Plag

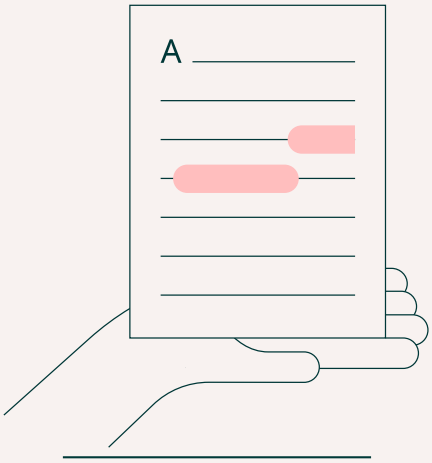
Звіт про оригінальність

● Оцінка схожості 13%

● Ризик плагіату НАЙВИЩИЙ

Ігор Кагало 2025-06-14 10:09

Посилання на звіт: 10bhy / Посилання користувача: qAHy



Ось вона –
Ваша звіт про оригінальність!

Ми раді повідомити, що перевірка вашого документа завершена, і результати вже готові! Наші алгоритми старанно працювали, щоб знайти збіги в наших базах даних.

На наступних сторінках ви знайдете результати перевірки:

Бали
Збіги
Посилання

Ваш документ було перевірено за такими джерелами:

- ✓ База даних інтернет-джерел
- ✓ База даних наукових статей
- ○ Глибока перевірка (наш вдосконалений алгоритм)

Бали

Збіги тексту	13%
Перефразування	0%
Цитований текст	0%
Неправильне цитування	0%
Збігів не знайдено	87%

Ризик плагіату НАЙВИЩИЙ

Ризик плагіату вказує, як збіги тексту розподілені по документу. Вищий ризик виникає, коли збіги з'являються близько один до одного, наприклад, у тому самому абзаці або розділі.

Оцінка схожості 13%

Оцінка схожості показує, скільки слів або символів у вашому документі збігаються з текстами інших документів, включаючи перефразовані тексти або неправильні цитати.

Збіги

ВСТУП

У сучасному світі інтенсивного розвитку та зростання обсягу інформаційного навантаження на студентів, особливої актуальності набувають інструменти персонального планування. З кожним роком навантаження тільки збільшується, тож необхідність в універсальному зручному додатку для легкого складання та редагування академічного розкладу, а також персонального розкладу дня, є необхідним та випереджаючим свій час рішенням. Адже використання одного мобільного додатку для організації навчального процесу дає змогу студентам ефективно керувати власним розкладом, уникати накладок у заняттях, а також зберігати важливі академічні події в одному зручному інтерфейсі. Тому тема розробки мобільного додатку для персонального студентського розкладу є актуальною як у технічному, так і в практичному аспекті.

Мета даного дипломного проєкту - розробка кросплатформенного мобільного застосунку для персонального планування розкладу студентів із використанням мови програмування JavaScript, фреймворку React Native та хмарного сервісу Firestore Firebase. Цей застосунок забезпечить зручний та зрозумілий інтерфейс, можливість додавання, редагування, видалення та фільтрації подій, а також підтримку повторюваних подій і кольорових категорій.

Для досягнення поставленої мети було визначено 39 наступні завдання: провести огляд сучасних мобільних технологій і фреймворків; обґрунтувати вибір інструментів розробки; спроєктувати інтерфейс додатку; реалізувати функціонал роботи з подіями розкладу (створення, редагування, повторення, фільтрація); протестувати програму на реальних пристроях; оцінити техніко-економічну доцільність впровадження 36 розробки.

36 Об'єктом дослідження є процес розробки мобільного додатку для організації особистого робочого розкладу студентів.

24 Предметом дослідження є програмні та інформаційні засоби, що забезпечують ефективну реалізацію функціоналу мобільного додатку для персонального планування.

24 У процесі дослідження були застосовані такі методи: аналізу і порівняння мобільних фреймворків, метод проєктування інтерфейсів користувача, моделювання даних у Firestore, методи програмної реалізації, тестування та візуальна валідація результатів.

Структура роботи складається з п’яти основних 30 розділів. У першому розділі розглянуто теоретичні основи мобільної розробки та кросплатформенних технологій. У другому — описано обґрунтування вибору технологій та архітектуру проєкту. Третій розділ присвячено безпосередній реалізації додатку: UI, логіка подій, кольори, повторення, фільтри. У четвертому розділі подано техніко-економічне обґрунтування проєкту. П’ятий розділ містить аналіз безпеки праці при роботі з комп’ютерною технікою.

Фактологічною основою дослідження стали офіційна документація React Native, Firebase, технічні статті та рекомендації щодо розробки мобільних додатків, власні спостереження і тестування додатку, а також результати реалізації функціоналу проєкту в середовищі Expo Go з використанням GitHub як системи контролю версій.

ЗАГАЛЬНІ ПОЛОЖЕННЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

Історія розвитку мобільних застосунків

Розвиток мобільних застосунків розпочався з появою перших мобільних телефонів на початку 90-х років XX століття. Перші застосунки були простими та обмеженими функціонально, наприклад, калькулятори, календарі та ігри типу «Змійка». Вони були попередньо встановлені на телефонах і не мали можливості завантаження або оновлення.

З появою смартфонів і 31 операційних систем, таких як Symbian, Windows Mobile, і згодом iOS та Android, мобільні застосунки почали набувати більшої популярності та функціональності. 2008 рік став переломним моментом у розвитку мобільних додатків, коли компанія Apple запустила App Store, а Google — Google Play (тоді Android Market). Ці платформи відкрили нові можливості для розробників, дозволивши їм створювати та розповсюджувати свої програми по всьому світу.

З того часу ринок мобільних застосунків стрімко зростав. Розробники почали використовувати нові технології, такі як GPS, камери, сенсори руху та інші можливості смартфонів, для створення більш складних та інтерактивних програм. Застосунки стали важливим інструментом у різних сферах життя, включаючи освіту, охорону здоров'я, розваги, бізнес і комунікації.

Сучасні мобільні застосунки використовують передові технології, такі як штучний інтелект, доповнена реальність, хмарні обчислення та багато іншого. Вони 14 стали невід'ємною частиною нашого повсякденного життя, забезпечуючи зручність і

доступність інформаційних та розважальних ресурсів у будь-який час та в будь-якому місці.

Одним з важливих напрямків розвитку мобільних застосунків є кросплатформенність. Завдяки використанню фреймворків, таких як React Native, Xamarin, Flutter та інших, 18 розробники можуть створювати програми, які працюють на різних операційних системах, зокрема iOS та Android, 18 з мінімальними змінами в коді. Це значно спрощує процес розробки та знижує витрати на підтримку та оновлення застосунків.

Таким чином, історія розвитку мобільних застосунків демонструє постійне вдосконалення технологій та розширення можливостей для користувачів. У майбутньому можна очікувати ще більших інновацій та інтеграції мобільних застосунків у всі сфери життя, що зробить їх ще більш важливими та незамінними інструментами в нашій повсякденній діяльності.

38 Типи мобільних додатків: нативні, гібридні, кросплатформенні

14 Мобільні додатки 14 стали невід'ємною частиною нашого повсякденного життя, забезпечуючи зручність і доступність інформаційних та розважальних ресурсів у будь-який час та в будь-якому місці. Одним з ключових аспектів 13 розробки мобільних додатків є вибір підходу до створення: нативний, гібридний або кросплатформенний. Кожен з цих підходів має свої переваги та недоліки, а також власні сфери застосування.

Нативні мобільні додатки розробляються для певної операційної системи (iOS, Android) з використанням відповідних інструментів та мов програмування. Наприклад, для розробки нативних додатків для iOS використовують мову програмування Swift або Objective-C, а для Android — Java або Kotlin.

Переваги нативних додатків:

Висока продуктивність і швидкість роботи: Нативні додатки мають прямий доступ до апаратних ресурсів пристрою, що забезпечує швидкий час відповіді та плавну роботу;

Покращений користувацький досвід: Нативні додатки можуть повністю відповідати дизайну та функціональності операційної системи, що робить їх більш зручними та інтуїтивно зрозумілими для користувачів;

Доступ до всіх функцій пристрою: Нативні додатки можуть використовувати всі можливості пристрою, такі як камера, GPS, сенсори руху та інше.

Недоліки нативних додатків:

Високі витрати на розробку: Розробка нативних додатків для декількох операційних

систем вимагає великих витрат на час і ресурси.

Триваліший час розробки: 13 Розробка окремих додатків для кожної платформи може займати більше часу, ніж інші підходи.

Гібридні мобільні додатки поєднують елементи нативних та веб-додатків. Вони розробляються за допомогою веб-технологій (HTML, CSS, JavaScript) і вбудовуються в нативні контейнери, що дозволяє їм працювати 13 на різних операційних системах. Найпопулярнішими фреймворками для створення 11 гібридних додатків є Ionic, PhoneGap і Apache Cordova. В таблиці 1.1 наведено основні 11 переваги та недоліки гібридних додатків.

