Розробка мобільних додатків

Лекція 9 - Побудова автентифікації та авторизації в мобільних додатках

Мета лекції:

- Сформувати чітке розуміння відмінностей між автентифікацією та авторизацією
- Проаналізувати сучасні методи автентифікації користувачів у мобільних додатках
- Освоїти принципи побудови системи реєстрації та входу з використанням Firebase Authentication
- Навчитись реалізовувати навігацію між екранами за допомогою Expo Router
- Реалізувати механізми збереження та відновлення сесії користувача між запусками додатку

Чому авторизація – ключовий компонент сучасних додатків

Персоналізований користувацький досвід

Більшість сучасних мобільних застосунків передбачає автентифікацію користувача. Це дає змогу ідентифікувати особу та надавати персоналізований контент: індивідуальну стрічку, список контактів, історію дій або налаштування. Без входу система не може адаптувати функціональність під конкретного користувача.

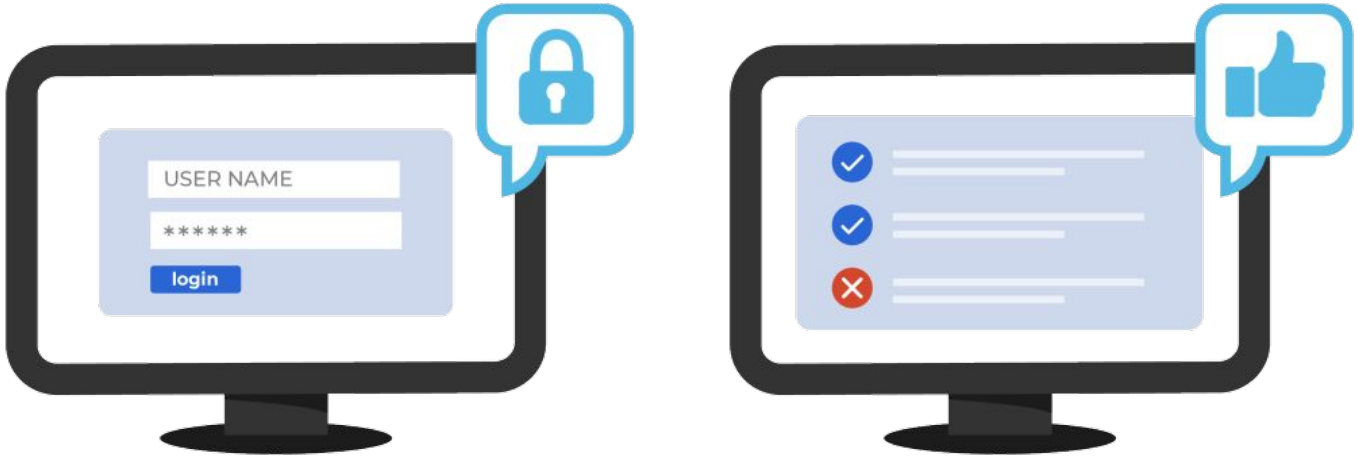
Контроль доступу та захист даних

Авторизація визначає рівень доступу користувача до ресурсів системи та обмежує виконання дій відповідно до його прав. Це забезпечує конфіденційність і цілісність даних: інформація одного користувача (повідомлення, нотатки, фінансові дані) є недоступною для інших. Таким чином, механізм входу виступає базовим рівнем захисту системи.

Зручність використання та управління сесією

Окрім безпеки, важливим аспектом є зручність взаємодії. Сучасні застосунки реалізують механізми збереження сесії, що дозволяє користувачу проходити автентифікацію лише один раз. Подальше автоматичне відновлення сесії забезпечує безперервність роботи та підвищує загальну якість користувацького досвіду (UX).

Автентифікація vs Авторизація



Автентифікація

Автентифікація - це процес перевірки особи користувача, під час якого система підтверджує, що користувач дійсно є тим, за кого себе видає. Це початковий і обов’язковий етап перед наданням доступу до захищених ресурсів інформаційної системи.

Приклад автентифікації:

Введення користувачем електронної пошти або номера телефону разом із паролем

Авторизація

Авторизація - це процес визначення рівня доступу автентифікованого користувача до ресурсів системи. Після встановлення особи користувача система визначає, які дії йому дозволено виконувати та до яких даних він має доступ.

