DOI: https://doi.org/10.36910/6775-2524-0560-2023-53-17
УДК: 004.72
**Домарацький Ілля Віталійович**[1], магістр
**Багнюк Наталія Володимирівна**[1], к.т.н., доцент
https://orcid.org/0000-0002-7120-5455
**Бортник Катерина Яківна**[1], к.т.н., доцент
https://orcid.org/0000-0001-5282-099X
**Тищук Інна Володимирівна**[2], к.е.н., доцент
https://orcid.org/0000-0003-1042-9495
[1]Луцький національний технічний університет, м. Луцьк, Україна
[2] Волинський національний університет ім. Лесі Українки, м. Луцьк, Україна

# ЗАСОБИ РОЗРОБКИ КРОСПЛАТФОРМЕННОГО МОБІЛЬНОГО ДОДАТКУ

**Домарацький І.В., Багнюк Н.В., Бортник К.Я., Тищук І.В. Засоби розробки кросплатформенного мобільного додатку.** У статті докладно розглядаються основні кроки та процеси створення кросплатформенного мобільного додатку. Стаття розкриває ключові виклики та стратегії, необхідні для розробки ефективних мобільних додатків, які відповідають високим стандартам якості та функціональності на різних операційних системах. Основні аспекти, які розглядаються в статті, включають важливість сумісності з різними платформами, оптимізацію продуктивності, забезпечення консистентності інтерфейсу та виклики, пов'язані з оновленням та підтримкою додатків. Стаття підкреслює, що успішна кросплатформенна розробка вимагає глибокого розуміння різних технічних аспектів, гнучкості у підходах до розробки та здатності ефективно реагувати на виклики, що виникають у цій сфері.
**Ключові слова:** Кросплатформенна розробка, Мобільні додатки, React Native

**Domaratskyi I., Bagniuk N., Bortnyk K., Tyschuk I. Cross-Platform Mobile Application development tools.** This article thoroughly examines the fundamental steps and processes involved in creating a cross-platform mobile application. It unveils key challenges and strategies necessary for developing effective mobile applications that meet high standards of quality and functionality across various operating systems. The main aspects discussed in the article include the importance of compatibility with different platforms, optimization of performance, ensuring interface consistency, and challenges related to the updating and support of applications. The article emphasizes that successful cross-platform development requires a deep understanding of various technical aspects, flexibility in development approaches, and the ability to effectively respond to challenges that arise in this field.
**Key words:** Cross-Platform Development, Mobile Applications, React Native.

**Постановка проблеми.**

У сучасному світі, який стрімко розвивається та змінюється під впливом цифрових технологій, кросплатформенна розробка стала одним із ключових напрямків у сфері створення мобільних додатків. Незалежно від того, чи йдеться про стартапи, які шукають шляхи швидкого входу на ринок, чи про великі корпорації, які прагнуть максимально охопити аудиторію, кросплатформенність набуває все більшого значення.

Зростання популярності мобільних пристроїв та різноманіття операційних систем ставлять перед розробниками важливе завдання: створювати додатки, які б не лише відповідали високим стандартам якості та функціональності, але й були доступними для широкого кола користувачів, незалежно від вибору операційної системи. Це викликає потребу в розробці таких методологій та підходів, які дозволяли б ефективно та з мінімальними витратами адаптувати додатки для різних платформ.

Кросплатформенна розробка відповідає на ці виклики, пропонуючи інструменти та технології, які дозволяють створювати універсальні рішення. Вона допомагає скоротити час та витрати на розробку, одночасно забезпечуючи ширше охоплення ринку. Проте, важливо розуміти, що цей підхід також несе в собі певні виклики та обмеження, які потребують глибокого аналізу та розуміння.

В цій статті розглянуто ключові аспекти кросплатформенної розробки, аналізуючи її переваги, виклики та найефективніші практики. Метою є надати комплексне розуміння цієї теми та вказати на потенціал, який вона може відкрити для розробників мобільних додатків у найближчому майбутньому.

**Викладення основного матеріалу.**

Ключовим елементом у кросплатформенній розробці є здатність ефективно вирішувати ряд викликів, які ця сфера ставить перед розробниками. Ці виклики визначають не тільки технічні аспекти розробки, але й загальну стратегію проектування та реалізації додатків.

Найбільш очевидним викликом є необхідність забезпечення сумісності додатку з різними операційними системами, такими як iOS та Android. Це вимагає ретельного розуміння особливостей кожної системи, а також вміння адаптувати додаток таким чином, щоб він не тільки коректно працював, але й зберігав однаковий рівень продуктивності та користувацького досвіду на різних пристроях.

