, комплексний погляд на їхній синергетичний ефект у нас практично відсутній. Автор вважає, що зараз існує нагальна потреба не просто цитувати міжнародний досвід, а розробляти конкретні, «приземлені» рекомендації, які враховуватимуть особливості нашого процесу виконання завдань. Потрібні інструменти, які працюватимуть тут і зараз, а не в ідеальних умовах підручника.

1.2. Характеристика Kanban як методології управління завданнями

Сформулювати сутність підходу можна наступним чином: Kanban (див. рис. 1.1) – це передусім інструментарій для управління робочими процесами[26]. Автор розглядає його як метод менеджменту для різноманітних професійних послуг, що за своєю природою є інтелектуальною працею. Застосування цього методу передбачає цілісний підхід до осмислення сервісів, де основний акцент зміщується на їхнє вдосконалення з позиції кінцевого замовника.

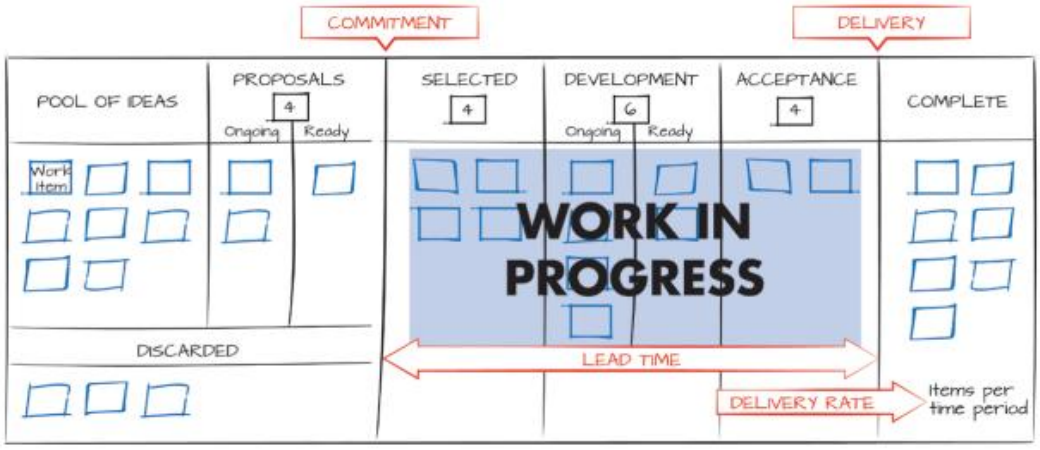


Рис. 1.1. Приклад Kanban застосунку [18]

Ключова роль у цій системі відведена візуалізації[18]. Оскільки інтелектуальна праця та її рух у межах робочого потоку зазвичай залишаються невидимими, Kanban дозволяє зробити ці процеси наочними [26, 18]. Це є критично важливим для ефективного управління бізнесом, зокрема для ідентифікації та менеджменту ризиків під час надання послуг. Завдяки впровадженню таких інструментів організація поступово розвиває адаптивну здатність, що дозволяє значно швидше реагувати на трансформації бізнес-середовища або зміну потреб клієнтів.

Варто також прояснити певну термінологічну плутанину: Kanban часто помилково ототожнюють із методологією або фреймворком. В інженерії програмного забезпечення під методологією розуміють прескриптивний підхід із чітко визначеними ролями та зонами відповідальності [26, 9]. Я вважаю за доцільне підкреслити, що Kanban не є процесною методологією у класичному розумінні.

Це, скоріше, метод управління, який варто застосовувати до вже існуючого способу роботи.

Отже, питання вибору між Kanban та певною методологією не є коректним. Навпаки, цей інструмент завжди впроваджується як доповнення до встановленого порядку роботи[26]. Автор бачить в цьому значну практичну цінність, оскільки метод не замінює існуючі практики, а слугує засобом їхньої оптимізації для стабільної відповідності очікуванням замовника.

Сам метод жодним чином не регламентує підходи до проектування дошок, що є принципово важливим фактором. Натомість обмеження привносять саме програмні інструменти, які підтримують Kanban: вони, як правило, пропонують уже готовий шаблон двовимірної сітки, де окремі панелі несуть інформацію про кожен конкретний робочий елемент[18].

Стовпці в такій структурі відповідають певним стадіям процесу. Причому деякі з них додатково розбиваються горизонтальними роздільниками – так звані «swimlanes», або ж «доріжки», що охоплюють кілька стовпців одразу й слугують для розмежування різних станів елементів у межах одного етапу[18]. А от команди, що конструюють фізичні дошки, тобто без будь-яких програмних рамок – поводяться інакше. Вони нерідко знаходять власні, часом доволі несподівані рішення для візуалізації потрібної їм інформації. Зокрема, автор звертає увагу на практику встановлення зв'язків між дошками різних підрозділів чи служб – це, по суті, вже інший рівень гнучкості, який стандартний шаблон просто не передбачає.

Хоча витоки системи пов’язані з ощадливим виробництвом, Kanban адаптований саме для інтелектуальної праці, результатом якої є нематеріальні товари та віртуальні послуги [26]. На відміну від виробничої сфери, тут ми маємо справу з активами, що не мають високих прямих витрат на зберігання, а варіативність у виконанні завдань визнається притаманною властивістю процесу. Тому першочергова увага приділяється саме підвищенню цінності та оптимізації потоку послуг, а не лише скороченню втрат.

Узагальнюючи, можна сказати, що Kanban базується на принципах ощадливості: фокусуванні на потоці задач, обмеженні обсягу незавершеної роботи та аналізі системи в цілому, а не індивідуальної продуктивності. На думку автора, саме такий еволюційний шлях розвитку, заснований на об’єктивних даних, дозволяє досягти максимальної ефективності в управлінні сучасними проєктами.

1.2.1. Обмеження роботи в процесі

Запровадження обмежень на обсяг незавершеної роботи (WiP-ліміти) фактично змінює саму логіку системи: замість «push»-підходу, коли завдання безперервно «проштовхуються» вперед, формується система «pull» – нове завдання не стартує раніше, ніж попереднє буде доведено до завершення або, у виняткових випадках, знято з виконання[14, 18]. Варто зазначити, що надлишок частково виконаної роботи – це не просто неефективність у загальному сенсі. Це пряма фінансова втрата. І що не менш суттєво, час циклу зростає, а разом із ним організація поступово втрачає спроможність вчасно реагувати: на запити клієнтів, на зміни зовнішнього середовища, на нові можливості, які не чекають.

Зовсім інша картина складається там, де управлінська логіка спрямована на максимальне завантаження людей і ресурсів. Постійна «зайнятість» заради самої зайнятості. Безперервний потік нових доручень, аби ніхто не «простоював». Автор зазначає, що за таких умов працівник, по суті, перетворюється на виконавця окремих інструкцій – він бачить лише своє завдання, але не бачить сервісу, частиною якого є. Широка картина розмивається. Розуміння того, як власна діяльність пов'язана із загальними цілями організації чи реальними потребами клієнта, поступово зникає, що мабуть, є найдорожчою прихованою ціною такого підходу.

1.2.2. Управління потоком

Потік роботи у Kanban підпорядкований кільком взаємопов'язаним цілям: максимізувати створення цінності, скоротити час виконання та зробити весь процес якомога передбачуванішим [9]. Звучить логічно, але на практиці ці цілі

нерідко тягнуть у різні боки. А оскільки результати роботи здебільшого є складними й неоднорідними, єдиним дієвим підходом залишається емпіричне управління: прозорість, регулярна інспекція, готовність адаптуватися[18]. Окремої уваги заслуговують вузькі місця, а саме, ділянки, де потік «стискається» через певний підпроцес, а також бар'єри, що виникають унаслідок залежностей від суміжних сервісів.

Тепер про те, що автор вважає ключовим у розумінні потоку цінності. Вартість затримки – це та частина цінності робочого елемента, яка безповоротно втрачається через відтермінування його реалізації [18]. Простіше кажучи: чим довше чекаємо, тим дорожче обходиться зволікання. Ця залежність є функцією часу, причому не обов'язково лінійною, швидкість «знецінення» може бути як стабільною, так і різко змінюватись залежно від обставин.

Щоб працювати з цим явищем системно, у Kanban застосовуються чотири архетипи, які описують, як цінність елемента реагує на затримку: expedite, fixed date, standard та intangible (див. рис. 1.2). Кожен із них – це, по суті, окрема логіка поведінки. На їх основі формуються класи обслуговування з відповідними політиками, що дозволяє приймати осмислені рішення щодо пріоритизації – а не просто діяти за принципом «що прийшло першим, те й виконуємо».

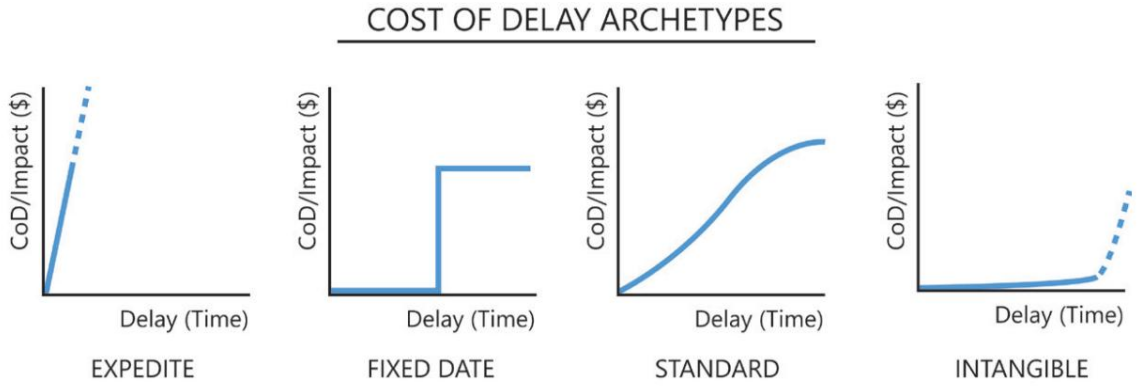


Рис. 1.2. Графіки ефективності архітектур вартість затрримки [18]

Відповідно до класифікації Дональда Райнертсена, в архітектурі застосунку логіка обробки дедлайнів та затримок має базуватися на чотирьох архетипах:

Критичний / Терміновий (Expedite): Характеризується стрімким лінійним зростанням збитків з першого моменту виникнення (наприклад, критичний баг, що блокує роботу всього застосунку на етапі тестування). В індивідуальному Kanban це завдання вимагає обходу стандартних лімітів незавершеного виробництва (WIP-лімітів) та негайного перемикання фокусу розробника.

Фіксована дата (Fixed Date): Ступінчаста функція, де ціна затримки є мінімальною до настання дедлайну (наприклад, дата релізу, дедлайн здачі модуля), після чого втрати зростають стрибкоподібно. Організація роботи передбачає інтеграцію алгоритму зворотного планування (backward scheduling) на основі історичних даних про середній час виконання схожих задач (Cycle Time).

Стандартний (Standard): Плавна S-подібна залежність втрати цінності продукту на ринку. Для розробника програмного забезпечення оптимізація таких завдань здійснюється через інтерфейс застосунку за допомогою алгоритму WSJF (Weighted Shortest Job First) – у роботу першими беруться фічі з найвищою цінністю та найкоротшим часом реалізації.

Нематеріальний / Прихований (Intangible): Протягом тривалого часу ціна затримки прагне до нуля, проте має точку перегину з подальшим експоненціальним зростанням. Типовий приклад – технічний борг або рефакторинг коду. Специфіка індивідуальної розробки полягає в тому, що за відсутності командного контролю ці завдання часто ігноруються, що згодом призводить до системного збою і раптової трансформації задачі в категорію Expedite.

1.2.3. Чіткість політик

Явні політики – це не просто регламент правил. Це спосіб формалізувати процес так, щоб він виходив за межі схематичного опису робочого потоку[18]. Разом із самим потоком політики створюють систему обмежень, яка одночасно окреслює простір для дій і задає характеристики, що виникають емерджентно тобто ті, які неможливо передбачити заздалегідь, але можна коригувати через експерименти.

Щодо вимог до таких політик, автор формулює доволі чітко: мінімалістичність, простота, видимість, послідовне застосування та можливість легко змінювати їх тими, хто безпосередньо надає послугу. І тут є один тонкий момент, на якому варто зупинитися окремо. Принципи «постійного застосування» та «легкої змінюваності» не суперечать одне одному, вони нерозривно пов'язані. Встановити WiP-ліміти й далі ніколи їх не переглядати, не порушувати тимчасово заради перевірки, чи спрацює інше значення – це вже не управління процесом, а його омертвіння [3, 18].

Поведінку складних систем не можна точно прорахувати наперед, навіть якщо вони керуються простими правилами. Політики, які здаються очевидними на інтуїтивному рівні: «чим раніше почнеш, тим раніше закінчиш», насправді нерідко дають цілком протилежний ефект [18]. Саме тому явне визначення та постійна видимість політик є не формальністю, а робочим інструментом: команда має розуміти, за якими правилами діє, і мати зрозумілий механізм для їх перегляду, коли ці правила перестають працювати або просто ігноруються [14].

До речі, WiP-ліміти – лише один із різновидів політик. Серед інших: правила розподілу та балансування потужностей, «Definition of Done» та інші критерії виходу елементів зі стадій процесу, правила поповнення черги із вибором нових завдань за наявності вільної потужності, а також класи обслуговування з їх відповідними пріоритетами.

1.3. Практичне використання Kanban-дошки

Процес побудови класичного інструменту Kanban незмінно починається з декомпозиції робочого потоку, тобто з банального визначення етапів, які згодом трансформуються у назви відповідних стовпців [8]|. На думку автора, тут важливо зафіксувати вектори, крізь які безпосередньо прокручується життєвий цикл продукту, послуги чи типових завдань команди. Як саме адаптувати цей простір під реальні бізнес-процеси? Автор пропонує розглянути кілька базових варіацій, що демонструють гнучкість системи. Наприклад, у сфері послуг ланцюжок може

виглядати як послідовність від первинного телефонного чи Skype-дзвінка до зустрічі, підписання офіційного договору та безпосередньої передачі технічного завдання профільному фахівцю.

Маркетингові або торгівельні напрямки вимагають зовсім інших маркерів. Для контент-маркетингу це буде шлях від розробки стратегії, її затвердження замовником до безпосередньої генерації контенту й запуску таргетованої реклами (див. рис. 1.3), тоді як у ритейлі (наприклад, при продажу одягу) логіка диктує лінійний ланцюг: отримання заявки – звортній зв'язок з клієнтом – збір товару – відправка через оператора – фіксація оплати. Досить простий, але робочий алгоритм.

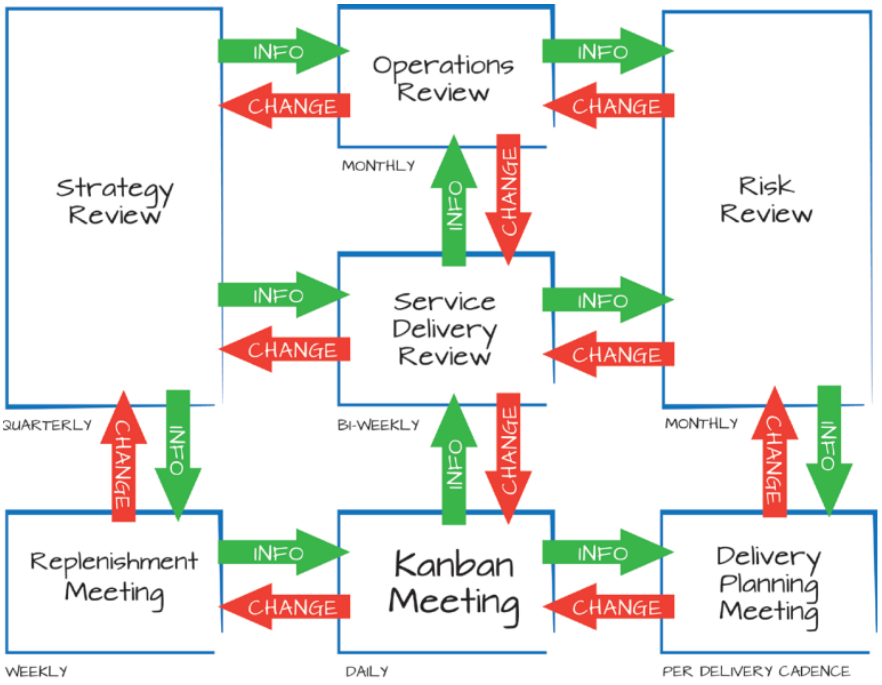


Рис. 1.3. Набірвипадків, що показують петлі зворотного зв'язку [18]

Фундаментом взаємодії в межах цієї концепції виступає безпосередньо Kanban-дошка. Вона є базовим інструментом для наочної візуалізації та тонкого налаштування робочих потоків. Матеріальні, так звані фізичні дошки й досі мають своїх палких прихильників серед команд – і це цілком виправдано через їхню тактильність та простоту, проте віртуальні аналоги стають критично важливими, коли мова заходить про сучасну гнучку розробку програмного забезпечення. Чому так? Відповідь криється в територіальній розподіленості фахівців, потребі

у наскрізному трекінгу задач та миттєвому доступі до інформації з будь-якої географічної локації.

Зрештою, формат носія не змінює головної суті. Дошка забезпечує прозорість дій, стандартизує операційні кроки команди та дозволяє оперативно помічати «вузькі місця» – певні затики, блокування чи критичні залежності між тасками[23]. Зазвичай базова модель пропонує класичну тріаду етапів: To Do, In Progress та Done [3, 26]. Але чи є вона універсальною для будь-якого бізнес-процесу? Автор вважає, що ні. Жорстка шаблонність тут лише шкодить, адже архітектуру дошки варто адаптувати під унікальні цілі проєкту. Зокрема, у розробленому автором мобільному застосунку для візуалізації процесів пропонується модифікована схема з можливими чотирма статусами завдання: планування, в роботі, перевірка та готово. Таке рішення, на думку автора, є значно ефективнішим для IT-напрямку, креативних індустрій або навіть для користувачів із синдромом дефіциту уваги, оскільки деталізація контролю спрощує фокусування на завданнях, які не мають жорстких дедлайнів, та сприяє процесу осмислення та покращення виконання конкретної задачі.

Оскільки методологія Kanban тримається на принципах абсолютної відкритості та комунікації в реальному часі, дошка стає єдиним валідним джерелом достовірних даних для всієї структури [23]. Сказане дає змогу зробити висновок, що цей інструмент за умови правильного моделювання повністю нівелює інформаційний хаос, фокусуючи команду на реальному перебігу розробки без зайвої бюрократії та суб'єктивних звітів.

Картки чи стікери, які мігрують між цими стовпцями, зазвичай наповнюються конкретикою – іменами клієнтів або локальними тасками. У колону «Створення контенту» логічно помістити дрібніші завдання, скажімо, підготовку п’яти публікацій для мережі Facebook чи аналогічної кількості для Instagram. Обмежень немає. Все залежить від стеку потреб та особистого комфорту виконавців.

Існує, втім, альтернативний шлях організації дошки, який автор вважає найбільш раціональним для швидкого старту через його мінімалізм. Йдеться про класичну тріаду стовпців: Backlog, In progress, Done (див. рис. 1.4) [8]. Завдання

спочатку акумулюються в беклозі, а в міру залучення виконавця переміщуються у статус «в роботі», фінішуючи у фінальній колонці після завершення всіх процесів. Варто зауважити, що у власному розробницькому рішенні автора ця схема дещо модифікована до чотирьох можливих статусів (планування, в роботі, перевірка, готово), що дозволяє більш гнучко контролювати статус виконання проєкту.