11 Таблиця 11 1.1 - 11 Переваги та недоліки гібридних додатків

11 Переваги гібридних додатків

11 Недоліки гібридних додатків

Зменшені витрати на розробку — один код на всі платформи

Нижча продуктивність — особливо при складних обчисленнях або графіці

Швидкий цикл розробки — оновлення без перевипуску додатків

Обмежений доступ до функцій пристрою — деякі нативні API можуть бути недоступні

Спрощене тестування та підтримка

Можливі візуальні або поведінкові відмінності між платформами (iOS / Android)

Використання веб-технологій (HTML, CSS, JS)

Залежність від фреймворків і обгорток , які можуть мати обмеження чи баги

Кросплатформенні мобільні додатки дозволяють розробникам створювати додатки для кількох операційних систем одночасно, використовуючи одну базу коду. Найпопулярнішими фреймворками для кросплатформенної розробки є React Native, Flutter і Xamarin.

Переваги кросплатформенних додатків:

Економія часу та ресурсів: Розробка єдиного додатку для всіх платформ знижує витрати на розробку та підтримку;

Швидке оновлення: Оновлення та виправлення помилок можуть бути швидко

застосовані для всіх платформ одночасно.

Недоліки кросплатформенних додатків:

Обмеження в продуктивності: Кросплатформенні додатки можуть бути менш продуктивними, ніж нативні, оскільки вони не завжди мають повний доступ до апаратних ресурсів пристрою;

Можливі проблеми з інтеграцією: Іноді можуть виникати проблеми з інтеграцією специфічних функцій платформи.

Вибір типу мобільного додатку залежить від конкретних вимог проєкту, бюджету, 9 часу та ресурсів. Нативні додатки забезпечують високу продуктивність та найкращий користувацький досвід, але потребують більше 9 часу та ресурсів на розробку. Гібридні та кросплатформенні додатки дозволяють швидше та дешевше створювати додатки для кількох платформ, але можуть мати обмеження у продуктивності та доступі до функцій пристрою.

Загалом, сучасні підходи 9 до розробки мобільних додатків забезпечують широкі можливості для створення ефективних та зручних рішень, що відповідають потребам користувачів. Технічний прогрес продовжує вдосконалювати інструменти та технології, що дозволяє розробникам знаходити оптимальні шляхи для реалізації своїх ідей та проектів.

Сучасні фреймворки 9 для розробки 17 мобільних додатків

17 Розробка мобільних додатків сьогодні є однією з найважливіших галузей IT-індустрії. З огляду на стрімкий розвиток технологій, розробники мають великий вибір інструментів та фреймворків, які дозволяють створювати якісні додатки 15 для різних платформ. У цій статті ми розглянемо найпопулярніші сучасні фреймворки 17 для розробки мобільних додатків, їхні переваги та недоліки, а також порівняємо їх між собою.

React Native – це один з найпопулярніших 9 фреймворків для розробки кросплатформенних мобільних додатків, розроблений компанією Facebook. Його основна перевага полягає в тому, що розробники можуть використовувати мову програмування 27 JavaScript 9 для створення додатків для iOS та Android. 27 React Native надає змогу створювати додатки, які мають продуктивність і зовнішній вигляд, схожий на нативні додатки, завдяки доступу до нативних компонентів та API.

Переваги фреймворку ReactNative:

Швидкість розробки завдяки використанню JavaScript та багатої бібліотеки

компонентів;

Активна спільнота розробників, яка постійно вдосконалює фреймворк;

Можливість легкого інтегрування з нативними модулями.

Недоліки фреймворку ReactNative:

Обмежена продуктивність порівняно з нативними додатками;

15 Проблеми з підтримкою деяких специфічних функцій платформ.

32 Flutter – це фреймворк, розроблений компанією Google, який 27 використовує мову програмування Dart. На відміну від React Native, Flutter компілюється безпосередньо в нативний код, що дозволяє створювати додатки з високою продуктивністю та плавною анімацією. Однією з головних переваг Flutter є його власний набір віджетів, 6 які дозволяють створювати красиві та інтерактивні інтерфейси користувача.

Переваги фреймворку Flutter:

Висока продуктивність завдяки компіляції в нативний код;

Широкий набір віджетів для створення інтерактивних UI;

Єдина кодова база для iOS та Android.

Недоліки фреймворку Flutter:

Відносно новий фреймворк, що 6 може призвести до нестачі деяких бібліотек та плагінів;

Необхідність вивчення мови програмування Dart.

Xamarin – це фреймворк, розроблений компанією Microsoft, який дозволяє створювати кросплатформенні додатки на мові програмування C#. Xamarin має два основних підходи: Xamarin.Forms 15 для створення інтерфейсу користувача, який можна спільно використовувати між платформами, та Xamarin.iOS та Xamarin.Android для створення платформенно-специфічного коду.

Переваги фреймворку Xamarin:

Можливість використовувати C# 6 для розробки додатків;

6 Повний доступ до нативних API та SDK;

Інтеграція 21 з іншими продуктами Microsoft, такими як Azure.

21 Недоліки фреймворку Xamarin:

Великий розмір додатків через необхідність включення Mono runtime;

Можливі проблеми з продуктивністю на складних інтерфейсах користувача.

З нативних фреймворків одними з найпопулярніших є Swift та Kotlin.

6 Swift – це мова програмування, розроблена компанією Apple для створення додатків під iOS, macOS, watchOS та tvOS. Swift є наступником Objective-C та має більш сучасний синтаксис, що робить його простішим у вивченні та використанні. Додатки, створені за допомогою Swift, мають високу продуктивність та можуть використовувати всі можливості платформи iOS.

Переваги фреймворку Swift:

Висока продуктивність та ефективність;

Повний 19 доступ до всіх нативних функцій та бібліотек iOS;

Сучасний та 21 простий у використанні синтаксис.

Недоліки фреймворку Swift:

Можливість розробки лише під платформи Apple;

23 Необхідність вивчення нової мови програмування.

23 Kotlin 16 – це мова програмування, розроблена компанією JetBrains, яка є основною мовою 6 для розробки додатків під Android. Kotlin є більш сучасною та безпечною альтернативою Java, яка дозволяє розробникам писати менш громіздкий та 19 менш схильний до помилок код. Додатки, створені за допомогою Kotlin, мають високу продуктивність та можуть використовувати всі можливості платформи Android.

Переваги фреймоврку Kotlin:

Сучасний та виразний синтаксис;

Безпека від null-посилань, що зменшує кількість помилок;

16 Повна сумісність з Java, що дозволяє використовувати 16 існуючий код.

Недоліки фреймворку Kotlin:

23 Необхідність вивчення нової мови програмування;

23 Менша кількість ресурсів для навчання порівняно з Java.

Вибір фреймворку для розробки мобільного додатку залежить від багатьох чинників, таких як вимоги проєкту, час, бюджет, та технічні обмеження. Кожен з розглянутих 28 фреймворків має свої переваги та недоліки, тому важливо ретельно оцінити всі можливості перед тим, як приймати рішення.

Кросплатформенні фреймворки, такі як React Native, Flutter та Xamarin, дозволяють швидко створювати додатки для кількох платформ одночасно, що економить час та ресурси. Проте, вони можуть мати певні обмеження у продуктивності та доступі до нативних функцій. Нативні фреймворки, такі як Swift та Kotlin, забезпечують високу продуктивність та 19 доступ до всіх можливостей платформи, але потребують більше часу та ресурсів на розробку.

У кінцевому рахунку, 28 вибір фреймворку залежить від конкретних потреб та умов проєкту. Технічний прогрес продовжує вдосконалювати існуючі інструменти та створювати нові, що дає розробникам безліч можливостей для реалізації своїх ідей. Незалежно від обраного підходу, головне – створювати 29 ефективні та зручні рішення, які задовольняють потреби користувачів.

Розробка кросплатформенних додатків на JavaScript та її особливості

У сучасному світі, де мобільні та десктопні додатки охоплюють більшість сфер людської діяльності — від освіти до медицини та фінансів — постає необхідність у створенні програмного забезпечення, яке можна 33 легко запускати на різних операційних системах. Традиційний підхід передбачає написання окремих програм для Android, iOS, Windows, macOS та Linux. Проте це потребує більше ресурсів, часу та команди розробників зі знанням різних мов програмування. Вирішенням цієї проблеми стала кросплатформенна розробка, зокрема за допомогою мови JavaScript та пов'язаних із нею фреймворків.

JavaScript, який із самого початку був мовою скриптів для браузерів, еволюціонував до одного з найпотужніших інструментів для створення веб-, мобільних та навіть десктопних застосунків. Завдяки поєднанню JavaScript з такими фреймворками як React Native, Ionic, Electron, NativeScript та іншими, стала можливою ефективна кросплатформенна розробка.

JavaScript — це мова програмування високого рівня з динамічною типізацією, подієво-орієнтованою парадигмою та широкою підтримкою асинхронних операцій. Його основні переваги в кросплатформенній розробці: універсальність — один і той самий

код може працювати на Android, iOS, Windows, Linux, macOS; велика екосистема — наявність численних бібліотек і фреймворків, які значно прискорюють розробку; велика спільнота — 5 велика кількість прикладів, документації, рішень на Stack Overflow; простота навчання — порівняно легка у вивченні, особливо для студентів технічних спеціальностей.

Основні фреймворки для кросплатформенних додатків на JavaScript – це ReactNative, Electron, Ionic.