Розробка кросплатформенних додатків часто супроводжується викликами, пов'язаними з оптимізацією продуктивності. Різні пристрої мають різні характеристики обладнання, тому важливо забезпечити, щоб додаток ефективно функціонував на всіх цільових пристроях без значних втрат у швидкості або реакції [1].

Ще одним важливим аспектом є забезпечення консистентності користувацького інтерфейсу та загального досвіду користувача. Важливо, щоб додаток не тільки виглядав однаково на різних платформах, але й пропонував схожу функціональність і зручність використання.

Розробка кросплатформенних додатків також вимагає ретельного планування оновлень та підтримки. З огляду на те, що зміни потрібно впроваджувати одночасно на різних платформах, це може створювати додаткові логістичні та технічні виклики.

Нарешті, важливим викликом є забезпечення безпеки та конфіденційності користувацьких даних у кросплатформенному середовищі. Розробники повинні враховувати різні стандарти.

У сфері кросплатформенної розробки мобільних додатків існує кілька ключових технологій, кожна з яких має свої унікальні особливості, переваги та обмеження. Розуміння цих технологій є важливим кроком для вибору найбільш підходящого інструменту для конкретного проекту.

1) React Native. Розроблений Facebook, React Native дозволяє розробникам використовувати JavaScript для створення нативних мобільних додатків. Ця технологія є популярною через свою здатність інтегрувати нативні компоненти інтерфейсу прямо у код JavaScript, що забезпечує гладке та нативне відчуття додатків. Вона ідеально підходить для проектів, де потрібен швидкий розвиток та високе перевикористання коду.

2) Flutter. Від Google, Flutter – це порівняно новий гравець у сфері кросплатформенної розробки, який використовує мову програмування Dart. Його особливістю є власна система віджетів, що дозволяє створювати високопродуктивні та візуально привабливі інтерфейси. Flutter підходить для тих, хто хоче створити візуально вражаючий додаток з плавною анімацією та високою продуктивністю на різних платформах.

3) Xamarin. Розроблений Microsoft, Xamarin використовує C# для створення додатків, які можуть працювати на багатьох платформах, використовуючи один і той же код. Xamarin особливо підходить для тих, хто вже знайомий з екосистемою Microsoft та хоче використовувати потужність .NET для розробки мобільних додатків. Він пропонує високий степінь інтеграції з нативними API кожної платформи.

Кожна з цих технологій має свої сильні та слабкі сторони, і вибір залежить від конкретних вимог проекту, включаючи вимоги до продуктивності, складності інтерфейсу, доступного бюджету та ресурсів, а також досвіду та навичок розробників. Розуміння цих ключових характеристик допоможе вибрати найбільш відповідну технологію для розробки кросплатформенного мобільного додатку. Для чіткішого порівняння ми порівняємо час запиту на кожній з технології (рисунок 1) [2].

Розробка кросплатформенних мобільних додатків стикається з низкою унікальних викликів, які потребують ретельного аналізу та стратегічного підходу. Вирішення цих викликів є ключовим для успішної розробки та впровадження додатків, які задовольняють потреби користувачів на різних платформах.

– Сумісність з різними платформами. Одним із головних викликів є забезпечення сумісності додатку на різних операційних системах, таких як iOS та Android. Це вимагає глибокого розуміння специфікацій кожної платформи та адаптації додатку таким чином, щоб він зберігав однакову функціональність та користувацький досвід на всіх пристроях.

– Продуктивність додатків. Важливим викликом є забезпечення високої продуктивності додатку на всіх цільових пристроях. Різні операційні системи та характеристики обладнання можуть впливати на продуктивність, тому необхідно оптимізувати додаток для забезпечення плавної роботи.

– Консистентність інтерфейсу. Забезпечення однорідного користувацького інтерфейсу та досвіду на різних платформах також є важливим викликом. Розробники повинні враховувати

різницю в дизайні та взаємодії користувачів з різними пристроями, забезпечуючи при цьому єдність і зручність використання.

–Оновлення та підтримка додатків. Підтримка та оновлення кросплатформенних додатків також є складним завданням, оскільки зміни та виправлення помилок потребують одночасного розгортання на різних платформах. Це вимагає чіткого планування та управління версіями.