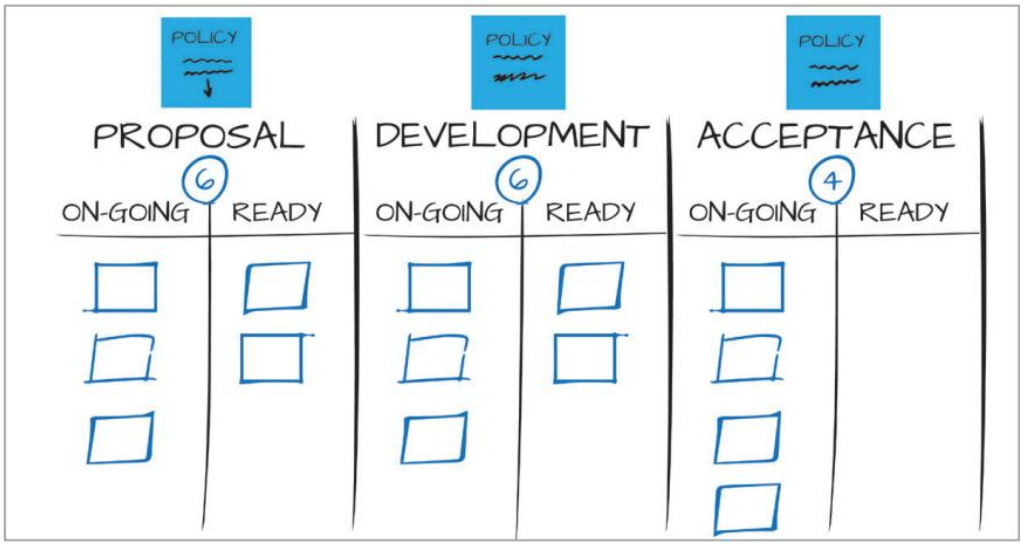


Рис. 1.4. Схема типової архітектури Kanban [18]

Щодо технічного втілення, то на початкових етапах інтеграції команди в систему автор рекомендує використовувати матеріальні носії – звичайний ватман, маркерну або коркову дошку зі стікерами. Фізичний контакт із завданнями покращує сприйняття методології на перших порах. Згодом, за потреби, здійснюється міграція на цифрові аналоги. Відомим прикладом такої хмарної системи є Trello, використання якої стає безальтернативним у випадку взаємодії з географічно розподіленими (віддаленими) членами команди. Отже, сказане дає змогу зробити висновок, що формат візуалізації підбирається суто під архітектуру конкретної команди для досягнення максимального результату.

1.4. Перехід на Kanban

Концепція Kanban пропонує раціональний і лаконічний підхід до генерації високоякісної цінності для кінцевого користувача, причому без виходу за межі бюджетних та часових лімітів [20]. За наявності чіткого пулу завдань і сформованої команди розгортання такої системи можна розпочати негайно. Як саме?

Першим кроком є фіксація високорівневої рутини виконавців. Потрібно розуміни, що повсякденна діяльність команди зазвичай є досить неоднорідною: це і комунікація з партнерами, і перманентний менеджмент електронної пошти, і виправлення операційних помилок, і збір відгуків [20]. Проте Kanban фокусує основну увагу на найбільш суттєвому векторі – розробці вдосконалень для продукту чи інфраструктури. Звісно, трапляються і критичні помилки (підготовка масштабних презентацій чи написання фундаментальних проєктних документів), які автор вважає за доцільне прирівнювати до стандартних процесів модернізації. На початкових етапах варто прагнути до максимальної лінеаризації процесів.

Для оптимізації рутини рекомендовано повністю виключати ті етапи, що відбуваються до або після безпосередньої зони відповідальності команди. Ба більше, якщо дві суміжні ітерації стабільно виконуються одним фахівцем (наприклад, написання коду, тестування та відгук), їх логічно об'єднати в один статус [18, 20]. Такий розподіл є універсальним. Він чудово підходить як для класичного IT-сектору, так і для користувачів із синдромом дефіциту уваги, оскільки усуває надмірну деталізацію, яка лише дезорієнтує людину під час роботи. За наявності сумнівів щодо архітектури дошки стандартний ланцюжок «специфікація – реалізація – перевірка – доставка» може слугувати надійною точкою відліку.

Наступний крок передбачає безпосереднє оновлення інформаційного простору – розміщення зафіксованих етапів на дошці в радіусі візуального доступу команди. Етимологія терміна «Kanban» доволі глибока: у системі «Toyota» це означало сигнальну картку, тоді як китайська інтерпретація апелює до процесу

споглядання дошки під час щоденних координаційних зустрічей. Можна використовувати класичні магнітно-маркерні дошки чи звичайні стікери на стіні [20]. Виникає закономірне питання: чому б не обмежитися виключно цифровими інструментами на кшталт відомого сервісу Trello? Візуалізація є наріжним каменем методу, і тактильна взаємодія з картками на початку стимулює когнітивне сприйняття значно краще. Але коли мова йде про розподілені команди або специфічні завдання, що не мають жорстких графіків, віртуальні рішення стають безальтернативними. Власне, саме тому автором і розробляється мобільний застосунок для візуалізації процесів, щоб поєднати мобільність із наочністю. Практика показує, що розміщення фізичної дошки поруч із робочими місцями формує відчуття спільної ідентичності, хоча офіси вздовж єдиного коридору забезпечують не менш ефективну спонтанну комунікацію, ніж робота в одному опенспейсі.

Третім, і, мабуть, найважливішим етапом є встановлення лімітів для хаосу, який іманентно притаманний будь-якій командній праці. Своєрідний маркер зрілості методології – це те, як саме вона нівелює цей хаос. У каскадній моделі Waterfall бар'єром виступає жорстке попереднє проектування та процедури запитів на зміни. У Scrum – фіксовані спринти. Kanban діє витонченіше, вводячи обмеження на обсяг незавершеної роботи. Це обмеження кількості карток у стовпчику виконує дві фундаментальні функції. По-перше, воно локалізує зону ураження при зміні пріоритетів чи архітектурних рішень. По-друге, ліміти регулюють загальну швидкість системи відповідно до пропускної здатності найповільнішого етапу. Адже вище голови не стрибнеш. Немає жодного сенсу продукувати завдання швидше, ніж їх здатне пропустити так зване «вузьке місце» [20].

Для початку автор рекомендує встановлювати WIP-ліміт для найслабшої ланки на рівні чисельності задіяних там працівників плюс 50% буфера. Надалі є сенс пропорційно масштабувати ліміти для інших статусів, постійно калібруючи значення на основі емпіричних даних для досягнення максимальної продуктивності. Сказане дає змогу зробити висновок, що Kanban є не просто інструментом

відображення завдань, а гнучким регулятором внутрішнього клімату проєкту, який оптимізує навантаження та усуває деструктивний інформаційний шум.

1.4.1. Адаптація після Waterfall

За своєю внутрішньою будовою Kanban є максимально прозорим і оперує загальнозрозумілими категоріями. Саме тому широке коло фахівців здатне інтегрувати його у свою практику без тривалого інструктажу чи подолання високих порогів входження. Звісно, адаптація потребує часу. Зазвичай це кілька тижнів на звикання та місяці для глибокого освоєння, але перші результати помітні майже одразу. Дивно, але навіть фахівці зі старим загартуванням швидко підхоплюють цей ритм. Уже за короткий проміжок часу приріст продуктивності, гнучкості та передбачуваності стає цілком вимірним, як для керівної ланки, так і для пересічних виконавців.

Що мається на увазі під класичним підходом Waterfall, який часто називають каскадним? Це звичка проєктувати, кодувати та тестувати завдання величезними пакетами в межах етапів, що тривають місяцями. Автор, аналізуючи світовий досвід таких гігантів, як Microsoft чи Boeing, зазначає, що подібна інертність часто гальмує процеси. З погляду студента, який досліджує розробку мобільного застосунку для візуалізації процесів, традиційні каскадні моделі є надто громіздкими через велику кількість специфікацій. На противагу цьому, у власному рішенні пропонується спрощена структура з трьох базових статусів: планування, в роботі, перевірка та готово. Таке структурування значно полегшує сприйняття завдань, які не мають жорстко фіксованого часу на виконання.

Щоб традиційна Waterfall-команда прийняла Kanban, необхідно зробити три речі: чітко пояснити, навіщо зміни взагалі потрібні; спертися на той реальний досвід, яким люди вже володіють; і скоригувати звичні практики рівно настільки, щоби процес став більш плавним і швидким. Після того як команда освоїться з новою моделлю, вона сама зможе вирішити, чи варто рухатися далі, до більш радикальних трансформацій. Але стартова точка лишається знайомою.

Варто розібратися, де саме традиційний Waterfall починає давати збої. За правильного виконання він стартує з твердої основи: ретельне дослідження ринку, прототипування, архітектурне проєктування та задокументовані вимоги – все це складає добре структурований план із зрозумілими залежностями та розподілом ресурсів. Такі плани зазвичай розраховані на рік і більше та розбиті на послідовні фази з фінальним етапом стабілізації.

Та реальність, як відомо, вносить корективи. Вже за кілька місяців після старту з'являються нові невизначеності, вимоги зміщуються, залежності перебудовуються, склад команди змінюється. Плани доводиться переписувати, частину роботи відкидати або переробляти. І ця ситуація не разова: цикл повторюється знову й знову в міру того, як ринок і відгуки клієнтів продовжують «тиснути» на продукт у процесі розробки. Накопичені непослідовності призводять до проблем із якістю, які, своєю чергою, розтягують і без того довгий етап стабілізації і дедлайни зсуваються.

На початку впровадження Kanban важливо зняти тривогу команди. Варто донести просту думку: «Фактично все, що ви робили раніше, залишиться. Нові ролі, нова термінологія – нічого з цього вам не нав'язуватимуть. Акцент зміщується на якість і на те, щоб найважливіша робота виконувалась у першу чергу. Щоденна розробка залишається такою ж. Єдиною відчутною зміною є спосіб, яким команда вирішує, що робити наступним. І цей вибір відображається на дошці, яку бачать усі» [20].

«Рисунки 1.5 та 1.6 є прикладами графіків виконаних завдань і невирішених дефектів за чотири дев'ятитижневі періоди. Перші два періоди відповідають етапам методології Waterfall: шість тижнів відведено на реалізацію та три – на валідацію. Наступні два періоди представляють безперервне виконання за методологією Kanban. Команда працює сім днів на тиждень (для спрощення розрахунків, а не як форма покарання). Темп виконання завдань залишається незмінним упродовж усіх чотирьох періодів – це є штучним допущенням, проте не вносить систематичного зміщення. (До результатів додано незначний випадковий шум з метою підвищення реалістичності отриманих даних.) [20, ст 63]»

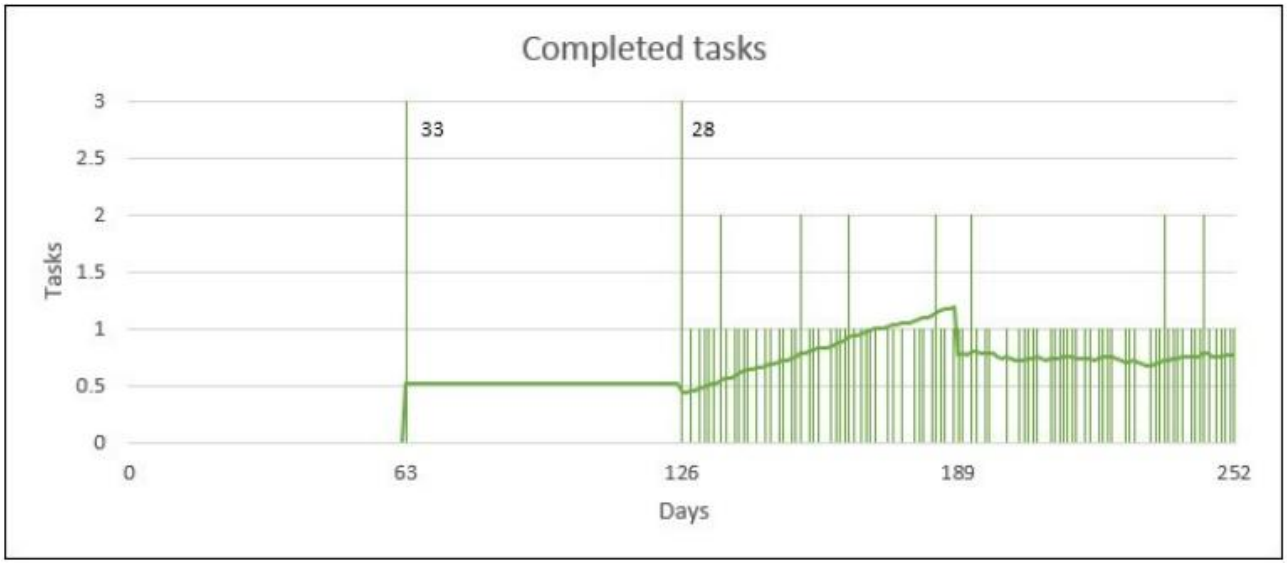


Рис. 1.5. Графік виконання завдань протягом чотирьох дев'ятитижневих періодів з накладеним ковзним середнім [20]

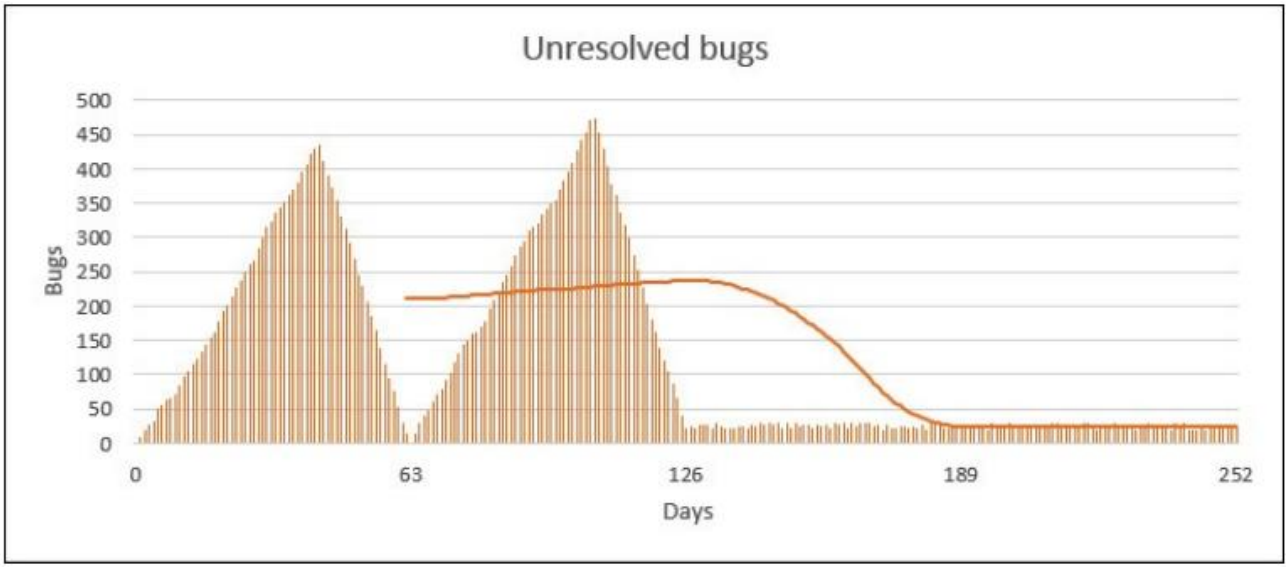


Рис. 1.6. Графік невирішених помилок протягом чотирьох дев'ятитижневих періодів з накладеним ковзним середнім [20]

Розглядаючи графік виконаних завдань, можна спостерігати, що всі завдання вважаються завершеними в останній день валідації впродовж перших двох періодів Waterfall. Два стовпці виконаних завдань за ці періоди перевищують висоту графіка приблизно в десять разів. Для зручності перегляду лінії ковзного середнього за дев'ять тижнів, що відображає продуктивність, масштаб було зменшено. Початкове значення цього показника становить приблизно 0,52 завдання на день (33 завдання / 63 дні), після чого зростає до

приблизно 1,2 завдання на день у міру того, як результати другого етапу Waterfall поєднуються з результатами Kanban, а згодом стабілізується на рівні близько 0,76 завдання на день із встановленням рівномірного потоку робіт у рамках Kanban [20].

Зростання показника з 0,52 до 0,76 відповідає підвищенню продуктивності на 46 відсотків. Це еквівалентно ситуації, коли, працюючи п'ять днів, команда виконує обсяг роботи, розрахований на сім із половиною днів – як якщо б вона напружено працювала у вихідні та при цьому все одно отримала їх у своє розпорядження [20]. Щоденна гнучкість Канбану також одразу очевидна, оскільки робота виконується щодня, а не постачається оптом.

Автор вважає, що саме «видимість» процесу є однією з головних переваг Kanban-підходу – особливо в контексті розроблюваного мобільного застосунку, де прозорість статусів завдань та можливість відстежувати прогрес по колонках дошки безпосередньо відповідає потребам кінцевого користувача. Це не просто управлінський інструмент, а радше спосіб мислення про роботу.

1.4.2. Адаптація після Scrum

Команди, які тривалий час практикують Scrum, зазвичай добре розуміють, що таке справжня гнучкість управління. Свого часу, перейшовши з класичного каскадного підходу (Waterfall), такі колективи змогли гідно оцінити очевидні переваги нового формату, хоча деяким фахівцям і досі може бракувати колишнього суворого розподілу обов’язків та, так би мовити, залізобетонної структурованості [18, 20]. Scrum справді непогано працює як поетапна методика для швидкого передавання готового продукту замовникам.

Проте тут є свої підводні камені, штучні обмеження. Часові рамки спринтів часто заганяють команду у глухий кут, заважаючи вчасно реагувати на зауваження клієнтів, коригувати плани чи випускати термінові оновлення. За ідеальних умов розробники мали б реагувати на зовнішні зміни миттєво, але Scrum примусово прив'язує будь-які дії до початкових нарад або до ретроспектив наприкінці циклу [20]. Звісно, це рятує від постійного смикання під час поточної

роботи. Але якою ціною? Плани затримуються, відгуки збираються повільніше, а кількість обов'язкових зустрічей лише зростає, відбираючи дорогоцінний час. Kanban натомість повністю усуває ці надумані кордони спринтів, забезпечуючи безперервний перебіг завдань.

На думку автора, такий диктат жорстких часових відрізків у Scrum створює надлишковий психологічний тиск на виконавців. У власному дослідженні, що присвячене створенню мобільного застосунку для унаочнення процесів, пропонується раціональна альтернатива – повна відмова від штучного часового лімітування. Це значно спрощує життя людям, які виконують творчі чи креативні завдання без фіксованого графіку, або ж користувачам із синдромом дефіциту уваги, для яких класичні дедлайни спринтів стають деструктивним чинником, що знижує внутрішню мотивацію.

До того ж Scrum перевантажений специфічною мовою. Усі ці терміни: sprint, increment, Daily Scrum, Product Backlog, Sprint Backlog, Scrum Master чи Product Owner – здатні добряче заплутати нову людину в команді. Чи сприяє це швидкому зануренню в роботу? Навряд чи. У Kanban все набагато простіше, адже метод дозволяє залишати будь-які звичні для колективу назви та ролі. Новачки адаптуються значно швидше, бо їм не доводиться вчити новий глосарій.

Звісно, люди схильні чинити опір змінам, навіть коли йдеться про заміну Scrum на Kanban. Щоб уникнути зайвого тертя, не варто подавати Kanban як радикальну революцію чи повне закреслення попередніх здобутків. Доцільніше показати його як природне продовження еволюції гнучкого підходу. Колектив треба просто заспокоїти. Наприклад, пояснити, що щоденна праця розробника залишиться незмінною, але зникне купа паперової та організаційної тяганини. Ми просто зменшимо кількість нарад, почнемо відразу бачити «вузькі місця» в роботі та чітко домовимося про правила завершення кожного окремого крокум [18].

При такому підході вся команда стає відповідальною за усунення перешкод, намагаючись стабільно видавати якісний результат. Керівник проєкту при цьому бере на себе роль інформаційного щита, транслюючи звіти про поступ

вищому керівництву організації, щоб не відволікати інженерів від суті [20]. Безпосередня взаємодія з клієнтами заохочується для всіх, хоча саме аналітики стежать за дотриманням інтересів замовника, навіть якщо позиція єдиного власника продукту відсутня.