React Native — один із найпопулярніших фреймворків, створений компанією Facebook у 2015 році. Він дозволяє 4 створювати нативні мобільні додатки для iOS та Android, використовуючи один спільний код на JavaScript та JSX. Його особливості включають компонентний підхід до створення інтерфейсу, гаряче оновлення (hot reload) для зручної розробки, а також доступ до нативних API через бридж між JavaScript та платформою.

Electron — це 5 фреймворк для створення десктопних кросплатформенних застосунків з використанням HTML, CSS і JavaScript. Створений компанією GitHub, Electron дає змогу створити застосунок один раз, і запускати його на Windows, macOS і Linux. Типові приклади застосунків на Electron — це Visual Studio Code, Slack (десктопна версія) та Discord. Electron має такі переваги як доступ до системного API, повноцінний Node.js у середовищі розробки, та єдиний код для всіх операційних систем.

Ionic — ще один фреймворк 4 для розробки мобільних застосунків. Його особливість — орієнтація на гібридні застосунки: додатки працюють у webview всередині оболонки. Він працює на основі HTML, CSS, Angular або React. Може використовувати Capacitor або Cordova 4 для доступу до нативних функцій і є придатним для швидкого створення прототипів.

Серед основних переваг кросплатформенних додатків варто виділити: єдиний код для декількох платформ, менші витрати на команду розробників, швидший час виводу продукту на ринок та простоту в обслуговуванні. Проте існують і певні недоліки: обмеження у продуктивності (особливо для графічно-інтенсивних застосунків), більший розмір застосунку (особливо у випадку Electron), проблеми з доступом до специфічних нативних функцій, а також додатковий рівень складності при налагодженні через використання бриджів.

Приклади застосування на практиці.Кросплатформенні JavaScript-рішення особливо популярні серед стартапів, студентських проєктів і малих підприємств. Наприклад, MyPlannerApp — студентський застосунок для керування розкладом, створений з використанням React Native та Firebase. Також тестувальні програми на Electron застосовуються для створення навчальних середовищ, лабораторій, інтерфейсів до

апаратних систем. Гібридні програми Ionic використовуються для швидкого створення MVP (мінімально життєздатного продукту).

ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ РОЗРОБКИ

Одним із головних завдань при розробці програмного забезпечення є вибір спеціальних засобів, які б полегшили та скоротили час розробки. Для реалізації додатку було використано редактор коду Visual Studio Code, який має зручний інтерфейс та багато плагінів, які спрощують роботу при розробці проєктів. Великим плюсом VSCode є зручне використання Git та Node.js.

Додаток для мобільного телефону був розроблений на фреймворкові React Native, основою якого є мова програмування JavaScript та який має базу звичайного React. В проєкті було використано не звичайний React Native ,а набір інструментів Expo, який містить готові конфігурації Android Studio / XCode , надає змогу управляти сертифікатами в Apple & Google та push-повідомлень

Мова програмування JavaScript

JavaScript було створено для того, щоб “оживити вебсторінки”. Програми цією мовою називаються скриптами. Їх можна писати прямо в коді HTML-сторінок і вони автоматично виконуватимуться при їх завантаженні. Скрипти записуються та виконуються як простий текст. Для запуску їм не потрібна спеціальна підготовка чи компілятор. У цьому плані JavaScript дуже відрізняється від іншої мови програмування — Java.

Сьогодні JavaScript може виконуватися не тільки в браузері, але й на сервері або на будь-якому пристрої, який має спеціальну програму — рушій JavaScript. Браузер має вбудований рушій, який інколи називають “віртуальною машиною JavaScript”. Різні рушії мають різні “кодові назви”. Наприклад:

V8 – в Chrome, Opera та Edge;

SpiderMonkey – в Firefox;

...Є також інші кодові назви як “Chakra” для IE, “JavaScriptCore”, “Nitro” і “SquirrelFish” для Safari, та інші.

Рушії складні. Але принцип роботи простий:

Рушій (вбудований, якщо це браузер) читає (“розбирає”) скрипт;

Потім він перетворює (“компілює”) скрипт у машинний код;

20 Після чого цей 2 машинний код виконується, причому досить швидко.

2 Рушій застосовує оптимізації на кожному етапі процесу. Він навіть слідкує за скомпільованим скриптом під час його виконання, аналізуючи 20 дані, що проходять через нього, 2 і оптимізує машинний код, зважаючи 20 на ці знання.

2 JavaScript — це кросплатформенна, об’єкто-орієнтована скриптова мова програмування,яка незаймає багато місйя. Всередині середовища виконаеея 3 JavaScript може бути пов’язаний з об’єктами даного середовища і надавати програмний контроль над ними.

3 JavaScript включає 7 стандартну бібліотеку об’єктів, наприклад, Array, Date і Math, а також базовий набір елементів, наприклад, оператори та контролюючі конструкції. Ядро JavaScript може бути розширено для різних цілей шляхом додавання в нього нових об’єктів, наприклад:

Клієнтський JavaScript 3 розширює ядро мови, надаючи об'єкти для управління 12 браузером та його об'єктною моделлю документа (DOM). Наприклад, клієнтські розширення дозволяють програмі розміщувати елементи в HTML-формі та 3 обробляти події користувача, такі як клацання, введення даних у форму та навігація по сторінках;

3 JavaScript на стороні сервера розширює ядро мови, надаючи об'єкти для запуску JavaScript на сервері. Наприклад, розширення на стороні сервера дозволяє програмі підключатися 3 до бази 3 даних, забезпечувати безперервність інформації між викликами з програми або виконувати маніпуляції файлами на сервері.

3 Однією з головних причин вибору 12 мови програмування JavaScript для розробки мобільного додатку MyPlannerApp є її універсальність та підтримка кросплатформенних рішень. JavaScript є основою фреймворку React Native, який 37 дозволяє розробляти мобільні додатки одразу для 4 Android та iOS з єдиною кодовою базою. Це 37 значно скорочує витрати часу на підтримку, тестування та оновлення додатку, оскільки відсутня потреба розробляти окремі версії на Swift або Kotlin.

JavaScript має величезну екосистему бібліотек, інструментів та спільноти, що активно підтримує розвиток технологій, пов’язаних з фронтендом і мобільною розробкою. Це дозволило легко інтегрувати в проєкт готові компоненти, такі як DateTimePicker, AsyncStorage, Expo Go, Firebase SDK та інші. Велика кількість документації, прикладів коду та відеоуроків також полегшила процес освоєння і впровадження окремих частин функціоналу додатку.

Серед інших переваг JavaScript — його гнучкість і динамічність. Це дозволяє швидко будувати прототипи, змінювати структуру даних, додавати нові компоненти без

необхідності жорсткої типізації чи складного компілювання, як у мовах низького рівня. Завдяки цьому, реалізація логіки повторюваних подій, динамічної фільтрації 4 та інтеграції з Firestore виконувалась швидко й ефективно.

Крім того, мова JavaScript є універсальною в екосистемі веб- і мобільної розробки. Її знання дозволяє легко адаптувати додаток до інших платформ, наприклад — створити вебверсію розкладу або додаток для настільних ОС через Electron. Це дає широкі перспективи для масштабування проєкту в майбутньому.

Отже, вибір JavaScript для реалізації мобільного застосунку MyPlannerApp є цілком обґрунтованим з технічної, економічної та стратегічної точок зору. Ця мова забезпечує швидкість розробки, кросплатформенність, багату екосистему, легку інтеграцію з сучасними інструментами та хорошу перспективу на майбутнє розширення функціоналу.

Фреймворк ReactNative

React Native було створено на основі React — фреймворк, який дозволяє розробляти кросплатформені мобільні додатки з використанням JavaScript.

Фреймворк React (або React.js) — це одна з найпопулярніших JavaScript-бібліотек для створення інтерфейсів користувача, розроблена компанією Meta (раніше Facebook). Вона використовує компонентний підхід, що дозволяє будувати складні UI-структури з окремих ізольованих блоків, які легко повторно використовуються. React працює з віртуальним DOM, що підвищує продуктивність оновлення сторінки, та підтримує 4 декларативний стиль програмування, завдяки якому код стає читабельнішим та легше підтримується.

Головною особливістю React Native є те, що компоненти інтерфейсу не рендеряться у веб-браузері, як у React.js, а трансформуються у нативні компоненти 4 Android та iOS. Це дозволяє створювати повноцінні мобільні додатки, що за виглядом та швидкодією не відрізняються від нативних.

4 React Native використовує ті ж принципи, що й React — компоненти, пропси, стан, обробка подій, що дозволяє легко переходити від веброзробки до мобільної. Окрім цього, фреймворк підтримує гаряче перезавантаження (hot reload), що значно пришвидшує розробку, дозволяючи бачити зміни майже миттєво без повного перезапуску додатку.

1 Facebook, Instagram, Skype, Pinterest, Uber і інші великі компанії використали цей фреймворк для розробки своїх мобільних додатків за допомогою цієї технології. Дуже величезним плюсом є використання однієї бази ,писати один код, а отримати

результат, який буде працювати відразу на двох 4 платформах, iOS і Android. Цей 1 фреймворк швидко розвивається та покращується , підтримується компаніями Facebook та Microsoft. Саме головне ,що цей продукт поширюється під ліцензією відкритого програмного забезпечення ,що дозволяє повністю бескоштовно реалізувати свої мобільні нативні додатки

1 У дипломному проєкті MyPlannerApp використання React Native дозволило реалізувати мобільний додаток для студентського розкладу швидко, зручно та ефективно. Його інтеграція з Expo (оболонкою для швидкого розгортання додатків без складного налаштування Android/iOS-середовища) дозволила обійти складності з білдами, тестуванням та нативним SDK.