–Безпека та конфіденційність. Враховуючи різні вимоги до безпеки на різних платформах, забезпечення надійного захисту даних користувачів є критично важливим. Розробники повинні інтегрувати надійні механізми шифрування та автентифікації, забезпечуючи при цьому дотримання відповідних норм і стандартів конфіденційності.

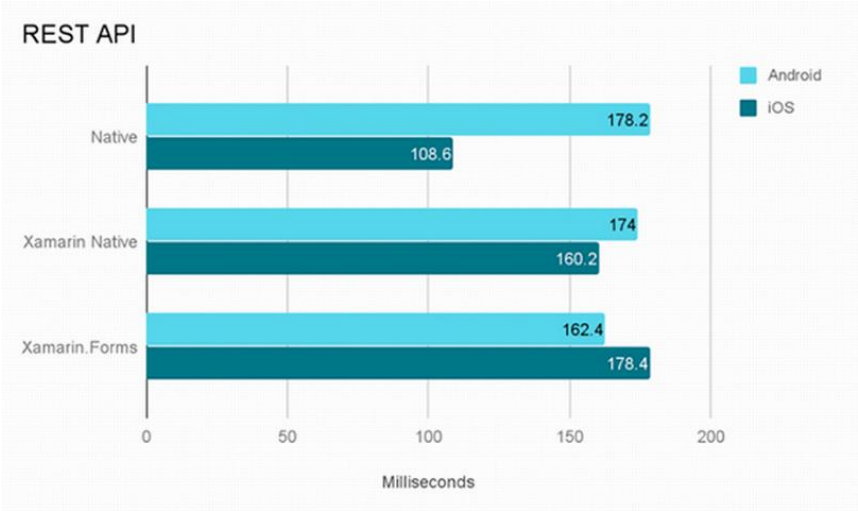


Рис. 1. Порівняння часу запиту на різних технологіях

Ці виклики вимагають від розробників не лише технічної компетентності, але й гнучкості у підходах до розробки, а також глибокого розуміння потреб кінцевих користувачів. Вміння ефективно реагувати на ці виклики визначає успіх кросплатформенного проекту.

Структура Store в даному додатку розроблена для ефективного управління станом та легкої масштабованості. Основні секції store включають:

–Користувачі: Store, який управляє станом користувачів, включаючи авторизацію та дані профілю.

–Дані Інтерфейсу: Store, що містить стан інтерфейсу користувача, включаючи налаштування та UI-елементи.

–Дані Категорій: Store для управління даними категорій, важливими для бізнес-логіки додатку.

Для управління станом використано React hooks від Redux. Це дозволяє ефективно інтегрувати стан Redux з компонентами React, забезпечуючи легкий доступ та оновлення стану.

Дотримання Best Practices щодо нормалізації даних у store, що допомагає уникнути дублювання даних та спрощує їх управління. Це забезпечує більш ефективне використання пам'яті та легше відстеження змін у стані.

Для обробки асинхронних дій та складної бізнес-логіки інтегруємо redux-thunk та redux-saga. Redux-thunk використовується для базових асинхронних операцій, в той час як redux-saga застосовується для більш складних сценаріїв, які вимагають керування багатьма побічними ефектами.

У розробленому додатку використано Redux для централізованого управління станом. Це дозволяє ефективно управляти складними даними та забезпечити однаковий потік даних по всьому додатку. Store організовано на основі трьох основних розділів: користувачі, дані інтерфейсу та дані категорій. Кожен розділ керується окремим reducer, що спрощує розуміння та масштабування коду

Прийнято модульний підхід, де кожен аспект стану управляється окремим модулем. Наприклад, модуль користувачів відповідає за всі дані та дії, пов'язані з авторизацією та інформацією про користувачів. Це забезпечує чітке розділення відповідальності та полегшує розуміння структури store.

Для підвищення продуктивності впроваджено такі техніки, як memoization із використанням useMemo та useCallback у React, щоб уникнути непотрібних rerenders. Селектори з Reselect допомагають ефективно обробляти та вилучати дані зі store. Також використане ледаче завантаження (lazy loading) для важких компонентів та даних, що знижує початкове навантаження на додаток.

Для реактивного оновлення стану в додатку використовуються хуки useSelector і useDispatch з Redux. Це дозволяє React компонентам відслідковувати зміни у сторі та відповідно реагувати на них, забезпечуючи актуальне відображення даних.

useSelector є хуком, який дозволяє витягувати дані з Redux store безпосередньо у компоненті React. Цей хук приймає функцію селектора як свій аргумент. Функція селектора визначає, які частини стану потрібно витягнути. Наприклад стан user як на (рисунок 2).