«Рисунок 1.7 та 1.8 є прикладами графіків виконаних завдань і невирішених дефектів за чотири чотиритижневі періоди. Перші два періоди відповідають 28-денним спринтам методології Scrum. Наступні два періоди відображають безперервне виконання за методологією Kanban. Команда працює сім днів на тиждень (для спрощення розрахунків, а не як форма покарання). Темп виконання завдань залишається незмінним упродовж усіх чотирьох періодів – це є штучним допущенням, проте не вносить систематичного зміщення. (До результатів додано незначний випадковий шум з метою підвищення реалістичності отриманих даних.) [20, ст. 76]»

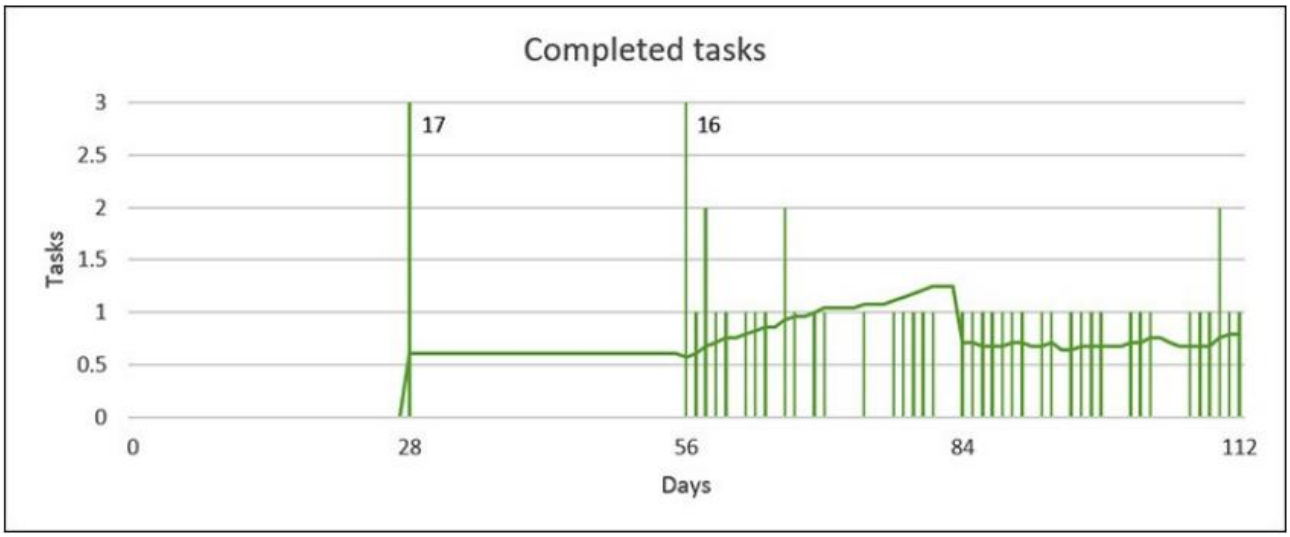


Рис. 1.7. Графік виконання завдань протягом чотирьох чотиритижневих періодів з накладеним ковзним середнім [20]

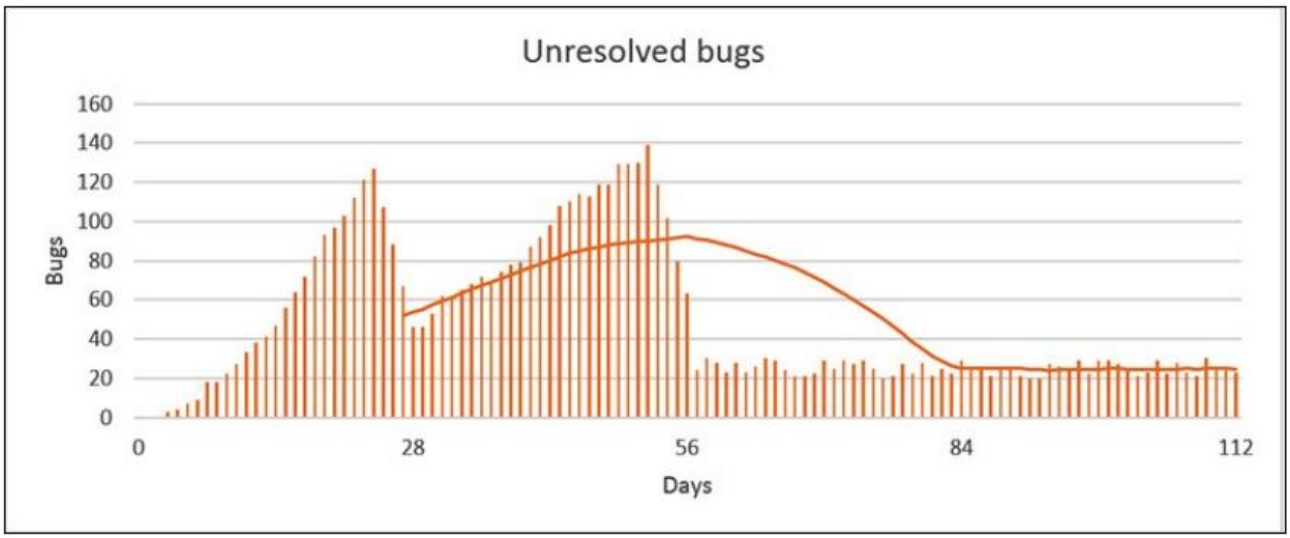


Рис. 1.8. Графік невирішених помилок протягом чотирьох чотиритижневих періодів з накладеним ковзним середнім [20]

Розглядаючи графік виконаних завдань, можна спостерігати, що всі завдання вважаються завершеними в останній день спринту впродовж перших спринтів Scrum. Два стовпці виконаних завдань за ці періоди перевищують висоту графіка приблизно в шість разів. Для зручності перегляду лінії ковзного середнього за чотири тижні, що відображає продуктивність, масштаб було зменшено. Початкове значення цього показника становить приблизно 0,61 завдання на день (17 завдань / 28 днів), після чого зростає до приблизно 1,2 завдання на день у міру того, як результати другого спринту поєднуються з результатами Kanban, а згодом стабілізується на рівні близько 0,76 завдання на день із встановленням рівномірного потоку робіт у рамках Kanban [20].

Механіка Kanban тримається на прозорій дошці, картках, обмеженнях WIP та правилах готовності. Оскільки процес стає безперервним, команда керує лише одним переліком завдань – беклогом [20]. Кожна картка тут є окремим шматочком цінності для користувача. На початковому етапі, який умовно назвемо «Специфікація», великі вимоги зазвичай розбивають на менші підзавдання, під кожне з яких створюють окрему картку. Фахівці самі рухають ці картки, тому стан справ завжди наочний. WIP миттєво підсвічують проблеми, що дозволяє реагувати на них прямо протягом дня, а не чекати ранкового збору [18]. Ба більше, правила перевірки діють на кожному етапі, гарантуючи якість тут і зараз. Що ж

до ролей Scrum Master чи Product Owner – їх можна спокійно зберегти на перших порах, а вже з часом, за потреби, гнучко перерозподілити обов'язки між колегами.

Сказане дає змогу зробити висновок, що перехід від Scrum до Kanban є логічним кроком для команд, які прагнуть позбутися зайвого організаційного шуму та адаптувати процеси під реальний життєвий ритм розробки.

1.5. Огляд існуючих командних Kanban-застосунків

Сьогодні досить важко уявити успішну командну працю без чіткого й наочного відображення поточних завдань. Візуалізація буквально рятує проєкти. Природне прагнення людини структурувати повсякденну рутину призвело до появи цілого плато електронних інструментів, які намагаються перенести класичні картки та дошки у цифровий простір. Проте, аналізуючи сучасний стан ринку програмного забезпечення, автор стикається із досить парадоксальною тенденцією. Провідні платформи, які колись починалися як прості та зрозумілі помічники, поступово перетворилися на перевантажені комбайни. Вони прагнуть вирішити всі проблеми одночасно. Але чи завжди такий надлишок інструментів іде на користь реальній продуктивності? Це велике питання.

З погляду автора, більшість відомих командних рішень створювалися з огляду на великі корпорації з їхнім суворим обліком, нескінченними звітами та жорсткими графіками. Але реальне життя часто вимагає більшої гнучкості. Для людей творчих професій, представників креативних індустрій чи тих же студентів, чия діяльність об'єктивно не має фіксованого розкладу, такі гігантські системи стають занадто заплутаними та важкими для щоденного використання.

У власному дипломному дослідженні автор окремо наголошує на тому, що достаток різноманітних віджетів, незрозумілих символів та прихованих налаштувань створює серйозний деструктивний шум. Особливо це критично для користувачів із синдромом дефіциту уваги та гіперактивності. Таким людям життєво необхідний спокійний, чистий інтерфейс, де кожна кнопка виконує виключно

зрозумілу дію без зайвого когнітивного навантаження. Натомість популярні командні застосунки часто ховають саму суть методології Kanban під купою супутніх функцій, які пересічному користувачеві взагалі не потрібні.

Для формування об'єктивного підходу та пошуку оптимальних орієнтирів, постає необхідність детально розібрати ключових гравців, які зараз домінують на ринку. Кожен із цих застосунків пропонує власну унікальну філософію організації потоку задач. Хтось намагається брати максимальною простотою, інші – залізобетонною інтеграцією з іншими сервісами чи глибинною аналітикою для менеджерів. Сказане дає змогу зробити висновок, що критичний аналіз існуючого командного інструментарію є обов'язковим теоретичним підґрунтям, яке дозволить обґрунтувати доцільність та практичну цінність створення нового, більш людиноорієнтованого мобільного рішення.

1.5.1. Досвід автора роботи з Jira

Проходячи переддипломну практику в підприємстві АТ «Таскомбанк», я особисто стикнувся із таким інструментом управління та відстежування виконання завдань, як Jira. Хоча мій досвід роботи з цим застосунком та компанією не є всеохоплючим та має свої прогалини, але дає потужну базу для аналізу та опису цього додатку, які зі сторони працівника компанії так і програміста і, на решті, розробника власного застосунку, базованого на методології Kanban. Це дає змогу краще описати цей інструмент, врахувати всі переваги та недоліки.

Якщо бути конкретним, наскільки це можливо враховуючи умови та положення конфіденційності банку, опис даного застосунку ведеться з позиції працівника технічної пітримки. Загалом отримані мною знання, досвід та інформація є цінним матеріалом для доповнення відомостей про Jira у загальному доступу в мережі Інтернет та послугує чудовим прикладом в розробці та оформленні мого власного затосунку.

Описуючи застосунок та його особливості саме в цьому банку, використовуєтья вебадаптація написана відділом програмної розробки, це дозволяє точно забезпечувати потреби та максимально покращує організацію банку. Для

уточнення, застосунок базується на оригінальному програмному забезпеченні Atlassian Jira, і не є його копією чи схожим застосунком (див. Додаток А).

Спроба зазирнути "під капот" фронтенду через інструментарій розробника виявила доволі строкату, гібридну еволюцію цієї платформи. З одного боку, маємо цілком сучасний прошарок. Новітні модулі, як-от інтерактивні Kanban-дошки, зібрані на базі React, маршрутизація адекватно обробляється React Router, а за стилізацію відповідає Emotion. Збирається та оптимізується цей код через класичну зв'язку Webpack і Babel. Звучить непогано. Аналітичний блок також вражає: для малювання складних діаграм згорання задач (Burndown) чи візуалізації швидкості (Velocity) розробники вживили потужну бібліотеку D3.js. Проте поруч із цим хайтеком міцно засів старий архітектурний спадок. Досі крутяться важкі модулі на базі Backbone.js та jQuery. Їх доводиться штучно підтримувати маніпуляторами на кшталт jQuery Migrate, суто щоб воно не розсипалося.

І тут виникає закономірне риторичне питання: чи дійсно звичайній людині потрібен весь цей монструозний функціонал для організації власних справ? Відповідь очевидна.

Власне, розбір цієї корпоративної глиби дозволив автору чітко кристалізувати практичну цінність власної кваліфікаційної роботи. Велика Enterprise-система є банально надлишковою. Гори legacy-коду та тисячі мікросервісів створювалися для гігантських корпорацій, а для одинака чи фрілансера такий інтерфейс перетворюється на повільний, перевантажений лабіринт. Через оці застарілі архітектурні вкраплення (той самий Backbone), спроба повноцінно маніпулювати дошкою з екрана мобільного телефону перетворюється на суцільне розчарування. Мобільна адаптивність відверто кульгає. До того ж, фокус уваги single-користувача постійно розмивається прихованими A/B-тестами (через сервіси на кшталт Statsig) та агресивною аналітикою. Графічний інтерфейс корпоративних платформ буквально "тисне" на користувача.

Цей негативний користувацький досвід заклав міцні орієнтири для проєктування власного застосунку. Головний принцип тут – тотальне очищення стеку.

Оскільки рішення розробляється для одиночного використання, важкі інструменти керування контентом безжально відсікаються. Замість гібридного хаосу потрібна єдина, монолітно-чиста архітектура.

Можна було б, звісно, піти шляхом використання вебфреймворків, орієнтуючись на React-підхід сучасної Jira. Проте автор вважає, що написання застосунку саме під мобільні телефони є найкращим рішенням для повсякденного використання. Тому основною операційною системою для розробки обрано Android, а мовою, яка забезпечує виконання основної логіки програми, виступає Kotlin. Вона зберігає та логічно доповнює інструментарні рішення Java, створюючи чудову базу для кодингу. Основні принципи такої програми – простота та надійність.

Щодо візуалізації метрик, то ідея з використанням D3.js у корпоративному сегменті є вдалою, тому, як майбутнє покращення, планується реалізувати схожу, але максимально полегшену мобільну генерацію графіків для відстеження особистої продуктивності (наприклад, Cycle Time чи Lead Time персональної дошки). Сказане дає змогу зробити висновок, що відмова від обтяжливого бекенду корпоративних систем на користь лаконічного, вузькоспрямованого мобільного інструменту дозволить створити продукт, який допомагатиме в роботі, а не вимагатиме нескінченного часу на власне налаштування.

1.5.2. Trello – найпопулярніший застосунок

Розглядаючи сучасний ринок інструментів для управління проєктами, неможливо оминути Trello як один із найпопулярніших сервісів візуалізації. В основі його архітектури лежить проста й зрозуміла кожній людині метафора – дошки, списки та картки, які разом утворюють гнучку трирівневу модель. Дошка відображає загальний простір проєкту, списки маркують конкретні етапи виконання завдань, а картки слугують гнучкими носіями інформації про окремі кроки чи ідеї [28]. Здавалося б, геніально у своїй простоті. Але чи так це насправді, коли користувач залишається сам на сам із програмою?

Кожна окрема картка в Trello здатна вмістити колосальний обсяг супутніх даних [28]. Автор підкреслює, що офіційне керівництво пропонує користувачам безмежні можливості: додавати розгалужені чек-листи, кольорові мітки, кінцеві терміни, вкладені файли, а також налаштовувати внутрішню автоматизацію дій за допомогою робота-помічника Butler. Така глибока кастомізація перетворює платформу на універсальний цифровий конструктор. Проте саме тут прихована пастка. Універсальність часто породжує хаос. Намагання адаптувати дошку під складні процеси призводить до того, що робочий простір обростає десятками маркерів, що банально замилює око. Це відволікає.

На основі аналізу офіційних матеріалів Trello [28], автор вважає, що для персонального планування, особливо в рамках розробки вузькоспрямованого мобільного застосунку для одного користувача, цей інструмент є занадто громіздким. Для чого одиночному розробнику чи студенту складна логіка взаємодії між командами або системи автоматизації роботи Butler? Це лише створює надлишковий когнітивний шум. У власному дипломному проєкті автора пропонується кардинально інший вектор – тотальне спрощення інтерфейсу та відсікання інформаційного баласту. Замість безмежної свободи створення хаотичних списків, як це реалізовано в Trello, власне мобільне рішення на Kotlin пропонує три базові стани завдання: планування, в роботі та готово.

Окремої уваги заслуговує екосистема додаткових модулів, відома як Power-Ups, що дозволяє інтегрувати до дошки сторонні календарі, аналітичні звіти та сторонні хмарні сервіси [28]. Безперечно, для великих команд це зручно. Але з погляду мобільної адаптивності, велика кількість підключених розширень перетворює мобільний екран на захаращену конструкцію. Користувачеві доводиться витрачати забагато часу на банальний скролінг екрана в пошуках потрібної картки.

Сказане дає змогу зробити висновок, що хоча Trello й демонструє чудовий приклад класичної візуалізації методології Kanban [28], його сучасний фокус чітко змістився у бік корпоративного сектору та багатокористувацьких команд. Створення ж легкого, персонального мобільного продукту для Android, повністю

позбавленого цих маркетингових "наворотів", дозволить повернути методології її початкову сутність – чисту та безперешкодну концентрацію людини на поточному потоці її власних задач.

1.5.3. ClickUp – багатофункціональний застосунок

Наступним вагомим об'єктом аналізу є платформа ClickUp, яка позиціонує себе на ринку під доволі амбітним гаслом – «один застосунок, щоб замінити всі інші» [22]. Вивчаючи офіційні навчальні матеріали ClickUp University, автор зазначає, що архітектура цієї системи спирається на надзвичайно розгалужену, багатоступеневу ієрархію, яка послідовно охоплює робочі простори (Workspaces), простори (Spaces), папки (Folders), списки (Lists) і лише в самому кінці – окремі завдання (Tasks) [22]. Така собі спроба запхнути весь офіс в один екран, де користувач може налаштовувати сотні параметрів під власні забаганки. Для великих корпоративних структур, де працюють тисячі людей з різними рівнями доступу, такий підхід, можливо, є виправданим. Але що відбувається, коли в систему заходить звичайна людина? Вона банально тоне у цьому морі функціоналу.

Платформа пропонує понад п'ятнадцять різних варіантів відображення даних, серед яких, окрім класичної Kanban-дошки (Board View), є списки, календарі, діаграми Ганта та навіть інтерактивні карти думок (Mind Maps) [\34]. Кожне завдання всередині дошки може обростати безліччю кастомних полів, тегами, підзавданнями кількох рівнів вкладеності та автоматизованими сценаріями [22]. Проте, за великим рахунком, за всією цією різноманітністю губиться сама суть наочності. Екран перетворюється на захаращену конструкцію, де через купу віджетів та різнокольорових індикаторів стає важко розгледіти реальний поступ роботи. Чи сприяє це концентрації? Навряд чи.

На думку автора, такий максималізм у дизайні та структурі інтерфейсу є головним деструктивним чинником для індивідуального використання. Персональний мобільний застосунок має бути очищений від будь-якого когнітивного шуму. Складна ієрархія ClickUp із нескінченними папками та списками [22] лише ускладнює життя single-користувачеві, особливо якщо йдеться про

студентів чи представників креативних професій, які виконують завдання без жорстких графіків. Власне мобільне рішення автора, свідомо відмовляється від цих маркетингових надлишків.

Окремо слід згадати про мобільну адаптацію аналізованої платформи. Спроба перенести такий колосальний обсяг функцій, включно з внутрішніми чатами, документами та аналітичними панелями інструментів, на екран смартфона призводить до суттєвого уповільнення роботи інтерфейсу [22]. Елементи стають занадто дрібними, а навігація вимагає великої кількості дій від користувача, що перетворює щоденне планування на рутину.

Сказане дає змогу зробити висновок, що ClickUp є класичним прикладом перевантаженої системи, яка через прагнення до всеохопності втратила початкову легкість та інтуїтивність методології Kanban. Саме тому розробка простого, орієнтованого на одну людину мобільного інструменту без зайвого інформаційного баласту є гостро актуальним завданням для сучасного ринку персональних застосунків.

1.5.4. Worksection – український аналог

Досліджуючи вітчизняний сегмент ринку програмного забезпечення, автор окремо зупинився на системі Worksection, яка пропонує досить специфічний погляд на організацію робочого простору. Головний наголос тут робиться на чітку ієрархію та можливість максимальної деталізації поточних завдань. Основою планування є класична деревоподібна структура, де кожна велика задача може розгалужуватися на безліч дрібніших елементів. Для наочності розробники додали функціонал Kanban-дошки, яка дозволяє бачити рух цих завдань по горизонтальних етапах. На екрані це відображається у вигляді стовпчиків, де картки перетягуються залежно від їхнього поточного стану [17]. Зручно? З одного боку, так. Проте система занадто сильно прив'язана до концепції колективної підзвітності та жорсткого контролю термінів.

У налаштуваннях кожної картки закладено чимало управлінських інструментів. Тут можна виставляти пріоритети, визначати дедлайни, прикріплювати

документи та запускати вбудований таймер для обліку кожної хвилини [17]. Платформа буквально змушує грати за суворими правилами, де для екстрених випадків навіть передбачено статус «горить», що сигналізує про критичне наближення термінів. Більше того, офіційне керівництво наголошує, що користувачі мають можливість створювати «підзадачі будь-якого рівня вкладеності», перетворюючи звичайний список справ на складне багаторівневе дерево [17]. Але для чого людині, яка працює наодинці з собою, стільки рівнів контролю? Велике питання. Це перетворює творчість на суху бюрократію.

Аналізуючи функціонал Worksection через призму власного дослідження, автор вважає, що такий підхід є кардинально хибним для індивідуального розробника чи представника креативних індустрій. Навіщо одиночному користувачеві складна логіка підзадач чи таймер, який постійно цокає над вухом і створює зайву тривожність? Життєво необхідним є спокійний, чистий простір. Саме тому у власному мобільному застосунку для Android, автор свідомо відмовився від заплутаної ієрархії, де кожна кнопка виконує виключно зрозумілу дію без прихованих підтекстів.