Expo як інструмент для розробки кросплатформенних додатків

Expo — це набір інструментів і сервісів, 5 який спрощує розробку мобільних додатків за допомогою React Native. Він забезпечує швидкий старт проєкту, зменшує залежність від нативного коду та дозволяє розробникам зосередитися на функціоналі та інтерфейсі додатку без глибокого занурення у нативну інфраструктуру Android чи iOS. Expo надає стандартну структуру проєкту та готовий набір API 5 для роботи з камерою, геолокацією, файловою системою, push-нотифікаціями та іншими функціями пристрою. На рисунку 2.1 показано загальний вигляд додатку ExpoGO.

Рисунок 1.1 – Expo Go 34 мобільний додаток

34 Однією з ключових переваг Expo є можливість миттєво тестувати додаток на реальному пристрої за допомогою застосунку Expo Go, доступного в Google Play та App Store. Це дозволяє запускати додаток без компіляції або встановлення Android Studio чи Xcode. Кожна зміна в коді миттєво синхронізується з пристроєм, що значно прискорює процес розробки.

Expo також пропонує інструменти для створення фінального білду (файлів .apk або .aab для Android та .ipa для iOS) за допомогою хмарної платформи EAS (Expo Application Services). Це дозволяє уникнути складних налаштувань середовища розробки, а процес збірки виконується на серверах Expo, що особливо зручно для новачків та команд 4 з обмеженими ресурсами.

4 У проєкті MyPlannerApp Expo стало основною платформою розробки завдяки своїй простоті, стабільності та гнучкості. З його допомогою було реалізовано швидкий старт проєкту, протестовано всі ключові функції (додавання подій, фільтрація, повторювання, кольорові категорії), а також забезпечено стабільну роботу додатку на різних пристроях без необхідності окремого налаштування під 4 Android та iOS. Таким чином, Expo суттєво прискорив розробку і зробив її зручною та ефективною.

Хмарна платформа Firebase для зберігання даних та синхронізації

Firebase — це хмарна платформа від компанії Google, яка надає набір сервісів для створення сучасних веб- і мобільних додатків. Одним із ключових компонентів Firebase є Cloud Firestore — гнучка, масштабована NoSQL-база даних у реальному часі, яка дозволяє зберігати та синхронізувати дані між користувачами та пристроями миттєво. Firestore забезпечує безперервну роботу додатку, навіть якщо пристрій користувача тимчасово втрачає з'єднання з інтернетом.

Основною перевагою Firestore є її реактивність: усі зміни в базі даних миттєво відображаються у користувацькому інтерфейсі завдяки використанню спостерігачів (onSnapshot). Це дозволяє реалізовувати динамічні екрани, де нові події, редагування або видалення відображаються без потреби ручного оновлення. Крім того, Firestore підтримує структуроване зберігання даних у вигляді колекцій і документів, що є зручним для організації розкладу подій. Приклад таблиці бази даних показано на рисунку 2.2.

Рисунок 2.2 – Приклад таблиці бази даних в Cloud Firestore

У проєкті MyPlannerApp Firebase використовується як основне сховище подій, категорій та налаштувань користувача. За допомогою Firestore реалізовано збереження даних про події розкладу (назва, дата, час, тип, колір, повторюваність тощо), можливість додавання та редагування подій у реальному часі, а також синхронізація даних на різних пристроях. Таким чином, Firebase значно спростив реалізацію бекенд-функцій без потреби створювати власний сервер.

Ще одним важливим аспектом є легкість інтеграції Firebase з React Native. Завдяки офіційному SDK для JavaScript і зручній документації, підключення платформи до додатку відбулося без ускладнень, а налаштування безпеки й авторизації можливо реалізувати за допомогою правил доступу у Firestore Security Rules. Це дозволяє забезпечити захист персональних даних студентів і контроль над правами доступу до подій.

Отже, використання Firebase у даному проєкті дозволило реалізувати зберігання та обробку даних швидко, надійно і масштабовано. Завдяки цьому платформа стала ключовим інструментом для забезпечення стабільної роботи додатку MyPlannerApp.

Середовище розробки додатків

Для написання додатку MyPlannerApp було використано редактор коду Visual Studio Code. Visual Studio Code — засіб для створення, редагування сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code є повністю безкоштовною і

працює на Windows, Linux і OS X. За основу Visual Studio Code Microsoft використала напрацювання вільного проекту Atom, що розвивається компанією GitHub. Редактор містить інструменти 5 для роботи з Git 1 і засоби рефакторингу, а також навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки.

1 Графічний інтерфейс програмного середовища Visual Studio Code див. на рисунку 2.3.

Рисунок 2.3 – Інтерфейс VisualStudio

1 Редактор VSCode має дуже багато розширень редактора, які полегшують життя розробника. Під час розробки були використані такі розширення: Prettier – полегшує читабельність коду в редакторі, дозволяє вільно писати код та під час зберігання приводить його в нормальний вигляд. Expo Tools – при роботі з файлами конфігурацій допомагає автоматично підтягнути потрібний пакет під час початку введення імені пакету. Simple 1 React Snippets – містить готові сніпети та команди, які використовуються в React

1 2.6Система контролю версій 8 Git та GitHub

8 Git — це розподілена система контролю версій, яка дозволяє ефективно керувати змінами у програмному коді. Вона є стандартом у 22 сфері 8 розробки програмного забезпечення, оскільки 22 забезпечує збереження історії змін, можливість повернення до попередніх версій, а також спільну роботу над кодом в команді. 10 Git працює локально на комп'ютері розробника і зберігає всі коміти, гілки та структуру проєкту, що робить його особливо зручним для незалежної та автономної роботи без постійного підключення до інтернету.

10 GitHub — це онлайн-сервіс для хостингу репозиторіїв, побудований на основі Git. Він дозволяє зберігати, синхронізувати та поширювати код у хмарі, а також забезпечує інструменти для відстеження помилок, управління задачами, рев'ю коду та співпраці 10 з іншими розробниками. Для індивідуальних проєктів, таких як MyPlannerApp, GitHub виступає як надійне сховище резервних копій та точка централізованого контролю змін. Загальний вигляд середовища GitHub показано на рисунку 2.4.

Рисунок 2.4 – Вигляд проекту в середовищі контролю версій GitHub

У процесі розробки додатку MyPlannerApp система Git використовувалася 8 для контролю версій на локальній машині, з частими комітами після кожного значущого етапу: створення інтерфейсу, налаштування Firebase, реалізація логіки повторювання подій тощо. GitHub було використано для віддаленого збереження проєкту, ведення історії змін, а також як засіб резервного копіювання у випадку втрати даних на

локальному комп’ютері.

РОЗРОБКА ТА ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ “MYPLANNERAPP”

Загальна архітектура додатку та основні функції

З урахуванням потреб студентів в плануванні свого особистого робочого графіку, я розробила мобільний додаток MyPlannerApp. Архітектура додатку побудована на логічних компонентах розділених за відповними їм функціями, що взаємодіють через спільну базу даних Firestore.

На рисунку 3.1 представлено структурну схему архітектури мобільного додатку MyPlannerApp, яка відображає ключові компоненти системи, їхню взаємодію між собою та з джерелами збереження даних.

Рисунок 3.1 Структурна схема архітектури додатку

Така архітектура дозволяє забезпечити гнучкість, масштабованість і розмежування відповідальностей між функціональними модулями. Компоненти інтерфейсу користувача, як-от головний екран, календар та екран налаштувань, тісно пов’язані з модальними вікнами для створення та перегляду подій, а також з базою даних Firestore та локальним сховищем AsyncStorage.

Користувач взаємодіє з додатком через головний інтерфейс, реалізований у файлі App.js, де також налаштовано навігацію між основними екранами. Один із таких екранів — HomeScreen.js — відповідає за відображення всіх запланованих подій. З цього екрана користувач може створити нову подію або редагувати наявну через спеціальне модальне вікно CreateEditEventModal, яке у свою чергу зчитує та зберігає дані у Firestore. Також з головного екрана можливий перехід до EventDetailModal, який надає детальний огляд події та дозволяє її редагування або видалення. Для синхронізації подій із Firestore використовується підписка на зміни через onSnapshot, що дозволяє оперативно оновлювати інтерфейс при зміні даних у хмарі.

Другий ключовий екран — CalendarScreen.js — відповідає за календарне відображення подій. Усі події, що відповідають вибраному часовому проміжку, також зчитуються з Firestore у реальному часі. Аналогічно до головного екрана, натискання на подію відкриває EventDetailModal, що дозволяє переглядати та змінювати деталі події. Таким чином, як календар, так і головний екран спираються на один і той самий механізм перегляду й взаємодії з подіями.