Реактивність: Коли стан у Redux сторі змінюється, useSelector автоматично виконує повторний рендеринг відповідного компонента, що використовує цей хук. Це гарантує, що компонент завжди має актуальні дані.

Оптимізація: useSelector оптимізує продуктивність за допомогою поверхневого порівняння результатів селектора. Якщо результат селектора не змінився, компонент не перерендериться [4].

useDispatch використовується для відправки дій до store Redux. Цей хук повертає посилання на функцію dispatch store, що дозволяє компонентам запускати дії Redux.

Активація Змін Стану: Через dispatch компоненти можуть відправляти дії, які активують редюсери в Redux. Це є основним способом зміни стану в додатку.

Гнучкість: useDispatch може використовуватися для відправки будь-яких дій Redux, включаючи асинхронні дії, які обробляються middleware, наприклад, за допомогою redux-thunk або redux-saga.

У додатку ці хуки використовуються для створення високо реактивного інтерфейсу. Наприклад, у компоненті, який відображає інформацію про користувача, використано useSelector для витягування цих даних зі store та useDispatch для відправки дії на оновлення інформації про користувача. Коли користувач оновлює свій профіль, дія відправляється через dispatch, і стор оновлюється, що автоматично ініціює оновлення відповідного компонента завдяки useSelector.

```jsx
import React from "react";
import { useSelector, useDispatch } from "react-redux";
import { updateUserAction } from "./actions";

const UserProfile = () => {
  const user = useSelector((state) => state.user);
  const dispatch = useDispatch();

  const handleUpdateUser = (userData) => {
    dispatch(updateUserAction(userData));
  };

  return (
    <View>
      <Text>{`Welcome, ${user.name}`}</Text>
      {/* More UI elements */}
      <Button
        onPress={() => handleUpdateUser(newUserData)}
        title="Update Profile"
      />
    </View>
  );
};
```

Рис. 2. Приклад обробки даних

В додатку використано декілька middleware з різними цілями:

– Redux-Thunk. Дозволяє обробляти асинхронні дії. Це надзвичайно корисно для взаємодії з зовнішніми API та обробки асинхронних запитів.

–Redux-Logger. Використовується для логування дій. Це допомагає відстежувати потік дій та станів у додатку, що є корисним для налагодження.

–Redux-Saga. Використовується для більш складних асинхронних операцій та управління побічними ефектами. Цей middleware дозволяє більш ефективно управляти асинхронною логікою та забезпечує більшу гнучкість, ніж Redux-Thunk [5].

Кожен з middleware використовується для конкретних цілей:

Redux-Thunk використовується для асинхронних дій, таких як запити до бази даних або зовнішніх API так як на (рисунок 3).

Redux-Logger допомагає у логуванні дій та змін стану, що полегшує розробку та відлагодження.

Redux-Saga використовується для більш складних сценаріїв, таких як обробка послідовності асинхронних операцій або виконання дій на основі певних умов.

```
// Action Creator with Redux Thunk for async operations
const fetchUserData = () => {
  return async (dispatch) => {
    try {
      const response = await fetch("https://api.example.com/user");
      const data = await response.json();
      dispatch({ type: "FETCH_USER_SUCCESS", payload: data });
    } catch (error) {
      dispatch({ type: "FETCH_USER_ERROR", error });
    }
  };
};
```

Рис. 3. Приклад використання Redux Thunk

Middleware інтегровані в Redux store стандартним способом за допомогою applyMiddleware з Redux. Це забезпечує гладке впровадження мідлварів та їх взаємодію з основною логікою Redux.

Впровадження цих middleware значно покращує процес розробки та тестування:

Розробка. Мідлвари, особливо Redux-Logger, надають важливу інформацію під час розробки, що допомагає швидко ідентифікувати та виправляти помилки. Redux-Thunk і Redux-Saga полегшують роботу з асинхронними процесами. Як під час запитів для отримання великого об'єму даних які відображені на (рисунок 4).

Тестування. Middleware дозволяють більш ефективно тестувати асинхронні дії та побічні ефекти. Інтеграція з тестовими фреймворками, наприклад, з Jest, дозволяє автоматизувати тестування асинхронної логіки.