Тайм-трекінг та фінансові звіти. Корисні речі для великого бізнесу, але цілковитий баласт для фрілансера. Щоденне планування у Worksection вимагає від користувача постійного заповнення купи полів, кліків по дрібних елементах інтерфейсу та моніторингу дедлайнів, що на маленькому екрані смартфона стає справжнім випробуванням для нервів. Мобільна версія просто не встигає за думкою людини через перевантаженість логіки бекенду.

Сказане дає змогу зробити висновок, що хоча Worksection і є потужним інструментом для командних агентств, для індивідуального користувача він є занадто громіздким. Це ще раз підтверджує актуальність розробки чистого, людиноорієнтованого мобільного Kanban-застосунку, повністю звільненого від інформаційного шуму.

1.5.5. Узагальнена оцінка та порівняння настільних аналогів системи

Таблиця 1.1

Порівняння стаціонарних застосунків типу Kanban

Критерій порівняння	Jira (Atlassian)	Trello (Atlassian)	ClickUp	Worksection
Основне призначення	Управління складними процесами розробки ПЗ (Agile / Scrum / Kanban).	Простий візуальний трекер завдань для невеликих команд та особистого користування.	Універсальна платформа «все-в-одному» для підвищення продуктивності організації.	Українська SaaS-система для бізнесу, клієнтського сервісу та командної роботи.
Складність інтерфейсу	Висока. Потребує тривалого налаштування та адміністрування.	Низька. Інтуїтивно зрозуміла з перших хвилин роботи.	Середня. Величезна кількість функцій може перевантажувати екран.	Низька / Середня. Лаконічний та чіткий інтерфейс.
Реалізація Kanban	Потужні дошки із жорстким контролем WIP-лімітів та інтеграцією зі сховищами коду.	Класичний Kanban (картки та списки), що є основою всієї системи.	Гнучкі дошки, які можна комбінувати з таблицями, списками та ментальними картами.	Наочні Kanban-дошки з акцентом на дедлайни, підзавдання та виконавців.
Метрики ефективності та аналітика	Максимальна. Графіки згоряння (Burndown), контрольні карти, Cycle Time, Lead Time.	Базова. Тільки через підключення сторонніх плагінів (Power-Ups).	Просунута. Кастомізовані віджети, облік часу, аналітика спринтів на дашбордах.	Орієнтована на бізнес. Таймтрекінг, звіти про завантаженість команди та фінанси.
Кастомізація процесів	Глибоке налаштування власних статусів, типів завдань (Epic, Story, Bug) та воркфлоу.	Обмежена. Зміна фонів, додавання міток та користувацьких полів.	Дуже висока. Дозволяє налаштувати під себе практично будь-який елемент системи.	Оптимальна. Налаштування статусів під специфіку компанії без зайвого ускладнення.
Особливості мобільного додатка	Орієнтований на моніторинг завдань розробки; адмініструвати з телефона незручно.	Чудово оптимізований, швидкий, повністю повторює логіку вебверсії.	Має широкий функціонал, але через це інтерфейс на мобільних екранах буває перевантаженим.	Зручний та легкий клієнт для оперативного зв'язку та контролю дедлайнів на ходу.
Походження та локалізація	США / Австралія. Повна локалізація, але підтримка орієнтована на глобальний ринок.	США. Повна локалізація, інтерфейс перекладено багатьма мовами.	США. Інтерфейс переважно англомовний, локалізація інших мов впроваджується.	Україна. Повна рідна локалізація, швидка українська техпідтримка, сервери в ЄС.

Проведений автором аналіз ринку програмного забезпечення виявив специфічну тенденцію. Більшість популярних інструментів управління проєктами, таких як Jira чи той самий ClickUp, розроблялися з огляду на великі команди. Вони потужні, проте для однієї людини їхній інтерфейс часто стає справжнім випробуванням через надмірну захаращеність налаштуваннями. Корпоративний сегмент диктує свої правила, де за бортом залишаються потреба звичайного розробника-одинака в елементарному спокої. Постає цілком логічне питання: чи варто взагалі витрачати години на конфігурацію спринтів у Jira, якщо потрібно просто візуалізувати свій щоденний поступ?

З іншого боку, існують Trello та вітчизняний сервіс Worksection, які позиціонуються як значно приязніші до користувача платформи. Проте проблема, криється в деталях. Аналізуючи ці інструменти під час підготовки кваліфікаційної роботи, автор помітив цікаву річ: навіть умовно прості дошки Trello зараз активно підштовхують до командної взаємодії, створення спільних робочих просторів та постійного онлайн-синхрону. Це відволікає. Що стосується системи Worksection, то вона взагалі заточена під фінансову звітність, тайм-трекінг бізнес-процесів та роботу з клієнтами, що для персонального трекера є повністю надлишковим і навіть шкідливим вантажем. Щиро кажучи, розробнику-одинаку, який хоче просто розкласти свої поточні завдання «по поличках», доводиться продиратися крізь купу непотрібних вікон, нагадувань та корпоративних сповіщень.

Тут і криється головна суперечність сучасних застосунків. Великі екрани настільних систем дозволяють розробникам ClickUp «вмістити все», але при спробі перенести цей досвід на мобільні рейки інтерфейс перетворюється на кашу. Мобільність вимагає лаконічності.

Описане дає змогу зробити висновок, що на ринку існує чітка ніша для легкого, мобільно-орієнтованого Kanban-застосунку. Він має забезпечити автономне керування робочими процесами для одного користувача.

1.6. Огляд Android рішень

У цьому підрозділі здійснено аналітичний огляд мобільних застосунків. Попри те, що більшість із розглянутих продуктів є адаптацією десктопних рішень, вони слугують показовими прикладами проектування інтерфейсу для мобільних платформ.

1.6.1. KanbanApp, як локальне мінімалістичне рішення

Досліджуючи численні програмні рішення на ринку, автор натрапив на продукт, який, власне, і став головним ідейним рушієм для власної розробки. Йдеться про KanbanApp. На тлі перевантажених корпоративних гігантів, що намагаються осягнути неосяжне, цей застосунок виглядає як справжній ковток свіжого повітря. Основна ідея полягає у забезпеченні максимально простого інструменту для візуального керування задачами з акцентом на автономність і швидкий доступ [24].

Чому він так чіпляє? Бо сповідує філософію радикального мінімалізму. Автор вважає, що його беззаперечна перевага – це повна автономність і приватність. Жодних хмарних підписок, обов'язкових акаунтів чи нескінченних синхронізацій, які лише гальмують роботу [24]. Застосунок не потребує реєстрації. Програма просто працює, а дані зберігаються локально на пристрої користувача, що забезпечує незалежність від серверної інфраструктури. У власному проєкті автор повністю запозичив цей підхід, адже відмова від реєстрації через електронну пошту дозволить користувачу відразу використовувати функціонал та загалом уникнути зайвого акаунту.

Однак, попри всю геніальність такого легкого Kanban-рішення, на практиці вилізла одна суттєва хиба. Інтерфейс. Він горизонтальний. Чи зручно це на вузькому екрані смартфона? Аж ніяк. Спроба перетягувати картки нескінченними свайпами вліво-вправо часто перетворюється на справжнє знущання над пальцями, особливо якщо колонок більше ніж три. Це просто неприродно для сучасної людини, яка звикла гортати стрічку новин зверху вниз.

Саме ця ергономічна нестиковка підштовхнула до ключового архітектурного зсуву в кваліфікаційній роботі. Надихнувшись легкістю KanbanApp, автор переосмислив його логіку та розробив власний мобільний застосунок. Кардинальна відмінність – вертикальне компонування простору. Запропоноване рішення об'єднує формат класичного to-do листа та Kanban. Замість того щоб змушувати людину губитися в нескінченних горизонтальних свайпах, користувач бачить компактний перелік, де етапи логічно розташовані зверху вниз. Така структура гарантує, що графічна складова програми не "тиснутиме" на користувача.

Отже, потрібно сказати, що KanbanApp став тим самим ідеальним прототипом, який наочно довів життєздатність локальних, незалежних від серверів рішень. Але лише шляхом розвороту дошки на 90 градусів – від горизонталі до вертикалі, вдалося досягти справжнього тактильного комфорту для мобільного формату, який придатний не лише за робочих обставин, а й для повсякденного життя. На закінчення зазначу, що для людей, які займаються креативними справами або IT-напрямком , а особливо для користувачів із синдромом дефіциту уваги та гіперактивності , такий гібридний вертикальний підхід дозволяє "компенсувати" неуважність без зайвого стресу. Сказане дає змогу зробити висновок, що критичне запозичення мінімалістичних ідей з їхньою подальшою ергономічною переробкою під реальні потреби людини створює значно цінніший продукт, ніж сліпе копіювання існуючих стандартів.

1.6.2. Brisqi – класичне хмарне рішення

Аналізуючи сучасні програмні рішення для індивідуального планування, варто детально розглянути застосунок Brisqi. Він позиціонується в каталозі Play Market як інструмент із базовою філософією offline-first. Це означає майже повну стартову автономність. Усі поточні дані записуються безпосередньо у локальну базу на самому пристрої користувача, а не зберігаються виключно на віддалених серверах, очікуючи стабільного підключення [21]. Розробники відверто заявляють, що їхній продукт створений суто для одинаків – незалежних авторів, студентів та фрілансерів. З нього свідомо прибрали всі громіздкі функції для

колективної взаємодії, які зазвичай перевантажують інтерфейс сучасних корпоративних систем. Саме такий вектор тотального мінімалізму та відсутності надлишкових інструментів спільного доступу лежить в основі мого власного кваліфікаційного проєкту, де головним орієнтиром є створення надійного рішення для візуалізації одиничних робочих процесів, графічна складова якого абсолютно не тиснутиме на людину.

Візуальний простір програми дійсно позбавлений зайвого шуму. Користувачам пропонують зручну підтримку форматування тексту, використання кольорових маркерів та спеціальний режим фокусування на конкретному списку. Звучить цілком практично. Але існує одна суттєва технічна деталь, що стосується безпосередньо мобільного клієнта. В офіційному описі сторінки додатка вказано: мобільний Brisqi функціонує радше як доповнення до основної десктопної програми, і для його повноцінного використання спочатку необхідно створити обліковий запис на сайті розробника. Автор вважає такий обов'язковий етап початкової авторизації через сторонні ресурси невиправданим бар'єром, тому у власній розробці взагалі відкинув потребу в реєстрації чи прив'язці до електронної пошти. Встановив і одразу працюєш.

Загалом логіка переміщення карток у Brisqi реалізована досить якісно. Вона дозволяє тримати концентрацію на поточних завданнях без постійного відволікання на другорядні елементи [21]. Відсутність складних багаторівневих меню та нав'язливих метрик робить його доволі комфортним у повсякденному використанні. Цей підхід до зменшення інформаційного навантаження є критично важливим для осіб із синдромом дефіциту уваги та гіперактивності, яким для підтримки робочого ритму життєво необхідний максимально чистий простір.

Проте, на мою думку, для досягнення абсолютної наочності саму архітектуру дошки можна спростити ще більше, що власне і було реалізовано у моєму застосунку, де канбан обмежується лише трьома базовими статусами: планування, в роботі, готово. Такий базовий, але інтуїтивно зрозумілий каркас дозволяє ефективно компенсувати розсіяність і найкраще підходить для творчих проєктів, які об'єктивно не мають чітких часових графіків.

Отже, потрібно сказати, що Brisqi є гідним прикладом системи локального типу, яка цілком обґрунтовано відмовилася від корпоративної складності на користь особистої приватності та швидкодії. На закінчення зазначимо, що попри продуману візуальну складову, залежність його мобільного додатка від попередньої веб-авторизації частково порушує концепцію справжньої автономності портативного пристрою. Сказане дає змогу зробити висновок: сучасний ринок усе ще потребує повністю незалежних, легких мобільних застосунків для одного користувача, де архітектурна простота та безпека локальних даних реалізуються відразу, без жодних компромісів чи вимог щодо попередньої реєстрації.

1.6.3. TickTick – персональні менеджер задач

Розглядаючи еволюцію персональних планувальників, варто звернути увагу на такий інструмент, як TickTick. Цей застосунок, здається, намагається всидіти одразу на двох стільцях. З одного боку, перед нами класичний менеджер списків справ (to-do list). З іншого – цілком серйозна спроба інтегрувати візуалізацію Канбан-дошок у повсякденну рутину звичайного користувача смартфона [27].

Власне кажучи, розробники додали чимало інструментів продуктивності. Тут вам і вбудований таймер Pomodoro для відліку робочих відрізків, і досить детальний трекер звичок, і календарна сітка. Здається, що це ідеальний цифровий застосунок для самоорганізації. Але чи дійсно людині, яка просто хоче впорядкувати свої поточні проєкти, потрібен цей таймер-помідор? Автор вважає, що такий функціональний вінегрет часто створює абсолютно протилежний ефект – замість чіткого фокусування на завданнях, людина починає банально гратися з налаштуваннями, обирати звуки для нагадувань чи безкінечно сортувати теги.

У межах роботи над власним кваліфікаційним проєктом автор ретельно аналізував саме цей гібридний підхід TickTick – поєднання to-do листа та Kanban-дошки. Сама ідея такого об'єднання є надзвичайно вдалою. Проте її реалізація на ринку викликає певні сумніви. Головна мета – створити надійний інструмент, графічна складова якого не буде «тиснути» на користувача. Власне рішення бере

за основу саму ідею гібриду, але безжально відсікає весь нав'язливий інформаційний баласт. Навіщо ускладнювати? Це дозволяє уникнути зайвих акаунтів. Такий мінімалізм є критично важливим для цільової аудиторії: студентів, айтівців та людей, які займаються креативними справами, де немає чітких графіків.

До речі, якщо уважніше придивитися до самого режиму Kanban у TickTick, стає зрозуміло одну річ. Він там працює швидше як другорядна опція відображення, аніж як фундаментальна філософія управління процесом [27]. Картки можна перетягувати. Так. Але на невеликому екрані телефону, коли тасків стає дійсно багато, ця дошка перетворюється на перевантажену зону, де орієнтуватися стає дедалі важче.

Отже, потрібно сказати, що TickTick є показовим прикладом того, як індустрія намагається універсалізувати підходи до управління часом, пропонуючи все й одразу. На закінчення зазначимо, що прагнення задовольнити абсолютно всі потреби користувача в одній програмі неминуче призводить до втрати тієї самої інтуїтивної простоти, яка є основою методу Kanban.

1.6.4. Порівняльний аналіз мобільних застосунків

Таблиця 1.2

Порівняння Android застосунків типу Kanban

Критерій порівняння	KanbanApp	Brisqi	TickTick
1	2	3	4
Основна концепція	Класичний вебсервіс для автоматизації Kanban-процесів, орієнтований на малий бізнес та фрілансерів.	Спеціалізований офлайн-додаток (offline-first) для персонального ведення Kanban-дощок із фокусом на приватність.	Універсальний комбайн-планувальник (To-Do list) з інтегрованою функцією відображення у вигляді дощок.
Характер реалізації Kanban	Традиційний. Строге дотримання методології, управління картками, аналітичні звіти (які часто є надлишковими).	Чистий та локальний. Візуально лаконічні дошки, підтримка тегів та Markdown без стороннього функціонального шуму.	Гібридний. Kanban є лише одним із режимів перегляду звичайного списку завдань; логіка методології дещо розмита.

Продовження таблиці. 1.2

1	2	3	4
Орієнтація на індивідуального користувача	Середня. Попри можливість одиничного-використання, архітектура системи розрахована на майбутнє розширення команди.	Максимальна. Створювався саме як персональний простір розробника чи творця, де немає колективної роботи.	Висока. Ідеальний інструмент для щоденної рутини однієї людини, але не для складних візуальних процесів.
Автономність та робота без мережі	Офлайн-дошка. Дані зберігаються локально на пристрої	Повна автономність. База даних зберігається локально на пристрої, синхронізація є лише опцією.	Переважно хмарна синхронізація; без мережі функціонал суттєво обмежується.
Складність мобільного інтерфейсу	Застарілий мобільний інтерфейс, який відчувається як звичайна адаптація вебсторінки.	Мобільна версія є, але вона дещо програє десктопній через специфіку пакування локальної бази даних.	Зразкова. Мобільний додаток є першочерговим (mobile-first), проте режим дошки на вузьких екранах все одно перетворюється на звичайні списки.

Проаналізувавши ці три системи, автор дійшов цікавого висновку. Кожна з них має критичний недолік, який заважає назвати її ідеальним мобільним Kanban-трекером для однієї людини.

Взяти, до прикладу, KanbanApp. Ну навіщо розробнику-одинаку хмарний інтерфейс із дизайном десятирічної давнини? Це просто дратує. Brisqi у цьому плані виглядає ковтком свіжого повітря – він повністю локальний за замовчуванням і приватний. Але тут постає інша проблема: архітектурно цей софт заточений під настільні комп'ютери, а його мобільна адаптація, щиро кажучи, залишається дещо незграбною. Що ж стосується TickTick, то це чудова програма для списків покупок чи нагадувань про тренування, але Kanban там прикручений «для галочки». Спроба керувати складним життєвим циклом розробки ПЗ через TickTick зазвичай закінчується тим, що користувач повертається до звичайних текстових нотаток.

Отже, на основі сказаного можна зробити висновок, що реальний баланс між чистотою методології Kanban, повною орієнтацією на мобільні екрани та

відсутністю зайвого командно-аналітичного вантажу наразі не досягнутий в жодному з розглянутих аналогів. Саме цей вакуум і має заповнити розроблюваний застосунок.

1.7. Висновки до розділу 1

Системний огляд предметної області разом із ретельним вивченням теоретичного підґрунтя та наявних на ринку програмних рішень дозволив автору окреслити низку важливих положень.

Насамперед, класичні канони методології Kanban, спочатку створені для великих команд, вимагають суттєвого переосмислення при переході на рейки персонального використання. Автор вважає, що обмеження кількості одночасно виконуваних завдань у поєднанні з наочним графічним відображенням робочого потоку – це базовий каркас для зменшення щоденного розумового навантаження. Але чи дають таку можливість популярні гіганти на кшталт Jira, Trello, ClickUp чи Worksection? Проведений аналіз свідчить, що ні. Вони надто громіздкі. Більшість комерційних таск-менеджерів грішать надлишковою архітектурою, орієнтованою суто на колективну взаємодію, та ще й вимагають постійної наявності мережі Інтернет, абсолютно ігноруючи потребу окремого фахівця.

РОЗДІЛ 2. ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ «KANBAN»

Другий розділ роботи присвячено теоретико-прикладному етапу проєктування мобільного застосунку «Kanban», орієнтованого на візуалізацію індивідуальних робочих процесів. У межах цього етапу здійснюється декомпозиція функціональних вимог до програмного продукту, обґрунтування архітектурних рішень та моделювання взаємодії внутрішніх компонентів системи. Створення чіткої проєктної специфікації є необхідною передумовою для подальшої інженерної реалізації застосунку та забезпечення його ергономічності.

2.1. Визначення вимог до мобільного застосунку

Формування вимог до програмного забезпечення є базовим етапом, який визначає архітектурні межі проєкту та критерії успішності його майбутньої інженерної реалізації. У межах цього підрозділу здійснено системний аналіз та класифікацію умов, яким має відповідати мобільний застосунок для візуалізації індивідуального Kanban-процесу. Систематизація цих параметрів дозволяє чітко відокремити користувацькі потреби від технічних обмежень платформи.

2.1.1. Функціональні вимоги

Формулюючи базовий перелік технічних умов, автор спирався на фундаментальні засади візуалізації потоків, викладені в офіційних настановах з методології Kanban, де основним аспектом виступає унаочнення поточних завдань [26]. Програма має забезпечувати безперешкодний повний цикл керування картками, які уособлюють окремі робочі одиниці. Чи доцільно впроваджувати складні матриці обмежень? Для колективної роботи – так, проте для індивідуального планування, надмірний контроль лише шкодить. Застосунок має функціонувати за принципом максимальної ергономічності. Кожна картка повинна містити текстове поле для теми та можливість фіксації додаткових підпунктів, виконання яких користувач може відмічати окремо.

Наступна вимога стосується безпосередньо структуризації самого простору дошки. Традиційний підхід передбачає гнучке управління етапами проходження завдань від початкової ідеї до завершення [20]. Проте комерційні інструменти часто зловживають кількістю підменю. На противагу цьому, у розроблюваному застосунку, використовується стартове меню з вибором проєктів, де перехід до конкретного відкриває власне дошку з трьома початковими статусами: планування, в роботі, готово. Це створює стійкий каркас. Ніяких прихованих підтекстів. Сказане дає змогу допустити, що чітке обмеження робочих статусів дозволяє людині ефективно компенсувати розсіяність і тримати фокус, особливо у творчих чи мистецьких сферах, які не мають жорстких часових графіків.