Третім основним компонентом є екран налаштувань — SettingsScreen.js. Його основна функція полягає у збереженні фільтрів та керуванні кольоровими категоріями подій. Усі

налаштування зберігаються у локальне сховище AsyncStorage, що дозволяє уникнути додаткових запитів до хмарного сховища при завантаженні додатку. Ці налаштування, зокрема фільтри за періодом і категоріями, згодом зчитуються на головному екрані та у календарі, що забезпечує персоналізований підхід до відображення подій.

Модальні вікна, такі як CreateEditEventModal та EventDetailModal, є спільними компонентами, які використовуються на різних екранах. Вони мають доступ як до Firestore для запису, оновлення чи видалення подій, так і до локальних налаштувань, якщо потрібно зчитати категорії кольорів або інші параметри. Вся інформація про події зберігається у Firestore в колекціях `events` та `categories`, а інформація про фільтри, сортування та обрані категорії зберігається в AsyncStorage.

У цілому, ця архітектура дозволяє реалізувати зручний, реактивний інтерфейс, який динамічно реагує на зміни 1 в базі даних та діє згідно з локальними вподобаннями користувача, зберігаючи високу продуктивність додатку.

Додаток реалізовано засобами мови JavaScript, фреймворка ReactNative та платформи Expo для кросплатформенної розробки. 22 В процесі розробки було використано найсучасніші технології. Додаток створено з метою полегшити навчальний та робочий процес студентів. Додаток MyPlannerApp має стати гнучким та універсальним програмним рішенням однієї з ключових проблем студентів – планування дня. З цим додадток надзвичайно швидко та просто можна розпланувати свій день, тиждень та навіть місяць.

Основні функції :

Створення нової події ;

Додавання нової події ;

Редагування події;

Видалення ;

Відображення подій у вигляді календаря на день/3дні/тиждень/місяць;

Фільтрація подій за періодом день/3дні/тиждень/місяць;

Сотрування подій за датою та назвою;

Налаштування кольорових категорій .

Тобто функціонал додатку повністю забезпечує мету користувача в плануванні свого розкладу дня, що є важливим для студентів, тим не менше додаток є універсальним,

тому ним можуть користуватися як в тому числі і викладачі, так і учні, та будь-хто, хто зацікавлений в ефективному плануванні.

Структура додатку показана 1 на рисунку 3.2.

1 Рисунок 3.2 – Структура додатку MyPlannerApp

Основні директорії з компонентами - це screens/, components/, styles/, firebase/.Папка screens/ містить файли HomeScreens.js, CalendarScreens.js та SettingsScreen.js.

HomeScreens.js – це головний екран, в ньому здійнюється 10 всі основні функції додатку. А саме відображаються події в вигляді карток, здійснюється додавання нової події. Також тут можна детально оглянути події, також їх 1 редагувати або видалити.

1 На екрані CalendarScreen.js відображаються події на календарній сітці. В залежності від опції, яка вибирається в налаштуваннях, календар може відображатися в 4 режимах. Режим «День» – показує всі події які заплановані на протязі сьогоднішнього дня. Режим «3 дні» - показує всі події на протязі 3 найближчих днів.Режим «Тиждень» показує події на протязі цього тижня, а режим «місяць» - на протязі цього місяця.

На екрані налаштувань SettingsScreen.js можна легко та зручно відфільтрувати та сортувати події на головному екрані, вибрати режим відображення календаря, а також тут можна редагувати та видаляти кольорові категорії.

В components/ винесені всі повторювані компонент. CreateEventModal.js — модальне вікно додавання події. EditEventModal.js — редагування події. EventDetailModal.js — перегляд і керування . EventCard.js — відображення у вигляді карток на головному екрані HomeScreen. EventForm.js — спільна логіка форми (дата, час, категорії, повторення).

В Firebase/config.js відбувається ініціалізація конфігураційний файлів Firebase та Firestore та в Navigation/AppNavigator.js – конфігурація маршрутизації та взаємодії між екранами Home, Calendar та Settings.

В директорію styles/ було винесено основні стилі для оптимізації коду, 10 а також для покращення його читабельності. Отже, тут знаходяться стилі для CreateEventModal в файлі addEventModalStyles.js, стилі для календаря – calendarStyles.js, стилі для вікна детального перегляду – eventDetailModalStyles.js, і стилі для екрану налаштувань – settungsStyles.js. App.js, package.json, package-lock.json – все це конфігураційні файли Expo та npm.

Взаємодію компонентів показано на рисунку 3.3 де показана блок-схема взаємодії компонентів при створенні або редагуванні події.

Рисунок 3.3 – Блок-схема взаємодії компонентів при додаванні або редагуванні події

Схема відображає логічну послідовність дій — від взаємодії користувача з інтерфейсом до збереження інформації в хмарну базу даних Firestore. Всі компоненти використовують реактивний підхід: після змін у Firestore інтерфейс автоматично оновлюється завдяки використанню підписки onSnapshot.

Структура даних у Firestore зберігається у вигляді колекцій events, яка зберігає події користувача. Типова структура одного документа:

{

"id": "event_id",

"title": "Лекція з алгебри",

"type": "Лекція",

"auditory": "ауд. 101",

"groupOrTeacher": "KI-21",

"note": "Контрольна робота",

"startTime": "2025-06-01T10:00:00",

"endTime": "2025-06-01T11:30:00",

"repeat": "weekly",

"repeatInterval": 1,

"repeatCount": null,

"repeatUntil": null,

"repeatGroupId": "uuid-1234",

"colorCategoryId": "color-abc"

}

Алгоритм взаємодії користувача з мобільним додатком

Перед створенням будь-якого мобільного додатку важливо чітко визначити сценарії взаємодії користувача з інтерфейсом та системними компонентами. Це дозволяє не

лише покращити зручність користування, але й забезпечити логічну послідовність дій, узгоджену з архітектурою проєкту. В контексті розробки додатку MyPlannerApp така схема стала основою для реалізації функціоналу, що охоплює роботу з подіями, їх повторюваність, збереження та перегляд.

Алгоритм, представлений у вигляді блок-схеми, є базою для реалізації послідовної й надійної логіки у мобільному застосунку. Це забезпечує не лише простоту взаємодії для кінцевого користувача, а й спрощує подальшу підтримку та розширення функціоналу системи.

На рисунку 3.4 представлено блок схему програмного коду додатку, вона демонструє основні етапи взаємодії користувача з інтерфейсом, починаючи з моменту запуску й завершуючи діями з подіями — створенням, редагуванням або видаленням.

Рисунок 3.4 – Блок-схема програмного коду

Після запуску додатку (`App.js`), система переходить до завантаження користувацьких налаштувань. На цьому етапі ініціалізуються фільтри, які задаються вручну або автоматично зчитуються з локального сховища `AsyncStorage`. Вони можуть включати обраний часовий період, кольорову категорію подій, формат відображення (день, тиждень, місяць) та інші критерії. Введення або вибір фільтрів відбувається через екран налаштувань `SettingsScreen.js`, після чого застосовані параметри впливають на подальший відбір інформації 1 з бази даних.

1 Наступним кроком є виконання запиту до Firestore — хмарного сховища, де зберігаються всі створені події. Додаток зчитує лише ті події, які відповідають критеріям фільтрації. Це дозволяє уникати надмірного трафіку та забезпечує швидке завантаження інтерфейсу. Завантажені події виводяться на головний екран `HomeScreen.js` або на екран календаря `CalendarScreen.js` — залежно від вибору користувача. Відображення реалізовано у вигляді карток або блоків подій у стилі Google Calendar.

Користувач має можливість взаємодіяти з кожною відображеною подією. 1 При натисканні на подію відкривається модальне вікно детального перегляду (`EventDetailModal.js`), у якому міститься повна інформація: назва, час початку і завершення, тип заняття, викладач або група, аудиторія, примітка та колір категорії. Із цього вікна користувач може або вийти без змін, або натиснути кнопку редагування чи видалення події.

Якщо вибрано редагування, відкривається окреме модальне вікно (`EditEventModal.js`), у якому користувач вносить зміни до полів події. У разі, якщо подія є частиною серії (тобто має параметр `repeatGroupId`), додаток дозволяє вибрати — редагувати лише

обрану подію чи всю серію. Після підтвердження змін, Firestore оновлює відповідний запис (або записи, якщо обрано серію). У випадку видалення події діє аналогічний підхід: користувач може видалити лише одну подію або всі пов'язані події, що мають однаковий `repeatGroupId`. Зміни відразу синхронізуються з базою Firestore, і відповідно оновлюється інтерфейс.

Інша гілка логіки реалізується у разі натискання на кнопку «Додати подію». Тоді відкривається модальне вікно `CreateEventModal.js`, у якому користувач заповнює інформацію про нову подію. Одним із ключових моментів при створенні є визначення режиму повторення: якщо користувач обирає `Repeat = "never"`, створюється лише одна подія. У протилежному випадку, коли подія має бути повторюваною (наприклад, щотижня або щомісяця), генерується серія подій із однаковим `repeatGroupId`, що дозволяє надалі групово їх обробляти. Усі створені події зберігаються у Firestore, після чого автоматично підвантажуються у відповідні компоненти інтерфейсу та відображаються на головному екрані або в календарі.