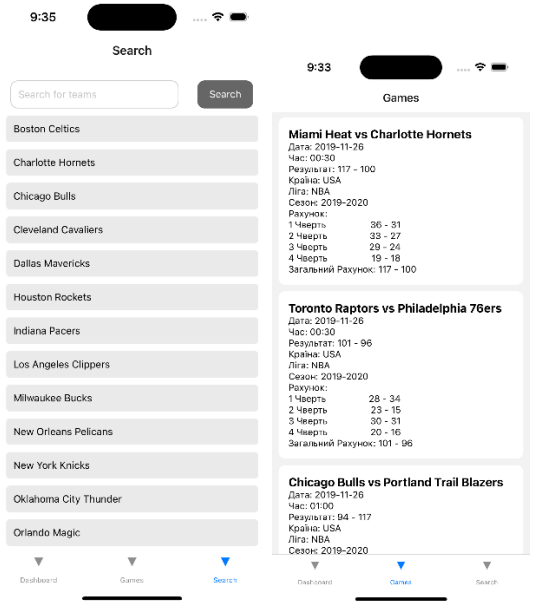


Рис 4. Відображення даних після збереження в store

**Висновки.**

У статті проведено всебічний аналіз кросплатформенної розробки мобільних додатків, з особливим акцентом на React Native, розглянуто ключові особливості, виклики та принципи роботи цієї платформи, а також детально описано її архітектурні підходи та механізм взаємодії Bridge між JavaScript та нативним кодом.

Завдяки своїй унікальній здатності поєднувати нативні компоненти з гнучкістю JavaScript, React Native відкриває широкі можливості для розробників. Це дозволяє ефективно створювати високопродуктивні мобільні додатки, що забезпечують чудовий користувацький досвід на різних платформах. Однак, успіх у кросплатформенній розробці також вимагає глибокого розуміння викликів, пов'язаних з управлінням продуктивності, сумісністю та безпекою.

Дослідивши можливі способи взаємодії даних в середині додатку без та з використанням принципів бібліотеки Redux однозначно стає зрозуміло, що їх використання пришвидшує та мінімізує процес розробки. Що дозволяє зосереджуватись тільки на основних аспектах розробки.

Зрештою, React Native – це потужний інструмент у руках розробників, який, за умови правильного використання може значно збільшити ефективність розробки та розширити досяжність мобільних додатків. Чи йдеться про стартапи, що шукають швидкого входу на ринок, чи про великі компанії, які прагнуть охопити максимально широку аудиторію, React Native пропонує гнучкі та ефективні рішення для розробки мобільних додатків.

**Список бібліографічного опису**

1. Guide to cross-platform app development and testing - keenethics. Keenethics. URL: https://keenethics.com/blog/cross-platform-app-development-and-testing (date of access: 29.11.2023).
2. Benefits of cross-platform app development | linkup studio. Linkup Studio. URL: https://linkupst.com/blog/advantages-of-cross-platform-development/ (date of access: 29.11.2023).
3. Awati R. What is cross-platform mobile development? – TechTarget Definition. Mobile Computing. URL: https://www.techtarget.com/searchmobilecomputing/definition/cross-platform-mobile-development (date of access: 29.11.2023).
4. Griffith C. What is cross-platform mobile application development? | ionic. Ionic: Enterprise App Platform. URL: https://ionic.io/resources/articles/what-is-cross-platform-app-development (date of access: 29.11.2023).
5. The six most popular cross-platform app development frameworks | kotlin. Kotlin Help. URL: https://kotlinlang.org/docs/cross-platform-frameworks.html#flutter (date of access: 29.11.2023).

**References**

1. Guide to cross-platform app development and testing - keenethics. Keenethics. URL: https://keenethics.com/blog/cross-platform-app-development-and-testing (date of access: 29.11.2023).
2. Benefits of cross-platform app development | linkup studio. Linkup Studio. URL: https://linkupst.com/blog/advantages-of-cross-platform-development/ (date of access: 29.11.2023).
3. Awati R. What is cross-platform mobile development? – TechTarget Definition. Mobile Computing. URL: https://www.techtarget.com/searchmobilecomputing/definition/cross-platform-mobile-development (date of access: 29.11.2023).
4. Griffith C. What is cross-platform mobile application development? | ionic. Ionic: Enterprise App Platform. URL: https://ionic.io/resources/articles/what-is-cross-platform-app-development (date of access: 29.11.2023).
5. The six most popular cross-platform app development frameworks | kotlin. Kotlin Help. URL: https://kotlinlang.org/docs/cross-platform-frameworks.html#flutter (date of access: 29.11.2023.