Оскільки надмірний і жорсткий контроль в індивідуальному плануванні може викликати психологічний спротив, у системі реалізовано концепцію «м'якого лімітування» (Soft WIP-limit) для фази активної роботи. Програма не блокує дії користувача при перевищенні ліміту, проте сигналізує про перевантаження через динамічну зміну кольору шапки колонки на попереджувальний та виведення інформуючогоповідомлення. Це дозволяє людині ефективно компенсувати розсіяність і тримати фокус, залишаючи за собою фінальне рішення.

Для найбільш ефективного використання обмеженого простору мобільного екрану втілено механіку «акордеон» можливість згортання/розгортання кожної колонки окремо. За замовчуванням усі робочі етапи представлені у розгорнутому вигляді, що дозволяє користувачеві відразу бачити загальну картину. Це зручно на старті. Однак у процесі безпосереднього виконання поточних завдань потреба постійно тримати перед очима абсолютно всі інформаційні блоки одночасно просто відпадає сама собою, оскільки надлишок елементів починає лише розпорошувати увагу користувача. Саме в такий момент стає у пригоді функція під назвою «Згорнути статус». При її активації на екрані залишається виключно заголовок відповідної колонки. Важливою функціональною вимогою є персистентність цього процесу: стан екрана (які колонки згорнуті, а які відкриті) повинен автоматично зберігатися у локальних налаштуваннях проєкту і відновлюватися при наступному запуску застосунку. Окрему увагу приділено специфіці

фінального етапу «Готово». Навіщо це потрібно? Річ у тім, що під час згортання цього статусу система додатково приховує кнопку безповоротного видалення завершених задач, що є свідомим кроком для запобігання випадковим натисканням через звичайну людську необережність під час гортання екрана.

Окрему увагу варто приділити архітектурі даних та взаємодії користувача з інтерфейсом мобільного додатка, що детально аналізується у вітчизняних дослідженнях щодо проектування програмного забезпечення. Програма має забезпечувати миттєвий відгук на дії користувача без необхідності постійного звернення до мережі. Тут автор вважає за необхідне реалізувати філософію повної автономності, коли реєстрація через електронну пошту взагалі не є обов'язковою, що дозволяє уникнути створення зайвих акаунтів. Дані повинні надійно зберігатися локально на пристрої. Просто і зрозуміло.

Отже, потрібно сказати, що архітектура функціональних вимог спрямована на подолання надлишковості. На закінчення варто зазначити, що кожна функція повинна виконувати виключно ту дію, яка безпосередньо зображена або описана на кнопці, що її активує, без потенційно незрозумілих символів чи заплутаних переходів.

2.1.2. Нефункціональні вимоги

Окреслюючи коло нефункціональних параметрів, автор першочергово виділяє вимоги до ергономічності та інтерфейсу користувача, оскільки саме вони визначають психологічний комфорт під час безпосередньої взаємодії з програмою [1]. Програмний продукт має володіти високим рівнем інтуїтивності. Візуальне середовище не повинно перевантажувати сприйняття чи чинити зайвий тиск на людину. Це критично. Для забезпечення цієї умови кожна функція має виконувати виключно ту дію, яка зображена на кнопці, що її активує, без використання потенційно незрозумілих символів чи заплутаних підменю.

На думку автора, такий підхід є життєво необхідним для цільової аудиторії проєкту, до якої належать студенти, розробники та представники креативних індустрій, а особливо особи із СДУГ, для яких надлишковий інформаційний шум

комерційних аналогів стає серйозним тригером у роботі. Графічна складова розроблюваного застосунку має допомагати тримати фокус, мінімізувати відволікання та забезпечувати тактильну плавність, в тому числі завдяки подійно-орієнтованому Drag & Drop механізму з інтерактивним підсвічуванням активних зон екрана.

Наступний важливий пласт – вимоги до конфіденційності, безпеки та надійності збереження інформації [1]. Мобільний застосунок будується на засадах повної автономності користувача. Програма не повинна вимагати обов’язкової реєстрації через електронну пошту чи прив'язки до сторонніх інтернет-сервісів. Весь масив створених даних, включаючи назви проєктів, карток та супутніх підпунктів, має записуватися і зберігатися виключно локально в пам'яті мобільного пристрою [6]. Такий підхід повністю нівелює ризики витоку персональних записів через серверні збої або відсутність стабільного мережевого підключення.

Технічні критерії швидкодії вимагають від системи миттєвого відклику на дії користувача. Час реакції інтерфейсу на перемикання між екранами чи зміну статусів завдань не повинен перевищувати часток секунди (до 100–150 мс) [6]. Програма має стабільно функціонувати на смартфонах із різною діагоналлю екрана, коректно адаптуючи вертикальне компонування чотирьох можливих етапів: планування, в роботі, перевірка, готово. Навіть у разі раптового закриття додатка чи повної розрядки акумулятора пристрою, поточний стан Kanban-дошки повинен автоматично фіксуватися в базі даних без втрати останніх внесених змін.

Отже, нефункціональні вимоги до розроблюваного застосунку повністю підпорядковані ідеї створення спокійного, безпечного та передбачуваного цифрового простору для однієї людини. На закінчення зазначимо, що відмова від складної хмарної архітектури на користь локальної взаємодії дозволяє досягти високого рівня швидкодії та тактильного комфорту. Лаконічні технічні стандарти, орієнтовані на реальні людські потреби, закладають надійний фундамент для успішного впровадження програмного продукту в повсякденне та професійне життя.

2.1.3. Технічні вимоги

Вибір технологічного стеку для реалізації мобільного рішення обумовлений необхідністю досягнення високої швидкодії та стабільності роботи системи на кінцевих пристроях користувачів. Основним середовищем розробки визначено платформу Android, що дозволяє спиратися на офіційні інструменти проектування та розширену документацію для розробників [19]. Конфігурація та автоматизація збірки проєкту реалізується за допомогою сучасного інструментарію Gradle Kotlin DSL, що забезпечує строгу типізацію та підвищує надійність керування залежностями на етапі компіляції.

Які саме технологічні інструменти здатні забезпечити таку безкомпромісну швидкодію та автономність? Для забезпечення виконання базової логіки застосунку автор вважає за доцільне використання мови програмування Kotlin, яка на сучасному етапі є офіційним стандартом для створення Android-програм [13]. Вона не просто зберігає і суттєво доповнює інструментальні підходи, притаманні мові Java, а й надає розробнику набагато безпечніший та лаконічніший синтаксис. Це мінімізує обсяг шаблонного коду та захищає систему від критичних помилок, пов'язаних із порожніми посиланнями [5]. Без зайвих ускладнень. Скорочення кодової бази безпосередньо впливає на швидкість компіляції та загальну легкість самого цифрового продукту, що є критично важливим для портативних пристроїв.

Програмне рішення розробляється з чітким фокусом на архітектуру offline-first. Що це означає на практиці? Будь-яка складна серверна архітектура чи постійна синхронізація з хмарними базами даних тут просто відсутні, через що, власне, і забезпечується миттєвий запуск програмного інтерфейсу. Локальна підсистема збереження даних має бути інтегрована безпосередньо в мобільний клієнт. Для забезпечення повної автономності мобільного застосунку критично важливим є вибір оптимального інструменту локального зберігання інформації. Оцінюючи архітектуру майбутнього рішення, автор свідомо відмовився від використання класичних реляційних баз даних на кшталт SQLite чи сучасних ORM-

бібліотек типу Room. Навіщо ускладнювати систему там, де це не потрібно? Це просто недоречно. Оскільки розроблюваний застосунок орієнтований на одного користувача, обсяги даних тут апріорі будуть невеликими.

Замість важковагових баз даних було обрано підхід із серіалізацією об'єктів у текстовий формат JSON за допомогою бібліотеки GSON від компанії Google. У контексті архітектури збереження цей метод функціонує як інструмент прямого відображення об'єктно-орієнтованих структур у плоскі файли пристрою. Простіше кажучи, вся сукупність сутностей персональної Kanban-дошки, від назв колонок до конкретних карток із завданнями, трансформується в один структурований текстовий рядок.

Механізм роботи такого збереження є максимально прозорим. Під час кожної значущої дії користувача (створення картки, зміна статусу чи перенесення дедлайну) поточний стан об'єктів у пам’яті смартфона переводиться бібліотекою GSON у формат JSON (процес серіалізації) та перезаписується у внутрішній файл програми. При повторному запуску застосунку відбувається зворотний процес – десеріалізація. Текст із файлу зчитується, а GSON миттєво відновлює з нього готові об'єкти для коду.

Побудова графічного інтерфейсу користувача (UI) покладається на класичну технологію Android Views. Оформлення макетів та ресурсів за допомогою XML забезпечує повний контроль над рендерингом, плавність роботи акордеонних механізмів та стабільне функціонування подійно-орієнтованого механізму перетягування елементів, реалізованого через нативний View.OnDragListener. Водночас, на рівні конфігурації у проєкт інтегровано компоненти androidx.compose.material3 (Theme.kt, Type.kt). Незважаючи на те, що активні екрани ініціалізуються через традиційний метод setContentView(), наявність цього інфраструктурного шару є технічною вимогою для забезпечення перспективної масштабованості системи та ізоляції токенів дизайну для майбутньої міграції на Jetpack Compose.

Додатковою технічною вимогою до підсистеми локального резервного копіювання є повна інтеграція з Storage Access Framework (SAF) за допомогою

системних інтентів ACTION_CREATE_DOCUMENT та ACTION_OPEN_DOCUMENT. Це дозволяє додатку безпечно імпортувати та експортувати файли бекапів через потоки введення-виведення (contentResolver), повністю відповідаючи суворим вимогам безпеки Scoped Storage сучасних версій ОС Android.

Таким чином, сформовані технічні вимоги підпорядковані концепції прагматичного проєктування. Поєднання нативного UI на Android Views, мови Kotlin та серіалізації через GSON у SharedPreferences дозволяє отримати продукт із високою швидкодією, низьким споживанням акумулятора та гарантованою безпекою локальних даних користувача.

2.2. Сценарії використання застосунку

Аналіз взаємодії користувача з програмним забезпеченням на етапі моделювання вимагає чіткого структурування послідовності його дій, що традиційно реалізується через побудову сценаріїв використання [10]. Оскільки проектований інструмент орієнтований виключно на індивідуальне планування, складні багаторівневі ролеві матриці, притаманні великим корпоративним системам, тут повністю відсутні [1]. Весь функціонал підпорядкований одному актору – кінцевому користувачу. Як саме людина взаємодіє з інтерфейсом у моменти найвищого розумового навантаження? Для відповіді на це питання автором було розроблено лінійну структуру прецедентів, де кожна функція виконує виключно ту дію, яка чітко написана на самій кнопці, що її активує. Це суттєво мінімізує когнітивний опір інтерфейсу. Просто і зрозуміло.

Перший ключовий сценарій охоплює процес ініціалізації та управління проєктним простором. Користувач запускає мобільний застосунок. Замість звичних для комерційних платформ вікон реєстрації чи вимог прив'язати електронну пошту, система відразу відображає головне стартове меню з вибором усіх раніше створених проєктів. Автор вважає такий крок відсутності авторизаційного бар'єру критично важливим, адже це дозволяє уникнути генерації зайвих

облікових записів і забезпечує миттєвий перехід до роботи. Користувач може створити новий проєкт, вказавши лише його назву, або вибрати існуючий зі списку, що зберігається локально в файлову систему пристрою. Перехід до конкретного проєкту автоматично відкриває власне робочу Kanban-дошку.

Наступний сценарій описує безпосередню роботу з потоком завдань на дошці, що є центральним елементом візуалізації робочих процесів [11]. Користувач ініціює створення картки та, за необхідності, додає внутрішні підпункти. Важливою особливістю інтерфейсу є те, що виконання цих підпунктів можна відмічати окремо прямо всередині картки, що динаміно змінюватиме статус завдання. Менше екранів – менше плутанини. За замовчуванням робочий простір містить лише три статуси завдань: планування, в роботі та готово. Користувач переміщує картки між цими стовпцями в залежності від поточного стану справ. На думку автора, такий прямолінійний підхід чудово підходить для завдань, які не мають чітких часових графіків чи можливості об'єктивної оцінки кількості витраченого часу. Така графічна складова абсолютно не тиснутиме на людину.

Третій сценарій визначає конфігурацію параметрів та адаптацію дошки. Користувач може гнучко налаштовувати логіку відображення інформації під свій поточний психоемоційний стан. Цей прецедент дозволяє активувати «м'який» WIP-ліміт для стовпця In Progress (задання максимальної кількості одночасних завдань), увімкнути або вимкнути механізм Drag & Drop, активувати відображення маркерів пріоритету, а також повністю приховати колонку Review, трансформуючи простір під класичну триетапну схему Kanban.

Отже, потрібно сказати, що розроблені сценарії використання мобільного застосунку повністю усувають технічний шум та оптимізують шлях користувача до виконання завдань. На закінчення зазначимо, що орієнтація на автономність та локальну взаємодію дозволяє досягти передбачуваної поведінки інтерфейсу за будь-яких умов експлуатації [10]. Сказане дає змогу зробити висновок: лаконічність прецедентів використання не лише спрощує архітектуру програмного коду на Kotlin, а й забезпечує високий рівень психологічного комфорту

користувача, створюючи для нього максимально чистий і функціональний цифровий простір.

2.3. Проєктування інтерфейсу користувача

Переходячи до питання безпосередньої розробки візуального каркаса програмного продукту, автор наголошує, що зовнішній вигляд системи та логіка її екранів мають утворювати єдине ціле з психологічними потребами кінцевого споживача. Як спроектувати мобільне середовище так, щоб воно не викликало роздратування вже після перших п’яти хвилин взаємодії? Сучасні тенденції в галузі UX/UI-дизайну вимагають відхилення від перевантажених графічних схем на користь так званого емоційного та заспокійливого проєктування [11]. Для індивідуального планувальника завдань це стає першочерговим завданням. Замість копіювання заплутаних багатосторінкових інтерфейсів великих систем на кшталт Jira або ClickUp, де користувачу доводиться блукати лабіринтами підменю, архітектура екранів створюваного додатка навмисно спрощується. Програма будується на принципі візуального мінімалізму. Це допомагає утримувати фокус.

В основу проектування інтерфейсу покладено концепцію, за якої кожна графічна кнопка виконує виключно ту дію, яка на ній зображена. Жодних подвійних трактувань чи прихованих свайпів, які лише заплутують людину. Графічна сітка жорстко обмежується вертикальним відображенням. Оскільки мобільний пристрій має обмежену площу дисплея, використання фіксованого меню з можливістю бачити весь потік завдань без складних перемикань дозволяє нівелювати когнітивне навантаження. По суті, користувач отримує чисте полотно, де відсутній будь-який зайвий «шум».

Процес проектування взаємодії (UX) свідомо виключає етап створення облікового запису. Менше формальностей. Користувач відразу потрапляє до стартового вікна із переліком власних проєктів, що суттєво підвищує загальну ергономічність і швидкість доступу до інформації. На думку автора, відмова від

складного хмарного компонування елементів та перенесення всієї візуальної логіки на клієнтську сторону мобільного пристрою дозволяє досягти миттєвого відгуку інтерфейсу, що вимірюється частками секунди. Це створює відчуття тотального контролю над процесом, де програма виступає непомітним і надійним помічником у повсякденних і творчих справах.

2.3.1. Концептуальні основи користувацького інтерфейсу

Проєктування графічного інтерфейсу для індивідуального таск-менеджера вимагає радикального перегляду класичних підходів до дизайну Kanban-дошок, які зазвичай перевантажені інструментами для командної комунікації та корпоративного контролю. Автор вважає, що для одного користувача надмірність елементів на екрані смартфона є головним деструктивним чинником, що руйнує фокус і створює відчуття хаосу. Саме тому в основу візуальної архітектури розроблюваного застосунку було покладено принципи цифрового мінімалізму. Це не просто данина сучасним графічним трендам. Насправді, коли площа дисплея обмежена кількома дюймами, кожен зайвий піксель або декоративна плашка починає конкурувати за увагу людини з її ж власними завданнями. Подібне перевантаження є неприпустимим для інструментів персональної продуктивності.

Для глибшого розуміння просторової організації екрана було залучено традиційну японську естетичну концепцію «Ма» (間). Вона трактує порожнечу не як відсутність чого-небудь або брак контенту, а як повноцінний структурний елемент, що надає ваги, форми та змісту об'єктам, розташованим поруч. Простір дихає. У контексті мобільного інтерфейсу Kanban-дошки філософія «Ма» трансформується у свідоме збільшення незаповнених зон навколо карток завдань та відмову від постійного відображення другорядних маркерів. Чи дійсно користувачу потрібно щомиті бачити теги пріоритетів, якщо він просто намагається зорієнтуватися в поточному списку справ на день? Риторичність цього питання очевидна. На етапі практичної побудови інтерфейсу це

підштовхнуло до рішення повністю приховати «зоровий шум» за замовчуванням.

Тут виникає певний парадокс, який часто помічають розробники-початківці на власному досвіді: чим простішим і «чистішим» здається інтерфейс зовні, тим складнішою виявляється його логічна архітектура. Автор припускає, що баланс між візуальним мінімалізмом та функціональною потужністю мобільного застосунку досягається через гнучкість системи налаштувань. Тобто екран залишається розвантаженим доти, доки сам користувач свідомо не захоче активувати специфічні інженерні інструменти, на кшталт лімітів незавершеної роботи (WIP) чи сортування карток за пріоритетністю. Спроба зробити інтерфейс абсолютно порожнім без можливості адаптації під глибокі потреби користувача могла б перетворити програмний продукт на гарну, але марну іграшку, що навряд чи задовольнило б вимоги до реального та функціонального застосунку.

Практичне втілення концепції «Ма» та мінімалізму чітко простежується в організації списків етапів виконання завдань. Замість традиційного горизонтального прокручування великої кількості колонок, яке на вузьких екранах телефонів зазвичай перетворюється на ергономічне жахіття через необхідність постійно гортати туди-сюди, було використано вертикальну модель згортання елементів типу «акордеон». Порожнеча навколо закритих етапів дозволяє людині миттєво оцінити загальний стан справ, не відволікаючись на деталі. Дошка не тиснутиме своїм об'ємом. Текст підзадач і плашки карток взаємодіють між собою виключно через вивірені мікро-відступи. Як результат, порожнє місце слугує природним візуальним роздільником, що дозволило повністю відмовитися від громіздких ліній розмежування чи важких контурних рамок.

Сказане дає змогу зробити висновок, що поєднання японської концепції «Ма» та принципів функціонального мінімалізму дозволяє побудувати інтерфейс, який не виснажує когнітивну сферу користувача під час щоденного планування. На закінчення зазначимо, що такий підхід до проєктування не лише вирішує питання естетичної привабливості мобільного застосунку, але й

безпосередньо покращує досвід взаємодії (UX). Користувач витрачає значно менше часу на орієнтацію в просторі дошки, що є критично важливим для збереження стану фокусу під час індивідуальної роботи.

2.3.2. Макети екранів

Безпосередня графічна архітектура екранних форм є логічним продовженням загальної концепції інтерфейсу, перетворюючи теоретичні засади індивідуального Kanban на конкретні інтерактивні зони. Як розпланувати простір маленького дисплея, щоб не перетворити корисний інструмент на хаотичне нагромадження ліній? Для цього розроблено серію взаємопов'язаних низькодеталізованих макетів, які визначають геометрію та навігаційну логіку елементів додатка [11]. Автор свідомо фокусується на концепції однієї активної зони на екрані, відсікаючи складні тривимірні ефекти чи багаторівневі списки, які часто зустрічаються у великих корпоративних системах планування. Це дозволяє зберегти простір чистим і легким для швидкого візуального сканування.

Першим базовим прототипом виступає екран управління проєктним портфелем. При запуску застосунку, оминаючи всі можливі вікна авторизації та тривалі процеси мережевої синхронізації (дані відразу підтягуються з вбудованої бази пристрою), користувач бачить лаконічний список створених робочих областей. У нижній частині екрану зафіксовано кнопку додавання нового проєкту, яка за своєю формою та текстовим підписом не допускає жодних двозначностей під час натискання. Клікнувши на назву будь-якого проєкту зі списку, система здійснює лінійний перехід до основного вікна – індивідуальної Kanban-дошки. Автор зазначає, що таке лінійне компонування екранів мінімізує кількість кроків, необхідних для початку безпосередньої фіксації ідей, що є архіважливим чинником для збереження концентрації у представників творчих та IT-професій.