Таким чином, розроблений алгоритм дозволяє забезпечити зручну та інтуїтивно зрозумілу 22 взаємодію користувача з подіями, у тому числі їх фільтрацію, перегляд, редагування, видалення та створення. Уся взаємодія побудована на асинхронному обміні даними з Firestore, що 8 забезпечує стабільну роботу додатку без необхідності постійного перезавантаження або очікування оновлення інтерфейсу. Реалізація підтримки повторюваних подій значно підвищує функціональність додатку, дозволяючи створювати не окремі події, а повноцінні шаблони розкладу.

3.3 Форма додавання та редагування події

Реалізація створення та редагування подій у додатку MyPlannerApp базується на чітко структурованій логіці, що дозволяє повторно використовувати код, забезпечити валідацію введених даних, зручну інтеграцію з Firebase Firestore та гнучку підтримку складної поведінки — зокрема повторюваності та кольорового маркування подій, можемо це розглянути на рисунку 3.5.

1 Рисунок 3.5 – Структурна схема логіки формування події через компонент EventForm.js

1 Для того щоб створити нову подію користувач натискає кнопку + в нижньому правому куті екрану, яка відриває модальне вікно додавання подію, що показано на рисунку 3.6 .

1 Рисунок 3.6 – Головний екран з кнопкою додавання нової події

Уся логіка розділена між двома контейнерними модальними компонентами —

`CreateEventModal.js` і `EditEventModal.js`, які відповідають за ініціацію відповідних дій (додавання або редагування), та спільним функціональним компонентом `EventForm.js`, що реалізує власне форму введення і логіку обробки даних.

Процес створення події у додатку реалізовано у вигляді багаторівневого модального вікна `CreateEventModal.js`, що містить чітко структуровану форму з інтуїтивно зрозумілим інтерфейсом, поділену на блоки. Уся логіка обробки введених даних винесена в окремий компонент `EventForm.js`, який також використовується у вікні редагування (`EditEventModal.js`). Така архітектура дозволяє централізовано керувати усією логікою створення, повторення, збереження кольорових категорій, валідації та формування подій у Firebase.

Інтерфейс форми має логічний поділ на кілька частин. На рисунку 3.7 показано блок вибору типу події

1 Рисунок 3.7 – Блок вибору типу події

Вибір типу події — як одна з шести заздалегідь заданих кнопок (Лекція, Лабораторна робота, Практична робота, Семінар, Залік або Інше). У разі вибору «Інше» користувач може ввести довільну назву. Такий вибір дозволяє швидко класифікувати події за призначенням.

Другим блоком є основна інформація показана на рисунку 3.6, яка містить обов’язкові поля: назва події (опціональна), дата, час початку та час завершення.

Рисунок 3.8 – Блок основної та додаткової інформації

Для вибору дати й часу використано зручні “DateTimePicker” рисунок 3.9, які запобігають помилкам формату та забезпечують мобільну адаптивність.

Рисунок 3.9 – DateTimePicker

Для зручності вибору часу та дати, було вирішено додати кнопки «готово», оскільки “DateTimePicker” зникає одразу після першого вибору дати або часу. При додані кнопки пікер не закривається доки не буде натиснуто «готово» що дає користувачу безмежну кількість часу над вирішенням і обранням відповідної дати або часу.

Пікери були обрані як зручним і готовим рішенням проблеми задання проміжку часу для подій. Поле дати відкриває пікер в вигляді календаря з кнопками «далі» та «назад», які дають змогу легко пересуватись від одного місяця до іншого, також можна пересуватись і по роках, для чого також присутня відповідна кнопка (див. рис.3.9).

Для зручного задання часу було обрано барабани (див. рис. 3.9), це забезпечує зручність для користувача, а для універсальності було обрано по-хвилинний проміжок.

Далі слідує блок додаткової інформації (див рис. 3.8) у якому користувач може ввести групу або викладача, аудиторію (місце проведення), назву предмету та примітку. Всі ці поля є не обов’язковими, проте суттєво розширюють змістовність події.

Назвати поле «Група/Викладач» було обрано в зв’язку з тим щоб додатком могли користуватись не тільки студенти але й викладачі, які, наприклад, потребують помічника у вигляді зручного планувальника подій. Таке рішення було прийняте свідомо замість того, щоб реалізовувати дві ролі користувачів (типу: «student», «teacher») або розділяти поля.

Це поле дозволяє зберігати або ім’я викладача (наприклад, «Проф. Іваненко»), або назву групи (наприклад, «КІ-21») — залежно від того, хто користується додатком. Таким чином, воно однаково підходит як студенту, так і викладачу, і не обмежує логіку використання. Кожен користувач просто вводить те, що йому актуально.

Використання єдиного текстового поля зменшує кількість полів у документі події у Firestore, спрощує запити та зберігає гнучкість у використанні. Це особливо важливо у NoSQL-базах, де надмірна структурованість не є обов’язковою й може ускладнювати масштабування.

У додатку немає авторизації за ролями (типу: student/teacher), тому немає сенсу ускладнювати модель даних. Це полегшує реалізацію та робить додаток простішим у використанні. Кожен самостійно вирішує, яку інформацію вводити.

У компоненті EventCard.js чи в календарі це поле можна легко виводити незалежно від того, що введено — ім’я чи назву групи. Таке об'єднання не створює зайвих складнощів при візуалізації.

Використання одного поля `Група/Викладач` замість впровадження ролей користувачів є оптимальним і доцільним рішенням у рамках додатку MyPlannerApp. Воно дозволяє досягти балансу між функціональністю та простотою, зменшує складність логіки, не вимагає впровадження системи користувачів та відповідає потребам обох цільових груп — студентів і викладачів.

Поле Аудиторія/Місце пояснюється тим що додаток створено не тільки для планування розкладу занять, а для планування всіх можливих подій на протязі дня. Тому було вирішено не розділяти поля як окремі «Аудиторія» і «Місце проведення», а створення одного універсального, що значно спрощує структуру, а також не перевантажує форму, оскільки для додатку важливо швидке, просте та універсальне створення події.

У додатку MyPlannerApp реалізована універсальна система повторюваності подій рисунок 3.10.

Рисунок 3.10 – Блок повторюваності. Загальний вигляд

Ця система повторюваності дозволяє значно пришвидшити процес планування типових занять, зустрічей або інших регулярних активностей. Повторюваність є невід'ємною частиною навчального процесу, і саме тому її підтримка була закладена в архітектуру з самого початку.

Кожна подія у Firestore може містити інформацію про те, чи вона повторюється, а якщо так — з якою частотою. Для цього використовується набір полів: repeat, repeatInterval, repeatCount, repeatUntil, а також спеціальний ідентифікатор repeatGroupId. Це дозволяє не лише зберігати основну інформацію про одну подію, але й логічно групувати усі події однієї серії.

Кожна подія у базі Firestore може містити такі поля, пов'язані з повторенням:

{

"repeat": "weekly",

"repeatInterval": 1,

"repeatCount": 8,

"repeatUntil": null,

"repeatGroupId": "rep-abc123"

}

Поле repeat визначає базовий режим повторення. Воно може мати значення never, daily, weekly, monthly, yearly або custom. У режимі custom користувач самостійно визначає інтервал повторення через поле repeatInterval, а також обирає, чи серія має закінчитися через певну кількість повторень (repeatCount), чи в конкретну дату (repeatUntil). Для прикладу, подія може повторюватися щотижня 8 разів, або щомісяця до 20 грудня.

Наприклад на рисунку 3.11 показано блок повторюваності подій, а саме режим "Кожні X днів"

Рисунок 3.11 – Блок повторюваності подій. Режим "Кожні X днів"

За допомогою цього режиму легко можна організувати подію, яка повторюється через 2 тижні, що актуально для розкладу пар коли є чергування тижнів парний/непарний. І в той же час форма не прив'язана до студентського розкладу і можна створювати будь-які

події, в тому числі це може бути робочий графік, або подія для квартальних чи річних звітів.

Ключовим полем є repeatGroupId — це унікальний ідентифікатор, який присвоюється всім подіям, створеним у рамках одного повторюваного циклу. Завдяки цьому додаток отримує можливість виконувати групові дії: видалення всієї серії, масове оновлення тощо. Це реалізується дуже просто: замість того щоб вручну видаляти кожну з 16 подій, достатньо виконати запит до Firestore по repeatGroupId, і система знайде всі відповідні записи.

Функціональні можливості повторюваності охоплюють більшість потреб користувача. Найбільш типові сценарії — це щотижневі або щоденні пари. Але, завдяки гнучкому custom-режиму, можна створити, наприклад, подію, що повторюється кожні 3 дні або 2 тижні. Це особливо актуально при складанні сесійних заходів, консультацій, чергувань викладачів, підготовки до заліків тощо.

З користувацької точки зору, повторюваність реалізована максимально зручно. При створенні події користувач обирає частоту, вказує інтервал та кількість повторів, після чого додаток автоматично генерує відповідну кількість подій з однаковим змістом і різними датами. Це дає змогу уникнути монотонного дублювання подій вручну, суттєво пришвидшуючи введення розкладу.

Таким чином, реалізована структура повторюваності є гнучкою, масштабованою та ефективною. Вона дозволяє користувачам заощаджувати час, створювати складні розклади з мінімальними зусиллями та централізовано керувати серіями подій у календарі. Завдяки уніфікованому підходу з використанням repeatGroupId, додаток підтримує як часткову (одна подія), так і повну (вся серія) обробку подій, що відповідає сучасним принципам зручності та UX.

Однією з найважливіших частин форми є «вибір кольору події» рисунок 3.12.

Рисунок 3.12 – Блок кольору події

Він реалізований за допомогою трьох слайдерів: червоний, зелений, синій (RGB), кожен з яких змінює відповідну компоненту кольору. Нижче відображається результат — реальний колір, який буде застосовано до картки події на головному екрані та в календарі.

Замість використання сторонніх бібліотек для вибору кольору, таких як «react-native-color-picker», у додатку MyPlannerApp було обрано власну реалізацію на основі RGB-структури. Це рішення зумовлене прагненням до максимального контролю, простоти та стабільності архітектури проєкту.

Головною перевагою є повний контроль над логікою збереження кольору. У більшості бібліотек колір передається у форматі HEX (`#AABBCC`) або HSL, тоді як структура Firestore набагато зручніше працює з об’єктами формату

При використанні сторонніх бібліотек значення кольору зазвичай повертаються у форматі HEX (#AABBCC) або HSL, тоді як у Firestore зручніше зберігати структуру виду:

{

"r": 33,

"g": 150,

"b": 243

}

Цей підхід дозволяє безпосередньо управляти кожною кольоровою компонентою, змінювати лише одну складову (наприклад, тільки `g`), формувати динамічні палітри та застосовувати колір у стилях без конвертації.

Ще однією перевагою є усунення залежності від стороннього коду. Сторонні бібліотеки часто включають в себе додаткові залежності, які ускладнюють підтримку проєкту, збільшують розмір застосунку, а іноді й вступають у конфлікт із базовими бібліотеками React Native — наприклад, `react-native-gesture-handler`. Власне рішення забезпечує гарантовану стабільність та передбачуваність.

Також важливою перевагою є простота й мінімалізм інтерфейсу. Готові колірні компоненти часто мають складні інтерфейси з колесами, альфа-каналами або пресетами, що лише перевантажують мобільний інтерфейс. У нашому рішенні достатньо трьох горизонтальних слайдерів для червоного, зеленого й синього каналів, що є максимально зрозумілим для користувача і чудово масштабується на різні екрани.

Власна реалізація кольорових слайдерів також надає інтерфейсу унікального зовнішнього вигляду, який можна легко стилізувати відповідно до загального дизайну програми. Це важливо для збереження стилістичної єдності, тоді як сторонні компоненти часто “випадають” з дизайнерського контексту.

Окрім цього, збереження і повторне використання кольору реалізовано у максимально простий спосіб: при створенні категорії об’єкт з назвою та кольором у форматі `{r, g, b}` зберігається у Firestore. У подальшому ці значення використовуються без додаткової обробки — як у стилях (`backgroundColor: rgb(...)`), так і при візуальному відображенні у списках вибору.

У підсумку, вибір на користь власної RGB-структури був стратегічно виваженим. Він забезпечив гнучкість, стабільність, адаптивність та простоту, що робить систему кольорових категорій у MyPlannerApp зрозумілою як для користувача, так і для розробника.

Одразу під слайдерами реалізовано функціонал створення нової категорії рисунок 3.13.

Рисунок 3.13 – Блок кольорових категорій

Користувач вводить назву, натискає «Зберегти категорію», після чого вона додається до колекції `colorCategories` у Firestore. У разі потреби можна обрати вже існуючу категорію з випадаючого списку. Таким чином, користувач формує особисту палітру для класифікації подій.

На рисунку 3.14 представлена структурна схема керування кольоровими категоріями.

Рисунок 3.14 – Структурна схема керування кольоровими категоріями

У процесі створення або редагування події користувач взаємодіє з компонентом EventForm.js, де передбачено вибір кольору. Тут реалізовано два сценарії. Перший — вибір кольору з існуючих категорій, які завантажуються з колекції colorCategories у Firestore. Другий — створення нової категорії, яка складається з унікальної назви та значення кольору у форматі RGB. Після цього нова категорія автоматично зберігається до Firestore і стає доступною для вибору в подальшому. Незалежно від обраного шляху, результатом є застосування кольору до події, що створюється або редагується.

Окремий функціональний модуль — SettingsScreen.js — відповідає за повне керування списком категорій. Саме тут реалізовано можливість редагування назви або кольору вже наявної категорії, а також її видалення. Усі зміни, здійснені на екрані налаштувань, автоматично синхронізуються з Firestore, що дозволяє зберігати актуальність даних у додатку без потреби в перезавантаженні.

Така реалізація забезпечує зручну та масштабовану систему категоризації подій, яка може бути адаптована під потреби як студентів, так і викладачів. Завдяки інтеграції з Firestore, усі зміни зберігаються у хмарі й можуть бути повторно використані в інших частинах додатку або між різними сеансами користування.

Категорії було введено для того щоб користувач до певного типу подій міг присвоїти певний колір. І наприклад в календарі в тижневому або місячному режимі можна було легко орієнтуватися в подіях не дивлячись на маленький масштаб

В кінці модального вікна розміщено дві кнопки: «Зберегти подію» та «Скасувати», які дозволяють завершити або відмінити створення/редагування. Кнопки адаптовані під

мобільні екрани та мають чітку візуальну ієрархію.

Після заповнення усіх полів відбувається формування об’єкта події. Він має таку структуру:

{

"title": "Фізика",

"type": "Лекція",

"auditory": "Ауд. 101",

"groupOrTeacher": "KI-21",

"note": "Розв’язування задач",

"startTime": "2025-06-01T10:00:00",

"endTime": "2025-06-01T11:30:00",

"repeat": "weekly",

"repeatInterval": 1,

"repeatCount": 8,

"repeatUntil": null,

"repeatGroupId": "group-45678",

"colorCategoryId": "cat-01"

}

Особливості створення повторюваних подій полягають у тому, що при натисканні "Зберегти", якщо обраний режим повтору не "ніколи", у Firestore створюється множина подій з розрахованими `startTime`/`endTime` відповідно до інтервалу і кількості/дати завершення. Усі вони пов’язані загальним `repeatGroupId`, що дозволяє в подальшому реалізувати групове редагування та видалення.

Процес видалення також враховує наявність `repeatGroupId`. У модальному вікні перегляду події (`EventDetailModal.js`) користувачу пропонується варіант:

«Видалити тільки цю подію»;

«Видалити всі пов’язані події».

У другому випадку застосовується запит до Firestore на видалення всіх подій з відповідним `repeatGroupId`.

Виділення форми в окремий компонент (`EventForm.js`) дозволило уникнути дублювання коду, адже і створення, і редагування використовують ту саму структуру. Це значно покращує підтримуваність додатку та дозволяє централізовано змінювати логіку валідації, обробки кольорів, повторюваності та формування об’єкта події.

Відображення подій у вигляді карток та детальний огляд подій

Один із ключових елементів взаємодії користувача з додатком MyPlannerApp — це головний екран, на якому відображаються усі заплановані події. Прототип головного екрану та карточки події показано на рисунку 3.15.

Рисунок 3.15 – Прототип вигляду картки на головному екрані

Саме через головний екран відбувається основна взаємодія користувача з розкладом, і тому до його реалізації було поставлено вимоги інтуїтивної зрозумілості, візуальної чіткості та швидкого доступу до важливої інформації. Взаємодію компонентів показано на рисунку 3.16.

Рисунок 3.16 – Компонентна схема головного екрану

Усі події, що зберігаються у Firestore, автоматично підвантажуються на головному екрані за допомогою механізму підписки onSnapshot. Це дозволяє у реальному часі відображати будь-які зміни: додавання, редагування або видалення подій. Для представлення кожної події використовується окремий компонент EventCard.js, який отримує дані як пропси та рендерить окрему кольорову картку.

Картка події рисунок 3.17 містить повну інформацію про подію: назву предмета, тип (лекція, семінар тощо), групу або викладача, аудиторію, час початку та завершення, а також коротку примітку.

Рисунок 3.17 – Вигляд подій у вигляді карток на головному екрані

Колір фону картки відповідає вибраній категорії події та реалізується шляхом побудови стилю на основі RGB-значень, отриманих із пов’язаної категорії в Firestore. Це дозволяє візуально класифікувати події за типом або іншим критерієм, залежно від того, як користувач визначив свої кольорові категорії.

Щоб уникнути перевантаження інтерфейсу, всі події відображаються у вигляді прокручуваного списку, а сортування та фільтрація подій керуються зі сторінки

налаштувань. Завдяки цьому на головному екрані відображаються лише ті події, які відповідають активному фільтру періоду (день, 3 дні, тиждень або місяць), що значно підвищує зручність користування та дозволяє сфокусуватись лише на актуальних заходах.

На рисунку 3.18 показано модальне вікно EventDetailModal.js детального перегляду події

Рисунок 3.18 – Детальний огляд події

Кожна картка події є інтерактивною. При натисканні на неї відкривається модальне вікно детального перегляду події — компонент EventDetailModal.js. У цьому вікні користувач бачить розширену інформацію про подію, включно з повною назвою, датою, часом, типом, місцем проведення, викладачем або групою, і текстом примітки. Окрім цього, користувач може виконати редагування або видалення події безпосередньо з модального вікна.