Макет самої Kanban-дошки проектується з урахуванням специфіки вертикального утримання мобільного телефона однією рукою [15]. У конфігурації за замовчуванням екран містить три послідовні вертикальні секції-контейнери: планування, в роботі, готово. У разі активації в налаштуваннях опціонального

параметра, у загальну ієрархію макета між етапами в роботі та готово динамічно інтегрується четверта секція – перевірка). Користувач може індивідуально згортати або розгортати кожну секцію в один клік. Оскільки одночасне горизонтальне відображення такої кількості колонок на екрані смартфона призвело б до надмірного стиснення тексту, макет використовує гнучке вертикальне прокручування та систему «акордеон» для швидкого перемикання між статусами в один клік. Самі картки завдань мають спрощену прямокутну геометрію.

Всередині кожної картки, окрім назви та короткого текстового опису, зарезервовано окрему зону для чек-лістів – внутрішніх підпунктів, помітки виконання яких можна відзначати безпосередньо на дошці. Менше відкриттів додаткових вікон. Будь-яке переміщення картки між стовпцями реалізується простим перетягуванням. Це робить процес управління тактильно приємним і передбачуваним.

Отже, потрібно сказати, що архітектура макетів екранів розроблюваного застосунку повністю усуває бар'єри між користувачем та його завданнями. На закінчення зазначимо, що графічне розташування елементів керування орієнтоване на миттєвий відгук системи та виключає будь-який візуальний «шум» [11]. Сказане дає змогу зробити висновок: детально продумана геометрія інтерфейсних форм забезпечує високий рівень автономності та психологічного комфорту, створюючи надійний і спокійний інструмент для ефективної організації індивідуальної праці.

2.3.3. UX/UI концепція

Формування цілісної UX/UI концепції індивідуального планувальника вимагає глибокого розуміння психології сприйняття кольорів та форм, що безпосередньо впливає на емоційний стан людини під час роботи [11]. Як саме колір або форма плашки здатні зменшити рівень щоденного стресу? Автор вважає, що візуальна філософія застосунку має базуватися на заспокійливій, монохромній палітрі з мінімальним використанням агресивних маркерів. Замість кричущих червоних чи жовтих відтінків, які у більшості комерційних планувальників

сигналізують про «прострочені дедлайни» і лише розганяють тривожність, тут застосовано м'які, природні тони. Система працює тихо. Без зайвих подразників. Це створює відчуття такого собі візуального прихистку, де користувач не відчуває зовнішнього тиску від самого інструменту, що, власне кажучи, і є головним для відновлення внутрішнього спокою.

Сучасний дизайн мобільних рішень часто “грішить” надлишком мікроанімацій. Спливаючі вікна, конфеті за виконане завдання, постійне мерехтіння елементів – все це трішки заважке для сприйняття, коли людина намагається зосередитися [15]. Концепція мого проєкту повністю відкидає такі декоративні ефекти. Кожна інтеракція є майже непомітною для ока: наприклад, зміна стану картки при її перетягуванні супроводжується виключно легким затемненням її області, що візуально підтверджує точність захоплення об'єкта. Що стосується текстового оформлення, то вибір шрифтової пари підпорядкований виключно критерію читабельності при різному рівні освітлення мобільного пристрою [5]. Чітка ієрархія заголовків дозволяє миттєво зчитувати суть завдань навіть “на ходу”.

Подібна шрифтова та колірна стриманість розробляється як інструмент компенсації розсіяності уваги, що критично важливо для цільової аудиторії проєкту, зокрема осіб із СДУГ, студентів та ІТ-фахівців. Відсутність динамічного мерехтіння елементів допомагає тривалий час утримувати фокус на головних процесах.

Особливе місце в UI-концепції посідає застосування законів гештальт-психології щодо взаємного розташування об'єктів. Картки завдань візуально сприймаються як єдині смислові блоки завдяки однаковим радіусам заокруглення кутів та м'яким напівтіням, які відокремлюють їх від загального тла дошки. Тут немає різких чорних ліній розмежування. Замість них використовується так зване повітря – вільний простір між блоками, який дозволяє екрану «дихати» та різниця кольору з фоном. Автор наголошує на тому, що відмова від обов'язкової прив'язки карток до жорстких часових шкал зміщує фокус з панічного спостереження за часом на безпосередній контроль самого процесу.

Отже, потрібно сказати, що UX/UI концепція мобільного застосунку виходить далеко за межі звичайного прикрашання інтерфейсу, перетворюючись на повноцінний засіб когнітивної підтримки особистості. На закінчення зазначу, що орієнтація на психоемоційну безпеку та заспокійливі візуальні паттерни робить взаємодію із системою максимально природною та невимушеною [11, 15]. Такий підхід до проєктування інтерфейсу на базі технологій Android Views та Material 3 забезпечує тривале та комфортне використання цифрового інструменту, допомагаючи людині залишатися в стані продуктивного фокусу без ризику швидкого ментального втомлювання.

2.4. Висновки до розділу 2

Потрібно сказати, що теоретичний фундамент, закладений на етапі аналізу предметної області, успішно трансформувався в чітку інженерну модель майбутнього застосунку. Проєктування – це завжди пошук компромісів. Автор переконався в цьому особисто під час розробки архітектурного каркаса. Головна складність полягала в тому, щоб поєднати сувору методологію Kanban із гнучкістю, якої очікує єдиний користувач від мобільного додатка. Але як не перевантажити екран пристрою? Питання проєктування логіки взаємодії виявилося дещо підступнішим, ніж здавалось спочатку.

Обрана архітектурна стратегія розподілу компонентів (зокрема, суворе відокремлення презентаційного шару від логіки збереження) дозволить уникнути заплутаності коду під час програмування додатка. Власне, думка автора тут цілком збігається з канонічними підходами до розробки під Android: застосунок має залишатися легким. Студентський аналіз аналогічних практичних кейсів показує, що надлишкові зв'язки між модулями часто «гальмують» роботу інтерфейсу. Саме тому в проєкт закладено абсолютну автономність.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

Етап практичної реалізації є ключовою стадією інженерного циклу, на якій теоретично обґрунтовані вимоги, структури даних та спроєктовані моделі трансформуються у функціонувальний програмний продукт. Цей розділ присвячений детальному опису процесу безпосереднього розроблення мобільного застосунку для візуалізації персональних робочих процесів на базі офіційної платформи Android SDK мовою програмування Kotlin.

У межах розділу висвітлено особливості декларативної конфігурації системи автоматизації збірки Gradle із використанням скриптів Kotlin DSL, описано програмні рішення presentation- та data-рівнів додатка, а також наведено декомпозицію вихідного коду ключових функціональних блоків: інтерактивної Kanban-дошки з превентивним контролем лімітів незавершеної роботи (WIP Limits) та модуля локальної персистентності даних.

3.1. Вибір технологій для реалізації

Визначення технологічного стеку для створення мобільного застосунку візуалізації персональних робочих процесів є критичним кроком, який зумовлює не лише швидкість розробки, а й архітектурну життєздатність усього проєкту. Перед автором постало непросте питання: яке саме середовище здатне забезпечити безкомпромісну швидкодію інструменту, що працює в режимі реального часу? Беручи до уваги той факт, що сучасний ринок мобільних технологій перенасичений різноманітними універсальними інструментами розробки, які часто перевантажують оперативну пам'ять пристрою зайвими бібліотеками, вибір було зроблено на користь офіційної платформи Android SDK та мови програмування Kotlin [19]. Тут немає місця компромісам, оскільки native-розробка дає безпосередній доступ до апаратних можливостей пристрою без додаткових посередницьких шарів.

3.1.1. Конфігурація застосунку

Управління життєвим циклом проєкту, конфігурацією залежностей та процесом компіляції реалізовано за допомогою сучасної системи збірки Gradle із використанням Kotlin DSL. Це дозволяє забезпечити строгу типізацію на етапі конфігурації та спрощує інтеграцію сторонніх бібліотек.

Лістинг 3.1. Фрагмент конфігураційного файлу збірки:

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
}

android {
    namespace = "com.example.kanban"
    compileSdk = 36

    defaultConfig {
        applicationId = "com.example.kanban"
        minSdk = 24
        targetSdk = 36
        versionCode = 1
        versionName = "1.0"
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }
    kotlinOptions { jvmTarget = "11" }
}

dependencies {
...
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-
android:1.8.1")
    implementation("com.google.code.gson:gson:2.10.1")
}
```

Мова Kotlin виступає ядром кодової бази додатка. Її вибір зумовлений вбудованим механізмом безпеки щодо нульових посилань (Null Safety), що ліквідує критичні помилки типу NullPointerException, а також нативною підтримкою концепції об'єктно-орієнтованого моделювання даних [25].

3.1.2. Загальна структкра даних

З метою побудови чіткої логіки (Domain layer) та унеможливлення переходу дошки в невалідні стани, технологічні етапи (колонки) жорстко типізовано через enum class. Для опису сутностей завдань використано механізм Kotlin Data Classes. Це дозволяє автоматично генерувати необхідний шаблонний код, оптимізуючи кодову базу проєкту.

Лістинг 3.2. Структура даних:

```
data class Task(
    val id: Int,
    val title: String,
    var status: String,
    val subTasks: MutableList<SubTask> = mutableListOf(),
    var priority: Int = 1 // 1=Low, 2=Medium, 3=High
)

data class SubTask(
    val title: String,
    var isDone: Boolean = false
)
```

3.1.3. Система збереження

Для реалізації концепції автономного функціонування застосунку (offline-first) без розгортання важких реляційних баз даних, було спроєктовано легкий рівень локального збереження інформації. Сховище реалізовано на базі системного інструменту SharedPreferences у зв'язці з бібліотекою GSON. Компонент LocalTaskStorage здійснює атомарні транзакції збереження та зчитування даних, трансформуючи об'єктні моделі Kotlin у лінійні JSON-рядки:

Лістинг 3.3. Методи локального збереження:

```
fun saveProjects(projects: List<Project>) {
    val json = gson.toJson(projects)
    sharedPref.edit().putString("projects", json).apply()
}

fun getProjects(): List<Project> {
    val json = sharedPref.getString("projects", null)
    val raw: List<Project> = if (json != null) {
        val type = object : TypeToken<List<Project>>() {}.type
```

```
            gson.fromJson(json, type)
        } else {
            emptyList()
        }
        return migrateProjectsIfNeeded(raw)
    }

    fun saveTasksForProject(projectId: Int, tasks: List<Task>) {
        val activeMap = loadActiveMap()
        activeMap[projectKey(projectId)] = tasks.toMutableList()
        saveActiveMap(activeMap)
    }
```

3.1.4. Загальна структкра даних

Для взаємодії з файловою системою пристрою під час створення резервних копій застосовано Storage Access Framework (SAF). Операції відкриття потоків та читання реалізовані в межах основного життєвого циклу MainActivity:

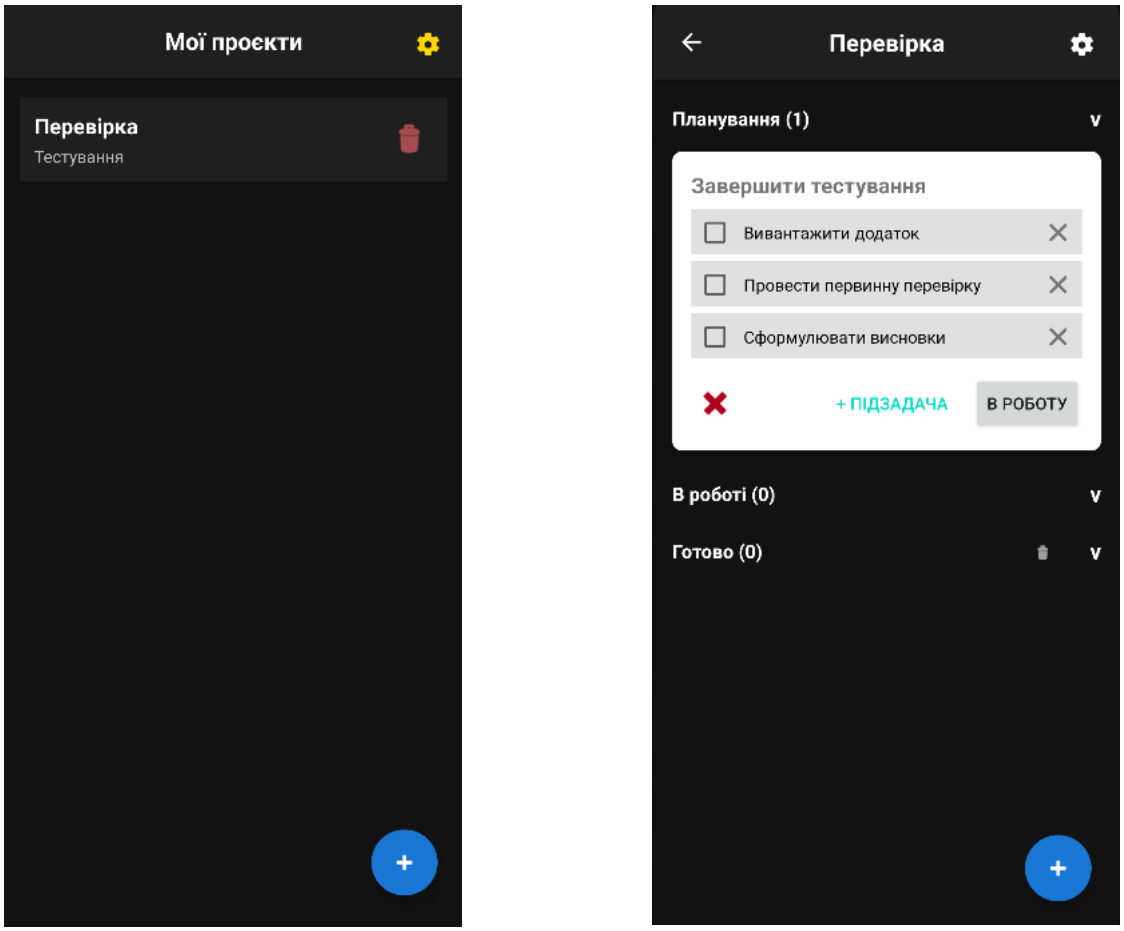
Лістинг 3.4. Метод імпорту та парсингу файлу резервної копії:

```
    private fun handleBackupImportUri(uri: Uri, flags: Int) {
        try {
            contentResolver.takePersistableUriPermission(uri, flags and
Intent.FLAG_GRANT_READ_URI_PERMISSION)
        } catch (_: Exception) {
        }

        val backupJson =
contentResolver.openInputStream(uri)?.bufferedReader().use {
it?.readText() }
        if (backupJson.isNullOrBlank()) {
            showInvalidBackupToast()
            return
        }

        val imported = importBackupPayload(backupJson)
        if (!imported) {
            showInvalidBackupToast()
            return
        }

        loadProjects()
        Toast.makeText(this, R.string.backup_imported,
Toast.LENGTH_SHORT).show()
    }
```

Отже, вибір офіційного інструментарію Android-розробки та мови Kotlin є повністю обґрунтованим з інженерного погляду. Такий підхід дозволяє реалізувати ідею автономного (offline-first) застосунку з максимальним рівнем безпеки та захисту персональних даних. Обраний технологічний стек закладає надійний фундамент для створення швидкого, передбачуваного та візуально стабільного інструменту індивідуального планування, спроможного ефективно функціонувати на більшості сучасних портативних пристроїв.

3.2. Реалізація користувацького інтерфейсу

Практичне втілення інтерфейсних рішень у межах розроблюваного застосунку спирається на використання стандартних, але гнучко налаштованих компонентів Android-платформи, що дозволяє досягти передбачуваної поведінки кожного візуального елемента [19]. Простір мобільного екрана обмежений. Саме тому для реалізації головного вікна (див. рис. 3.1,а) та індивідуальної Kanban-дошки (див. рис. 3.1,б), було застосовано оптимізовані контейнери списків, які забезпечують плавне вертикальне прокручування без надмірного споживання оперативної пам'яті пристрою. Як змусити важкі картки завдань рухатися по екрану абсолютно без затримок? Автор вважає, що ключовим фактором тут є відмова від глибокого вкладення елементів (layout nesting) у розмітці екранних форм, адже кожен додатковий рівень ієрархії змушує процесор пристрою робити подвійну роботу під час перемальовування графіки [15]. Код має бути пласким.



а) б)

Рис. 3.1. Загальний вигляд програми: а) головне меню, б) меню Kanban

3.2.1. Система згортання колонок

Відповідно до концепції цифрового мінімалізму та відмови від ергономічно неефективного горизонтального прокручування колонок на вузьких екранах смартфонів, у застосунку реалізовано вертикально-орієнтовану модель відображення Kanban-дошки. Усі технологічні етапи (Backlog, In Progress, Review, Done) розташовані послідовно один під одним, а доступ до їхнього вмісту керується за допомогою інтерактивного згортання елементів типу «акордеон» (див. рис. 3.2).

Лістинг 3.5. Реалізація механіки «Акордеон»:

```
    private fun setupAccordionHeaders() {
...
        btnClearDone.setOnClickListener {
            showClearDoneConfirmation()
        }
```

```
findViewById<LinearLayout>(R.id.llBacklogHeader).setOnClickListener {
            toggleSection(rvBacklog, tvBacklogChevron, isBacklogExpanded)
{ isBacklogExpanded = it }
        }

findViewById<LinearLayout>(R.id.llInProgressHeader).setOnClickListener {
            toggleSection(rvInProgress, tvInProgressChevron,
isInProgressExpanded) { isInProgressExpanded = it }
        }

findViewById<LinearLayout>(R.id.llReviewHeader).setOnClickListener {
            toggleSection(rvReview, tvReviewChevron, isReviewExpanded) {
isReviewExpanded = it }
        }
        findViewById<LinearLayout>(R.id.llDoneHeader).setOnClickListener
{
            toggleSection(rvDone, tvDoneChevron, isDoneExpanded) {
                isDoneExpanded = it
                btnClearDone.visibility = if (it) View.VISIBLE else
View.GONE
            }
        }
    }

// Ініціалізація діалогового вікна підтвердження
    private fun showClearDoneConfirmation() {
        AlertDialog.Builder(this)
            .setTitle(R.string.clear_done_confirmation)
            .setPositiveButton(R.string.clear) { _, _ -> clearDoneTasks()
}
            .setNegativeButton(R.string.cancel, null)
            .show()
    }

// Логіка відображення акардеона та шеврона
         private fun toggleSection(recyclerView: RecyclerView, chevron:
TextView, currentExpanded: Boolean, onStateChanged: (Boolean) -> Unit) {
             val nextExpanded = !currentExpanded
             TransitionManager.beginDelayedTransition(boardContainer,
transition)
             recyclerView.visibility = if (nextExpanded) View.VISIBLE
else View.GONE
             chevron.text = if (nextExpanded) "v" else ">"
             onStateChanged(nextExpanded)
             persistSettings()
         }
```

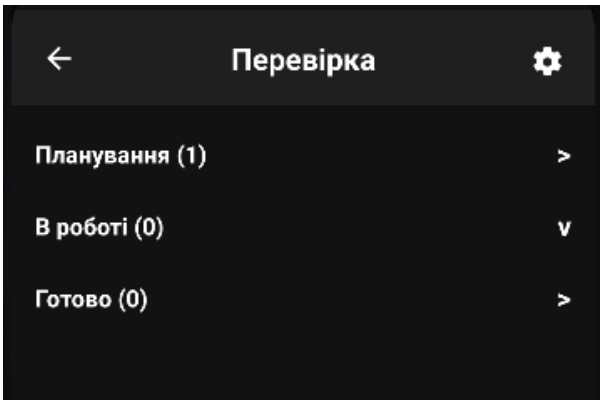


Рис. 3.2. Відображення згорнутих колонок

3.2.2. Перетягування завдань

Для забезпечення природної та передбачуваної взаємодії користувача з елементами планування, процес зміни статусів карток реалізовано через пряме тактильне маніпулювання об'єктами. Замість емуляції складних жестів через слухачі дотиків (OnTouchListener), у проєкті задіяно офіційний нативний механізм операційної системи Android (див. рис. 3.3). Архітектуру цього процесу інкапсульовано у спеціалізованому класі TaskDragHelper.

Процес ініціалізації перетягування починається з пакування ідентифікатора завдання (taskId) в об'єкт системного буфера обміну ClipData із чітко визначеним MIME-типом. Для створення візуального відгуку оригінальний елемент стає напівпрозорим (alpha = 0.4f):

Лістинг 3.6.1. Drag and Drop:

```
// Ініціалізація нативного життєвого циклу перетягування картки
    fun startDrag(view: View, taskId: Int) {
        val clipData = ClipData(
            ClipDescription("task", arrayOf(MIME_TASK_ID)),
            ClipData.Item(taskId.toString())
        )
        draggedView = view
        view.alpha = 0.4f
        view.startDragAndDrop(clipData, View.DragShadowBuilder(view),
taskId, 0)
    }

    fun attachDropTarget(
        recyclerView: RecyclerView,
        targetStatus: String,
        onDrop: (taskId: Int, targetStatus: String) -> Unit
```

```
    ) {
        if (!defaultBackgrounds.containsKey(recyclerView)) {
            defaultBackgrounds[recyclerView] =
recyclerView.solidBackgroundColor()
        }
```

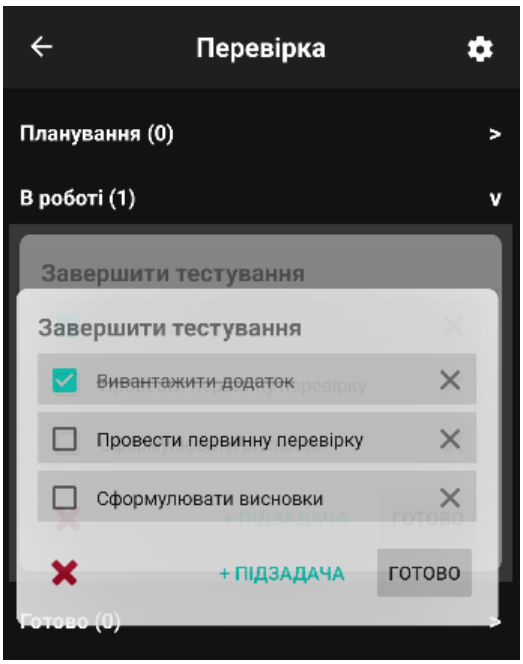


Рис. 3.3. Відображення перетягування

Обробка переміщення тіні картки (*Drag Shadow*) над колонками статусів реалізується через призначення слухача View.OnDragListener на контейнери RecyclerView. Логіка слухача строго обробляє фази життєвого циклу перетягування

Лістинг 3.6.2. Drag and Drop тіні:

```
// Обробка подій перетягування на цільових контейнерах-стовпцях
        recyclerView.setOnDragListener { _, event ->
            when (event.action) {
                DragEvent.ACTION_DRAG_ENTERED -> {
                    if (event.clipDescription?.hasMimeType(MIME_TASK_ID)
== true) {
                        highlightColumn(recyclerView, true)
                        true
                    } else {
                        false
                    }
                }
                DragEvent.ACTION_DRAG_EXITED -> {
                    highlightColumn(recyclerView, false)
                    true
                }
```

```
                DragEvent.ACTION_DROP -> {
                    highlightColumn(recyclerView, false)
                    val taskId = parseTaskId(event) ?:
return@setOnDragListener false
                    onDrop(taskId, targetStatus)
                    true
                }
                DragEvent.ACTION_DRAG_ENDED -> {
                    highlightColumn(recyclerView, false)
                    draggedView?.alpha = 1f
                    draggedView = null
                    true
                }
                else -> true
            }
        }
```

3.2.3. Графічні праметри

Відповідно до методологічних вимог концепції заспокійливих технологій (Calm Technology), візуальне оформлення інтерфейсу повністю виключає використання жорстко закодованих (hardcoded) значень кольорів, розмірів або шрифтових параметрів безпосередньо в атрибутах файлів розмітки XML. Уся графічна, колірна та шрифтова палітра програми централізована у системних ресурсах стилізації, які утворюють архітектурний шар теми оформлення додатка:

- colors.xml – декларує семантичні токени колірної палітри, де пріоритет свідомо віддано пастельним, приглушеним тонам із низьким рівнем насиченості та контрастності для мінімізації стресового впливу на зорову систему користувача та запобігання сенсорному перевантаженню.
- themes.xml (або styles.xml) – об'єднує визначені колірні та шрифтові токени у глобальну тему додатка на базі компонентів Material Design.

Така програмна архітектура стилізації створює єдине джерело істини (Single Source of Truth) для дизайну застосунку. Вона дозволяє гнучко масштабувати візуальний інтерфейс, гарантує стилістичну монолітність програми на всіх екранах та виключає появу непередбачуваного графічного «шуму», що безпосередньо покращує користувацький досвід (UX).

Таким чином, програмна реалізація графічного інтерфейсу повністю відображає задекларовані принципи мінімалізму та когнітивної безпеки. Використання нативних (native) компонентів Android SDK дозволило створити стійкий до навантажень візуальний шар, який не залежить від сторонніх графічних рушіїв чи важких зовнішніх бібліотек [19], [11]. Детально пророблена логіка взаємодії екранів та елементів керування усуває технічний дискомфорт, перетворюючи мобільний застосунок на надійне цифрове полотно для щоденного фокусування користувача на важливих завданнях.

3.3. Реалізація функціональних можливостей Kanban-дошки

За кулісами візуального відображення, розібраного у попередніх підрозділах, ховається справжнє серце програмного продукту – логічна модель керування безпосереднім потоком завдань. Кожне переміщення інформаційного елемента є не просто графічним ефектом зміни його координат на склі мобільного пристрою, а чітко верифікованою транзакцією у локальному сховищі даних. Цей процес повністю підпорядковується закладеним правилам життєвого циклу методології Kanban [26].

3.3.1. Алгоритмічна реалізація життєвого циклу завдань

Як перетворити звичайну цифрову дошку з простого переліку нотаток на динамічний інструмент саморегуляції? Зважаючи на те, що архітектура програми побудована навколо концепції offline-first із постійною серіалізацією об'єктів у текстовий формат JSON. Використання цілочисельного ідентифікатора (Int) та базового рядкового типу (String) для статусу кардинально спрощує роботу бібліотеки GSON. Під час атомарного збереження стану дошки в SharedPreferences, системі не потрібно витрачати процесорний час на виклик кастомних адаптерів типів (TypeAdapters), оскільки рядкові значення нативно трансформуються у JSON-формат. Такий підхід робить структуру даних максимально «пласкою» та легкою.

Лістинг 3.7. Алгоритм зміни статусів завдань та валідації методологічних обмежень:

```
// Реалізація детермінованого скінченного автомата руху завдань
    private fun moveTask(task: Task) {
        when (task.status) {
            "BACKLOG" -> applyTaskStatusChange(task, "IN_PROGRESS")
            "IN_PROGRESS" -> applyTaskStatusChange(task, if (isReviewEna-
bled) "REVIEW" else "DONE")
            "REVIEW" -> applyTaskStatusChange(task, "DONE")
            "DONE" -> { /* no action for completed tasks */ }
        }
    }


// Атомарна зміна стану об'єкта та верифікація лімітів Work-in-Progress
    private fun applyTaskStatusChange(task: Task, newStatus: String) {
        if (task.status == newStatus) return

        val isMovingToInProgress = newStatus == "IN_PROGRESS"
        task.status = newStatus
        persistTasks()
        updateBoards()

        if (isMovingToInProgress && wipLimitEnabled) {
            val inProgressCount = allTasks.count { it.status ==
"IN_PROGRESS" }
            if (inProgressCount >= maxInProgressCount) {
                showWipLimitWarning()
            }
        }
    }
```

3.3.2. Обмеження роботи в процесі

Фундаментальним правилом методології Kanban є обмеження обсягу незавершеної роботи (WIP) для запобігання когнітивному перевантаженню користувача. Під час програмної реалізації цього механізму автор зіткнувся з дилемою вибору між жорстким блокуванням інтерфейсу (Hard Limits) та системою лояльного попередження (Soft Limits) (див. рис. 3.4).

З огляду на те, що інструмент орієнтований на персональне використання, де часто виникають непередбачувані термінові завдання, було обрано

архітектуру м'яких лімітів. Вона не відбирає у користувача контроль над процесом, а діє як віртуальний радник.

Лістинг 3.8. Soft WIP:

```
    private fun onTaskDropped(taskId: Int, targetStatus: String) {
        if (!isDragEnabled) return
        val task = allTasks.find { it.id == taskId } ?: return
        if (targetStatus == "REVIEW" && !isReviewEnabled) return
        applyTaskStatusChange(task, targetStatus)
    }

    private fun updateWipLimitState(inProgressCount: Int) {
        val shouldWarn = wipLimitEnabled && inProgressCount >= maxInPro-
gressCount
        if (shouldWarn == wipLimitWarningActive) return
        wipLimitWarningActive = shouldWarn
        if (shouldWarn) {
            llInProgressHeader.setBackgroundColor(ContextCom-
pat.getColor(this, R.color.wip_warning_background))
            tvInProgressHeader.setTextColor(ContextCompat.getColor(this,
R.color.wip_warning_text))
            tvInProgressChevron.setTextColor(ContextCompat.getColor(this,
R.color.wip_warning_text))
        } else {
            llInProgressHeader.background = originalInProgressHeaderBack-
ground
            tvInProgressHeader.setTextColor(originalInProgressHeader-
TextColor)
            tvInProgressChevron.setTextColor(originalInProgressChevron-
TextColor)
        }
    }

// Попереджувальне сповіщення про Роботу в Процесі
    private fun showWipLimitWarning() {
        Snackbar.make(boardContainer, R.string.wip_limit_reached_message,
Snackbar.LENGTH_LONG).show()
    }
```

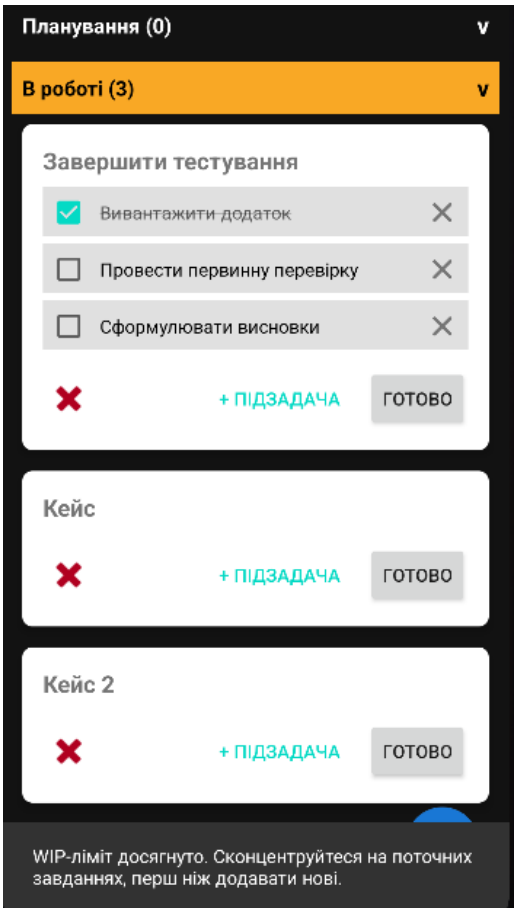


Рис. 3.4. Результат досягнення WIP-лімітів

3.3.3. Багаторівнева система конфігурації та збереження налаштувань дошки

Для забезпечення максимальної гнучкості та персоналізації робочого процесу, в архітектуру застосунку закладено багаторівневу (ієрархічну) систему налаштувань. Оскільки інструмент дозволяє користувачеві вести кілька незалежних проєктів, виникла інженерна необхідність розділити глобальні параметри програми та специфічні налаштування конкретної дошки.

Рішення цієї задачі реалізовано через патерн резервного успадкування (Fallback Mechanism), де параметри поділяються на базові (Master Settings) та локальні (Project Settings) (див. рис. 3.5).

Лістинг 3.9. Структура моделі даних конфігурації:

```
// Модель конфігурації функціоналу та візуального стану дошки
data class BoardSettings(
    val dragEnabled: Boolean = true,
    val reviewEnabled: Boolean = false,
```

```
    val priorityEnabled: Boolean = false,
    val priorityChangeEnabled: Boolean = false,
    val wipLimitEnabled: Boolean = false,
    val maxInProgressCount: Int = 3,
    // Збереження станів інтерфейсу (акордеона)
    val backlogExpanded: Boolean? = null,
    val inProgressExpanded: Boolean? = null,
    val reviewExpanded: Boolean? = null,
    val doneExpanded: Boolean? = null
)
```

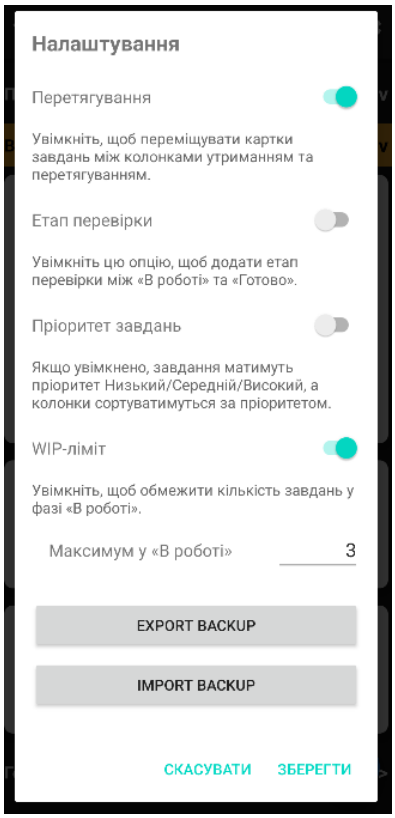


Рис. 3.5. Панель налаштувань проєкту

Щоб позбавити користувача необхідності налаштовувати кожну нову дошку з нуля, система використовує концепцію Майстер-налаштувань (див. рис. 3.6) (Master Defaults). Вони зберігаються в SharedPreferences ізольовано, під ключем MASTER_SETTINGS_KEY, і керуються з головного екрана MainActivity.

Лістинг 3.10. Методи розв'язання ефективної конфігурації:

```
// Визначення ефективних налаштувань для конкретного проєкту
fun getEffectiveSettings(project: Project): BoardSettings =
    project.settings ?: getMasterDefaults()


// Отримання базового (глобального) шаблону налаштувань
```

```
    fun getMasterDefaults(): BoardSettings {
        val json = sharedPref.getString(MASTER_SETTINGS_KEY, null) ?:
return BoardSettings()
        return gson.fromJson(json, BoardSettings::class.java) ?: Board-
Settings()
    }

    // Запис локальних налаштувань із прив'язкою до ідентифікатора
проєкту
    fun updateProjectSettings(projectId: Int, settings: BoardSettings)
{
        val projects = getProjects().toMutableList()
        val index = projects.indexOfFirst { it.id == projectId }
        if (index == -1) return
        projects[index] = projects[index].copy(settings = settings)
        saveProjects(projects)
    }
```

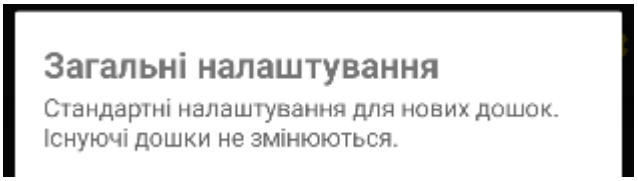


Рис. 3.6. «Майстер» панель налаштування

Модифікація та персистентна фіксація налаштувань відбувається безпосередньо під час взаємодії користувача з Kanban-дошкою. Контролер KanbanActivity динамічно зчитує обчисілену конфігурацію, транслює її на стан інтерфейсних елементів, а у разі маніпуляцій (наприклад, згортання колонки) – ініціює зворотний запис.

Лістинг 3.11. Механіка застосування та збереження конфігурації в KanbanActivity:

```
        private fun applyBoardSettings(settings: BoardSettings) {
            isDragEnabled = settings.dragEnabled
            isReviewEnabled = settings.reviewEnabled
            isPriorityEnabled = settings.priorityEnabled
            isPriorityChangeEnabled = settings.priorityChangeEnabled
            wipLimitEnabled = settings.wipLimitEnabled
            maxInProgressCount = settings.maxInProgressCount

            isBacklogExpanded = settings.backlogExpanded ?: true
            isInProgressExpanded = settings.inProgressExpanded ?: true
            isReviewExpanded = settings.reviewExpanded ?: true
            isDoneExpanded = settings.doneExpanded ?:
!settings.reviewEnabled
```