Особливістю реалізації є підтримка логіки видалення для повторюваних подій. Якщо подія належить до серії (має repeatGroupId), то користувачу пропонується вибір: видалити лише обрану подію або всю серію подій. Модальне вікно видалення показано на рисунку 3.19.

Рисунок 3.19 – Видалення події

Це реалізовано за допомогою перевірки наявності ідентифікатора повторюваності та подальшого виконання або одиничного deleteDoc, або масового writeBatch, якщо потрібно видалити всі записи з однаковим repeatGroupId.

Важливо зазначити, що інтерфейс модального вікна зберігає стилістику всього додатку, відображає колір події та дозволяє максимально швидко взаємодіяти з даними. Це забезпечує не лише зручність, але й послідовність у користувацькому досвіді (UX), що особливо важливо для мобільного додатку.

У підсумку, головний екран і модальне вікно перегляду події є центральними точками контакту користувача з додатком. Їхня реалізація базується на принципах доступності, ефективності й візуальної чистоти. Така структура дозволяє користувачу швидко ознайомитися з розкладом, внести зміни або скасувати подію за лічені секунди, що робить додаток справді практичним інструментом для щоденного планування.

Екран «Календар» та «Налаштування». Та їх взаємодія

Окрім головного екрана, в якому події відображаються у вигляді карток, додаток MyPlannerApp містить ще один ключовий розділ — екран “Календар”, що дає змогу

переглядати події в хронологічному контексті, схожому на відомі календарні сервіси (наприклад, Google Calendar). Цей екран створено для користувачів, які звикли мислити у часових блоках, і потребують швидкої візуальної оцінки вільного чи зайнятого часу.

Реалізація календаря побудована на основі бібліотеки `react-native-big-calendar`, яка дозволяє створювати гнучкий інтерфейс з підтримкою кількох режимів відображення. Наприклад режим календаря на місяць – рисунок 3.20.

Рисунок 3.20 – Календар в режимі "Місяць"

Сам вибір режиму відображення не змінюється вручну на самому екрані календаря, а задається через екран “Налаштування”, що дозволяє уникнути перевантаження інтерфейсу та зберегти єдину точку управління фільтрацією. Екран налаштувань показано на рисунку 3.21.

Рисунок 3.21 – Екран налаштувань. Вибір режиму календаря

На екрані календаря кожна подія відображається як кольоровий блок, розміщений відповідно до свого `startTime` та `endTime` Рисунок 3.22 відображає вигляд події в рамках календаря.

Рисунок 3.22 – Вигляд події в режимі календаря

Колір події відповідає вибраній користувачем категорії. У режимах "день", "3 дні" та "тиждень" блоки масштабуються по вертикалі залежно від тривалості події, що дозволяє швидко оцінити обсяг навантаження на конкретну годину чи день. У режимі "місяць" події відображаються як позначки з мінімальною інформацією (назва і колір), що зручно для глобального планування.

Кожен календарний блок є інтерактивним. При натисканні на нього відкривається `EventDetailModal.js` — модальне вікно детального перегляду, що вже описано у попередньому пункті. Це дозволяє не лише переглядати дані, а й виконувати редагування чи видалення безпосередньо з календаря. У разі, якщо подія належить до серії з повторюваністю, користувачу також доступна функція групового видалення.

Важливою частиною додатку є екран “Налаштування”, який виконує роль центру управління фільтрацією та сортуванням подій. На цьому екрані реалізовано два незалежні блоки налаштувань: фільтрація для головного екрана і фільтрація для календаря. Кожен блок містить випадаючий список, де користувач обирає період: день, 3 дні, тиждень або місяць. Обране значення зберігається локально й використовується для динамічного підвантаження подій з Firestore відповідно до заданого діапазону .

Розділ сортування та фільтрації подій на екрані налаштувань показано на рисунку 3.23.

Рисунок 3.23 –. Фільтрація та сортування подій на головному екрані

Окрім фільтрації, екран “Налаштування” містить функцію сортування подій на головному екрані — за алфавітом або за часом початку. Це дає змогу ще більше персоналізувати інтерфейс користувача, зробити розклад зрозумілішим, а роботу з ним — ефективнішою.

На рисунку 3.24 зображено керування кольоровими категоріями

Рисунок 3.24 – Екран налаштувань. Налаштування кольорових категорій

Користувач може редагувати назву категорії, змінювати її колір або повністю видалити. Це дозволяє адаптувати систему кольорового маркування під свої потреби без необхідності переходити в інші частини додатку.

У результаті, екрани “Календар” та “Налаштування” є важливою частиною логіки планування у MyPlannerApp. Календар забезпечує гнучке візуальне представлення подій, дає змогу швидко переміщуватись у часі та переглядати розклад на різні періоди. Налаштування — це інструмент точного контролю, що дозволяє користувачам персоналізувати інтерфейс відповідно до своїх звичок. Обидва екрани працюють синхронно, з повною підтримкою реактивного оновлення даних та узгодженої взаємодії з Firestore.

На рисунку 3.25 показано повний цикл обробки вибору фільтра та сортування для головного екрана додатку.

Рисунок 3.25 – Схема взаємодії фільтрації та сортування на головному екрані

Користувач на екрані SettingsScreen.js встановлює бажаний фільтр періоду (наприклад, день, тиждень, місяць) та метод сортування (за алфавітом або за часом початку). Після вибору обидва параметри зберігаються у AsyncStorage, що дозволяє зберігати налаштування навіть після перезапуску додатку.

Після переходу на головний екран (HomeScreen.js) збережені значення зчитуються з AsyncStorage. Фільтр використовується для отримання подій з Firestore лише за обраний період, а сортування — застосовується вже після отримання даних. Це дозволяє відображати на екрані лише актуальні події, впорядковані у зручному для користувача форматі. Кожна подія виводиться через компонент EventCard.js.

Такий підхід дозволяє гнучко керувати обсягом даних, що відображаються, зберігати консистентність інтерфейсу та адаптувати головний екран до індивідуальних звичок користувача.

На рисунку 3.26 зображена схема взаємодії фільтрації на екрані календар.

Рисунок 3.26 – Схема взаємодії фільтрації на екрані Календар

Усі налаштування фільтрації для екрана CalendarScreen.js виконуються через екран SettingsScreen.js. Користувач має 35 можливість обрати один з чотирьох варіантів: день, 3 дні, тиждень або місяць. Обраний варіант зберігається у AsyncStorage, що дозволяє зчитувати його навіть після закриття програми.

При відкритті календаря на екрані CalendarScreen.js застосунок зчитує значення фільтра з AsyncStorage і використовує його для запиту даних з Firestore — тільки тих подій, які лежать у межах обраного часового періоду. Отримані події виводяться у форматі, який підтримує бібліотека react-native-big-calendar: кожна подія відображається відповідно до часу початку і завершення, а також масштабується залежно від тривалості.Ця схема демонструє послідовний потік даних від користувацького вибору до візуалізації подій у календарі. Такий підхід дозволяє реалізувати контрольоване фільтрування, зберігаючи зручний і стабільний користувацький досвід.

Посилання

Це джерела виділених збігів у вашому документі. Кожен збіг позначено темно-зеленим числом, яке відповідає вказаному тут джерелу. Джерела впорядковані за схожістю — чим вищий бал, тим сильніше збіг.

#	Джерело	%
1	essuir.sumdu.edu.ua	4.1%
2	github.com	1.8%
3	ir.nmu.org.ua	0.8%
4	library.sspu.edu.ua	0.7%
5	er.nau.edu.ua	0.4%
6	er.nau.edu.ua	0.4%
7	elartu.tntu.edu.ua	0.3%
8	ir.nmu.org.ua	0.3%
9	bulletin-chstu.com.ua	0.3%
10	ela.kpi.ua	0.2%
11	openarchive.nure.ua	0.2%
12	reposit.nupp.edu.ua	0.2%
13	pnn.com.ua	0.2%
14	dspace.znu.edu.ua	0.2%
15	iq.vntu.edu.ua	0.2%
16	dspace.puet.edu.ua	0.2%
17	production-ready.dev	0.2%
18	run-it.com.ua	0.2%
19	uk.wikipedia.org / Лямбда-вирази_у_C++	0.2%
20	uk.javascript.info	0.1%
21	nt.ua	0.1%
22	repository.hneu.edu.ua	0.1%
23	gadgets-room.ru	0.1%

#	Джерело	%
24	ue-bulletin.com.ua	0.1%
25	ir.stu.cn.ua	0.1%
26	uk.javascript.info	0.1%
27	dspace.pnpu.edu.ua	0.1%
28	itp.elit.sumdu.edu.ua	0.1%
29	alwelayh.com	0.1%
30	card-file.ontu.edu.ua	0.1%
31	best-free-soft.at.ua	0.1%
32	openarchive.nure.ua	0.1%
33	openarchive.nure.ua	0.1%
34	biblio.umsf.dp.ua	0.1%
35	essuir.sumdu.edu.ua	0.1%
36	ir.library.knu.ua	0.1%
37	eir.zp.edu.ua	0.1%
38	ir.lib.vntu.edu.ua	0.1%
39	77.93.36.128	0.1%

Plag

Дякуємо, що перевірили
свій документ за допомогою
Plag!