```
}

private fun persistSettings() {
    val id = projectId.toIntOrNull() ?: return
    sharedPrefHelper.updateProjectSettings(
        id,
        BoardSettings(
            dragEnabled = isDragEnabled,
            reviewEnabled = isReviewEnabled,
            priorityEnabled = isPriorityEnabled,
            priorityChangeEnabled = isPriorityChangeEnabled,
            wipLimitEnabled = wipLimitEnabled,
            maxInProgressCount = maxInProgressCount,
            backlogExpanded = isBacklogExpanded,
            inProgressExpanded = isInProgressExpanded,
            reviewExpanded = isReviewExpanded,
            doneExpanded = isDoneExpanded
        )
    )
}
```

3.4. Висновки до розділу 3

Можна зробити висновок, що програмна реалізація функціональних можливостей Kanban-дошки перетворює звичайний мобільний інтерфейс на повноцінне середовище управління персональною продуктивністю. Особливу увагу звертаю на каскадну модель налаштувань, яка дозволяє як встановити бажані параметри, так і гнучко налаштовувати кожну дошку.

Автор свідомо відмовився від важких зовнішніх баз даних. Локальна серіалізація об'єктних моделей у звичний формат JSON стала основою збереження інформації. Програма працює абсолютно автономно.

На закінчення зазначу, що суворе дотримання лімітів незавершеної роботи та статус «Готово» дозволяють користувачу наочно бачити вузькі місця у власних повсякденних процесах.

РОЗДІЛ 4. ТЕСТУВАННЯ ЗАСТОСУНКУ ТА ОЦІНКА РЕЗУЛЬТАТІВ

Цей розділ присвячений комплексному аналізу надійності, верифікації функціональних параметрів та оцінці практичної ефективності розробленого мобільного застосунку. Процес тестування є критично важливим етапом життєвого циклу розробки програмного забезпечення, який дозволяє гарантувати стабільність архітектурних рішень, коректність роботи бізнес-логіки в автономному режимі (offline-first) та відсутність дефектів у критичних сценаріях користувацького досвіду.

У межах розділу наведено методологію проведення тестування, описано тестові сценарії для перевірки логіки скінченного автомата транзиту завдань і механізмів контролю лімітів незавершеної роботи (WIP Limits). Також здійснено підсумкову оцінку відповідності реалізованого графічного інтерфейсу задекларованим принципам цифрового мінімалізму та концепції заспокійливих технологій (Calm Technology).

4.1. Методика тестування

Валідація працездатності та оцінка якості розробленого програмного забезпечення є критичним етапом, без якого неможливо гарантувати відповідність продукту заявленим технічним вимогам [10]. Тестування системи – процес складний. Але як саме гарантувати стабільність локального автономного інструменту без постійного моніторингу з боку серверної частини? Автор вважає, що оптимальним шляхом є комбінування декількох підходів, де поєднуються класичні методи «білої скриньки» для внутрішніх алгоритмів та «чорної скриньки» для перевірки реакції інтерфейсу [2]. Таке поєднання дозволяє виявити приховані дефекти ще на ранніх стадіях збірки додатка. Менше помилок у релізі. Програма працює стабільно.

Головний акцент у ході перевірки було зміщено на модульне тестування (Unit testing) бізнес-логіки, яка керує станами індивідуальної Kanban-дошки. Якщо часові метрики будуть обчислюватися з помилками, весь аналітичний

зміст проєкту просто втрачається, оскільки користувач отримуватиме хибні дані про власну ефективність [4]. Що стосується безпосередньо самого процесу розробки тестів, то тут застосовувалися автоматизовані сценарії, які імітують граничні випадки: наприклад, спробу штучно додати нову картку до стовпця, де ліміт незавершеної роботи вже повністю вичерпано. Як би там не було, деякі баги все одно вилазять у найменш очікуваних місцях, тому ручна перевірка також мала місце. Зрозуміло, що охопити автоматизованими тестами абсолютно кожен клік неможливо, та й навряд чи потрібно.

Окремим, але надзвичайно важливим викликом стало юзабіліті-тестування інтерфейсу на реальних пристроях із різною роздільною здатністю екрана. Для користувачів творчих професій або осіб із СДУГ критично, щоб анімація перетягування карток не сіпалася і не викликала когнітивного роздратування. На етапі тестування спеціально відстежувався час відгуку екрана на дотик, оскільки навіть мікроскопічна затримка у відображенні здатна зруйнувати стан потоку та відволікти людину від роботи.

Інструментальне тестування користувацького інтерфейсу супроводжувалося певними труднощами через специфіку взаємодії з елементами візуалізації даних. Емуляція жестів перетягування (Drag-and-Drop) потребувала точного налаштування координат віртуального екрана [10]. Під час тестів іноді виникала невелика плутанина з позиціонуванням карток у горизонтальних списках, коли додаток намагався одночасно обробити і жест прокручування стовпців, і підйом самої графічної плашки. Проте, завдяки чіткому розділенню архітектурних шарів, усі неполадки вдалося локалізувати суто всередині presentation-рівня, не зачіпаючи логіку збереження даних [2].

Сказане дає змогу зробити висновок: що сформована методика тестування дозволила всебічно оцінити стійкість додатка до різноманітних сценаріїв використання фахівцем. Поєднання автоматизованих скриптів контролю станів із ручною перевіркою ергономіки забезпечило високу надійність індивідуального планувальника. Реалізований комплекс тестування підтвердив готовність

мобільного застосунку до тривалого щоденного використання без ризику раптової втрати важливих користувацьких даних.

4.2. Розробка тестових сценаріїв

Таблиця 4.1.

Матриця сценаріїв функціонального тестування застосунку

ID	Назва тест-кейсу	Початкові умови	Кроки виконання	Очікуваний результат
TC-01	Транзит станів: перенесення з Backlog в In Progress	Створено задачу із status = "BACKLOG"	Виклик методу moveTask(task)	Статус задачі змінюється на "IN_PROGRESS", викликається persistTasks() та updateBoards().
TC-02	Динамічний обхід етапу Review (якщо вимкнено)	isReviewEnabled = false, задача в статусі "IN_PROGRESS"	Виклик методу moveTask(task)	Система минає етап REVIEW і переводить задачу безпосередньо в "DONE".
TC-03	Спрацювання тригера WIP Limit	wipLimitEnabled = true, maxInProgressCount = 3. У стовпці вже є 3 задачі.	Переміщення 3-ї задачі в стовпець "IN_PROGRESS".	Задача переміщується, але спрацьовує метод showWipLimitWarning(), на екрані з'являється попередження.
TC-04	Резервне успадкування налаштувань (Fallback)	Новий проєкт, у якого project.settings == null	Виклик getEffectiveSettings(project)	Метод повертає глобальний шаблон налаштувань за замовчуванням getMasterDefaults().

4.3. Аналіз результатів та оцінка ефективності

Реакція графічної оболонки на внутрішні переходи скінченного автомата повністю відповідає задекларованій логіці розробки. Зокрема, початковий транзит картки відображено на рис. 4.1. Тут зафіксовано момент перенесення завдання до стовпця «В роботі» за сценарієм тест-кейсу ТС-01.

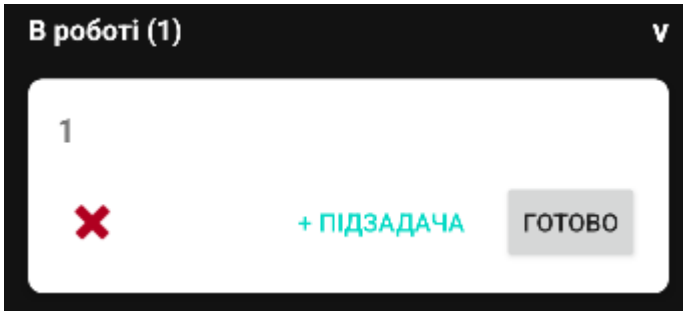


Рис. 4.1. Візуалізація картки перенесеної до «В роботі»

Як видно з рис. 4.1, система успішно змінює статус задачі. Візуальний шар реагує миттєво. Зміна статусу об'єкта відбувається без жодних видимих затримок чи збоїв у рендерингу, що підтверджує стабільність presentation-рівня застосунку.

Дещо іншу інтерфейсну поведінку – вже з урахуванням індивідуальних конфігурацій дошки – ілюструє рис. 4.2. На цьому скриншоті відтворено виконання тест-кейсу ТС-02. Мова йде про випадок, коли проміжний етап «Перевірка» був заздалегідь вимкнений користувачем у налаштуваннях.

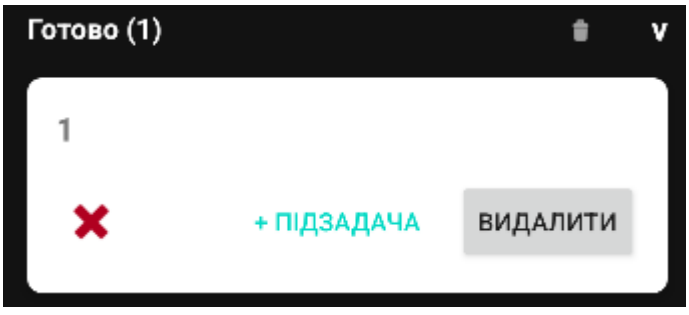


Рис. 4.2. Візуалізація картки перенесеної до «Готово»

Як видно з рисунка, логіка обходу спрацьовує бездоганно. Система оминає непотрібний стовпець і переводить задачу безпосередньо в статус «Готово».

Графічний інтерфейс системи реагує на події скінченного автомата згідно із задекларованими вимогами. На рис. 4.3 продемонстровано інтерфейс вікна Kanban-дошки у момент спрацювання тригера перевищення ліміту Work-in-Progress (Тест-кейс TC-03).

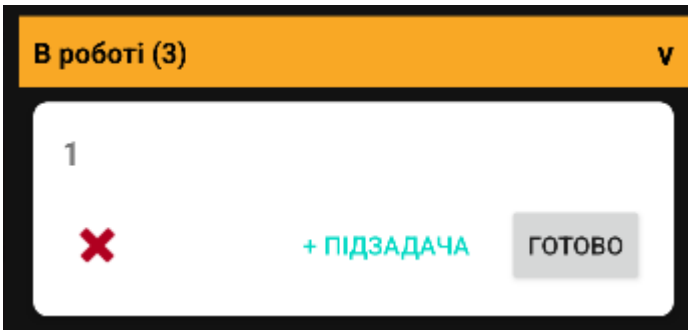
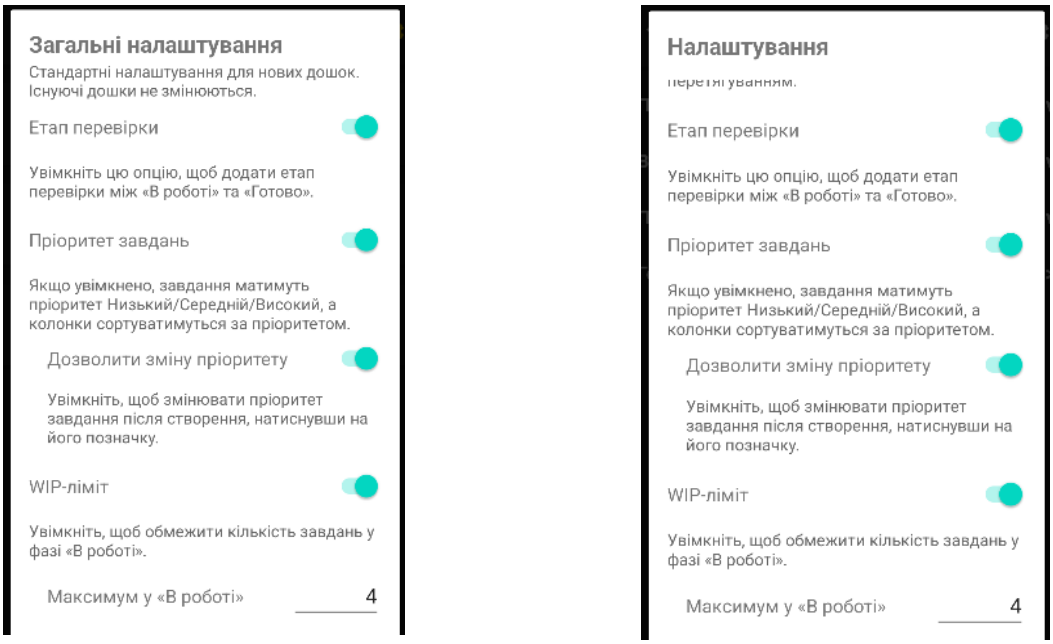


Рис. 4.3. Візуалізація попередження системи при порушенні лімітів WIP

Як видно з рис. 4.3, система не блокує дію користувача жорстко, а підсвічує стовпець, що підтверджує успішну реалізацію концепції Calm Technology.

Наостанок варто розглянути роботу підсистеми збереження та конфігурації екрана (див. рис. 4.4). Тест-кейс TC-04 верифікує логіку роботи з налаштуваннями за замовчуванням при створенні нових елементів.



а) б)

Рис. 4.4. Порівняльний аналіз меню налаштувань а)«Master» налаштування б)«Project» налаштування

Внесені користувачем параметри лімітів не просто записуються в пам'ять пристрою, а й автоматично стають базовим шаблоном для кожної нової дошки. Отже, усувається необхідність повторної рутинної налаштування робочого простору вручну. На закінчення зазначу, що всі базові сценарії валідації станів підтвердили повну працездатність візуальної та логічної структури мобільного застосунку.

4.4. Висновки до розділу 4

Сформований комплекс верифікації став фінальним і, мабуть, найбільш безкомпромісним суддею для розробленого застосунку. Перевірка працездатності – це завжди дещо рутинна, але критично важлива робота. Особливо коли йдеться про автономний інструмент, позбавлений постійної підтримки з боку серверної частини.

Отже, поєднання автоматичного моніторингу внутрішніх станів із ручною перевіркою інтерфейсних дрібниць забезпечило той рівень надійності, який повністю нівелює ризик раптової втрати чи деградації користувацьких даних у майбутньому.

ВИСНОВКИ

Сказане дає змогу зробити висновок, що весь шлях від первинного теоретичного пошуку до створення готового програмного продукту став для автора не просто сухим інженерним завданням, а глибоким дослідженням на стику методології управління та цифрової ергономіки. За каркасом проєкту стоїть серйозне переосмислення класичного інструментарію. Як виявилося, пряме перенесення командних принципів Kanban на рейки індивідуального використання вимагає кардинальної зміни пріоритетів. Популярні комерційні системи, такі як Jira, Trello чи ClickUp, занадто громіздкі. Вони перевантажені функціями колективної взаємодії та намертво прив’язані до всесвітньої мережі. Але чи здатні вони захистити фокус уваги окремого фахівця? Практика показує, що ні. Автор вважає, що саме концепція заспокійливих технологій (Calm Technology) у поєднанні з м'якими лімітами незавершеної роботи (WIP Limits) є ледве не єдиною противагою хронічному інформаційному вигоранню, яка так необхідна сучасній людині, зокрема й користувачам із проявами СДУГ.

На думку автора, рішення побудувати систему за принципом offline-first із використанням локальної серіалізації об’єктних моделей у формат JSON було повністю виправданим, а суворе розділення презентаційного шару (Presentation Layer) та підсистеми збереження дозволило заперти всі інтерфейсні капризи всередині графічної оболонки. Програма працює стабільно. Вона не «вилітає» при критичних навантаженнях чи спробах навмисного пробиття встановлених методологічних меж, а делікатно підсвічує порушення колірними тригерами, зберігаючи стан психологічного комфорту людини.

На закінчення варто згадати, що створений мобільний застосунок є цілісним, повністю автономним інженерним рішенням, яке готове до тривалої щоденної експлуатації. Результати випробувань на реальному фізичному пристрої та емуляторах довели, що індивідуальна візуалізація робочих процесів на засадах Kanban суттєво полегшує процеси саморегуляції. Це повертає людині контроль над власним часом у нашому занадто галасливому цифровому середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бердашенко В. Аналіз швидкодійності відтворення інтерфейсу мобільних додатків на кросплатформенних платформах Flutter та React Native [Електронний ресурс] / Іванов О. – Електрон. дан. – Д. : УДНТ, 2022. – Режим доступу: World Wide Web. – URL: https://crust.ust.edu.ua/server/api/core/bitstreams/c4a4c51c-fe15-44c5-9d14-745d99ab6f79/content – Загл. з екрану (переглянуто 29.05.2026).

2. Підгерський І. О. Система організації щоденних справ з дошкою Kanban [Електронний ресурс] / Тихоход В. – Електрон. дан. – К. : КПІ ім. Ігоря Сікорського, 2024. – Режим доступу: World Wide Web. – URL: https://ela.kpi.ua/server/api/core/bitstreams/d211d7a3-2b2e-4431-9bd8-b59f108ada6f/content – Загл. з екрану (переглянуто 29.05.2026).

3. Поета І. Розробка мобільного додатку на React Nativ – 2021. - 29 с.

4. Соболєва А. В. ОСОБЛИВОСТІ МЕТОДОЛОГІЇ KANBAN ТА SCRUM

ПРИ РЕАЛІЗАЦІЇ ПРИНЦИПІВ AGILE-МАРКЕТИНГУ [Електронний ресурс] / Радченко Г. А. – Електрон. дан. – К. : Держ. ун-т «Київський авіаційний інститут», 2023. – Режим доступу: World Wide Web. – URL: https://er.kai.edu.ua/server/api/core/bitstreams/ed41e903-f5e4-4c90-87e5-5acd0b031899/content – Загл. з екрану (переглянуто 29.05.2026).

5. Kotlin. Вступ [Електронний ресурс] / W3Schools UA. – Електрон. дан. – Режим доступу: World Wide Web. – URL: https://w3schoolsua.github.io/kotlin/kotlin_intro.html – Загл. з екрану (переглянуто 29.05.2026).

6. Гомілко О.О. Розробка мобільних застосунків за допомогою технологій Flutter та Dart [Електронний ресурс] / Жежерун О.П. – Електрон. дан. – К. : НаУКМА, [рік]. – Режим доступу: World Wide Web. – URL: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/826f66d9-5bf0-48c2-a443-e596584b690e/content – Загл. з екрану (переглянуто 29.05.2026).

7. Демківська Т.І. Розробка інформаційної системи управління ІТ-проєктів із застосуванням фреймворку Kanban / Т.І. Демківська, М.С. Андрійченко // VII Міжнародна науково-практична конференція «Мехатронні системи: інновації та інжиніринг» : тези доп. – К. : КНУТД, 2023. – С. 183–184.

8. Канбан-дошка для роботи: що це, як і навіщо використовувати [Електронний ресурс] / Happy Monday Media. – Електрон. дан. – К. : Happy Monday, 2026. – Режим доступу: World Wide Web. – URL: https://happymonday.ua/kanban-doska-dlja-raboty-kak-ispolzovat – Загл. з екрану (переглянуто 29.05.2026).

9. Kanban: основні принципи та користь [Електронний ресурс] / Laba. – Електрон. дан. – К. : Laba, 2023. – Режим доступу: World Wide Web. – URL: https://laba.ua/blog/1529-kanban-principy-i-polza – Загл. з екрану (переглянуто 29.05.2026).

10. Харенко Д.О. Розробка Android-застосунку для допомоги ведення здорового способу життя мовою Kotlin : кваліф. робота бакалавра : спец. 121 Інженерія програмного забезпечення / Д.О. Харенко ; [наук. кер. С. Шевченко] ; Держ. ун-т інформаційно-комунікаційних технологій, Каф. інженерії програмного забезпечення. – К., 2024. – 68 с.

11. Вередін М. О. Особливості UX/UI-дизайну при розробці вебсайтів та мобільних застосунків / Козуб Г. // [Назва конференції МЦНД] : тези доп. – Ужгород : МЦНД, 2025. – С. 122-125.

12. Самсоненко Є.С. Використання мікрологістичної системи «Канбан» на виробництві / Є.С. Самсоненко // МАРКЕТИНГОВИЙ ТА ЛОГІСТИЧНИЙ МЕНЕДЖМЕНТ : тези доп. – Дніпро : ХАДІ, 213. – С. 561–564.

13. Скрипник Т. Управління ІТ-проектами: використання гнучких методологій / Т. Скрипник, Л. Вознюк, Є. Мандзюк // Herald of Khmelnytskyi National University. Technical Sciences. – 2025. – Vol. 353, Iss. 3.2. – С. 224–230.

14. Що таке Канбан [Електронний ресурс] / BrainRain. – Електрон. дан. – К. : BrainRain, 2024. – Режим доступу: World Wide Web. – URL:

https://brainrain.com.ua/uk/shcho-take-kanban/ – Загл. з екрану (переглянуто 29.05.2026).

15. Roman T. Що можна і чого не можна робити в мобільному UX/UI дизайні [Електронний ресурс] / T. Roman // UX/UI UA (Medium). – 2024. – Режим доступу: World Wide Web. – URL: https://medium.com/uxuiua/%D1%89%D0%BE-%D0%BC%D0%BE%D0%B6%D0%BD%D0%B0-%D1%96-%D1%87%D0%BE%D0%B3%D0%BE-%D0%BD%D0%B5-%D0%BC%D0%BE%D0%B6%D0%BD%D0%B0-%D1%80%D0%BE%D0%B1%D0%B8%D1%82%D0%B8-%D0%B2-%D0%BC%D0%BE%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%D1%83-ux-ui-%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD%D1%96-6b4a6eab9ed3 – Загл. з екрану (переглянуто 29.05.2026).

16. Лізогуб А. IMPROVING THE EFFICIENCY OF INNOVATION AND INVESTMENT PROCESSES OF THE ENTERPRISE USING MODERN MANAGEMENT PRACTICES (EXAMPLE KAIZEN, KANBAN AND SIX SIGMA). // Науковий вісник Одеського національного економічного університету. – 2024. – С. [89-96].

17. Управління завданнями [Електронний ресурс] / Worksection. – Електрон. дан. – К. : Worksection, 2026. – Режим доступу: World Wide Web. – URL: https://worksection.com/ua/faq/managing-tasks.html – Загл. з екрану (переглянуто 29.05.2026).

18. Anderson D.J. Essential Kanban Condensed [Electronic resource] / D.J. Anderson, A. Carmichael. – Electronic data. – Seattle : Lean Kanban University Press, 2016. – Mode of access: WWW. – URL: https://www.thescrummaster.co.uk/wp-content/uploads/2019/09/Essential-Kanban-Condensed-v1.0.0.pdf – Title from screen (переглянуто 29.05.2026).

19. Android Developers [Electronic resource] / Google LLC. – Electronic data. – Mountain View : Google, 2026. – Mode of access: WWW. – URL: https://developer.android.com/develop – Title from screen (viewed 29.05.2026).

20. Brechner E. Agile Project Management with Kanban / E. Brechner. – Redmond : Microsoft Press, 2015. – 160 p. – 171 с.

21. Brisqi [Електронний ресурс] / Brisqi Inc. – Електрон. дан. – Режим доступу: World Wide Web. – URL: https://play.google.com/store/apps/details?id=com.brisqi.brisqiapp – Загл. з екрану (переглянуто 29.05.2026).

22. ClickUp University [Electronic resource] / ClickUp. – Electronic data. – San Diego : ClickUp, 2026. – Mode of access: WWW. – URL: https://university.clickup.com/ – Title from screen (viewed 29.05.2026).

23. Вередін М. О. Особливості UX/UI-дизайну при розробці вебсайтів та мобільних застосунків / Козуб Г. // [Назва конференції МЦНД] : тези доп. – Ужгород : МЦНД, 2025. – С. 122-125.

24. Kanban for Android [Електронний ресурс] / Kanban App. – Електрон. дан. – Режим доступу: World Wide Web. – URL: https://play.google.com/store/apps/details?id=io.kanbanapp – Загл. з екрану (переглянуто 29.05.2026).

25. Kotlin Documentation [Electronic resource] / JetBrains. – Electronic data. – Prague : JetBrains, 2026. – Mode of access: WWW. – URL: https://kotlinlang.org/docs/home.html – Title from screen (viewed 29.05.2026).

26. The Official Kanban Guide [Electronic resource] / Kanban University. – Electronic data. – Seattle : Mauvius Group Inc., 2021. – Mode of access: WWW. – URL: https://resources.kanban.university/wp-content/uploads/2021/06/The-Official-Kanban-Guide_A4.pdf – Title from screen (viewed 29.05.2026).

27. TickTick: To Do List & Calendar [Electronic resource] / Appest Inc. – Electronic data. – Mode of access: WWW. – URL: https://play.google.com/store/apps/details?id=com.ticktick.task – Title from screen (viewed 29.05.2026).

28. Trello Guide [Electronic resource] / Atlassian. – Electronic data. – Sydney : Atlassian, 2026. – Mode of access: WWW. – URL: https://trello.com/guide – Title from screen (viewed 29.05.2026).

ДОДАТКИ

Додаток А

Jira