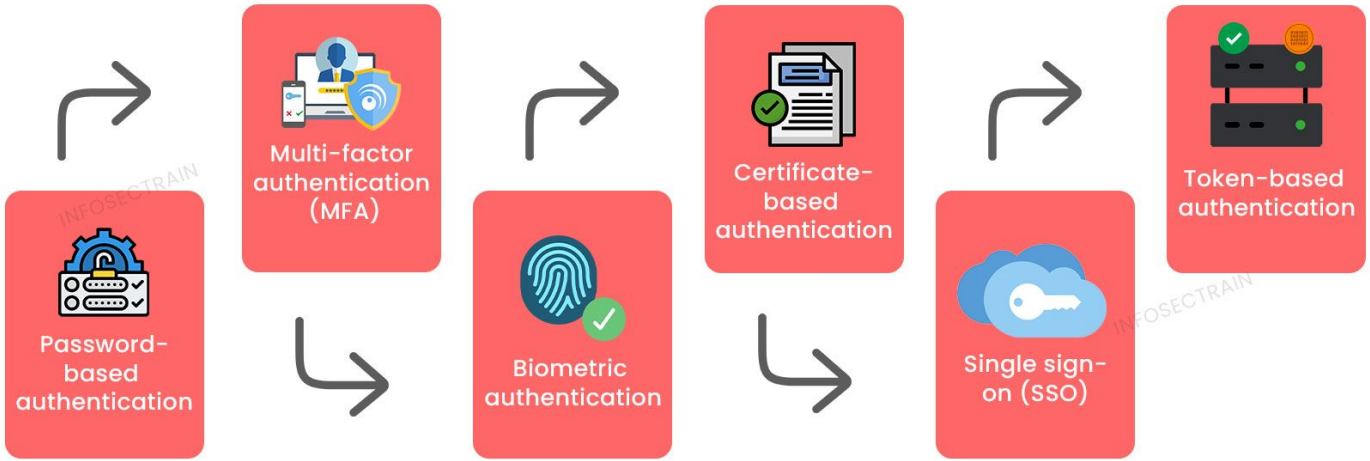
Основні підходи до реалізації авторизації:

- **Ролі користувачів** - визначення набору прав залежно від ролі (наприклад: адміністратор, модератор, користувач)
- **Права доступу** - деталізовані дозволи на виконання конкретних дій (читання, створення, редагування, видалення даних)
- **Власність ресурсів** - обмеження доступу на основі належності даних (користувач має доступ лише до власних ресурсів)

Приклад авторизації:

У застосунку для роботи з нотатками користувач після входу отримує доступ лише до власних записів і може їх редагувати. Навіть за наявності ідентифікатора (ID) чужого запису система не надасть доступ.

Основні типи авторизації в мобільних додатках



Автентифікація за допомогою логіна та пароля

Це один із найпоширеніших і найбільш зрозумілих способів автентифікації. Користувач створює обліковий запис, використовуючи унікальний ідентифікатор (зазвичай електронну пошту або номер телефону) та пароль, які застосовує для подальших входів у систему.

Переваги:

- Простота реалізації та інтеграції
- Інтуїтивно зрозумілий підхід для більшості користувачів
- Повний контроль над обліковими записами без залучення сторонніх сервісів

Недоліки:

- Необхідність запам’ятовування пароля або використання менеджерів паролів
- Ризик застосування слабких або повторно використаних паролів
- Потенційна вразливість за відсутності захищеного з’єднання (HTTPS)
- Додаткові витрати на реалізацію механізмів відновлення пароля та підтвердження email

OAuth 2.0 - авторизація через сторонні сервіси

OAuth 2.0 - це відкритий протокол авторизації, який дозволяє користувачам входити в застосунок, використовуючи наявні облікові записи сторонніх сервісів (Google, Facebook, Apple тощо). Під час входу користувач проходить автентифікацію на стороні провайдера, після чого застосунок отримує токен доступу та базову інформацію профілю.

Переваги:

- Швидка реєстрація та вхід без створення окремого пароля
- Зменшення бар’єру входу для нових користувачів (покращення конверсії)
- Делегування безпеки надійним провайдерам із розвиненою інфраструктурою захисту
- Отримання верифікованих даних користувача (зокрема email)

Недоліки:

- Залежність від доступності сторонніх сервісів
- Складніша логіка реалізації порівняно з класичною автентифікацією
- Необхідність наявності облікового запису у відповідному провайдері
- Обмежений контроль над процесом автентифікації з боку розробника

Біометрична авторизація (FaceID / TouchID)

Біометрична автентифікація дозволяє користувачам отримувати доступ до застосунку або підтверджувати дії за допомогою унікальних фізіологічних характеристик, зокрема відбитка пальця або розпізнавання обличчя. Сучасні мобільні пристрої оснащені відповідними сенсорами (Touch ID, Face ID) та надають розробникам стандартні API для їх інтеграції.

Принцип роботи:
Біометричні дані зберігаються виключно на пристрої користувача у захищеному середовищі і не передаються через мережу. Застосунок взаємодіє із системним API, яке виконує перевірку та повертає лише результат автентифікації (успішно/неуспішно), без доступу до самих біометричних даних.

Переваги:

- Високий рівень зручності — миттєвий доступ без введення облікових даних
- Підвищена безпека завдяки унікальності біометричних характеристик
- Локальне зберігання даних без передачі на сервер
- Ефективність для швидкого повторного входу або підтвердження дій

Недоліки:

- Залежність від наявності відповідного апаратного забезпечення
- Не всі користувачі мають налаштовану біометрію
- Необхідність резервного способу автентифікації (PIN-код, пароль)
- Зазвичай використовується як додатковий фактор (2FA), а не основний метод

На основі **SMS / OTP**

OTP — це метод автентифікації, за якого користувач отримує унікальний одноразовий код на номер телефону або електронну пошту. Такий код має обмежений термін дії (зазвичай 1–10 хвилин) і стає недійсним після використання або завершення цього часу. Метод дозволяє здійснювати вхід без створення та запам’ятовування пароля.

Переваги:

- Відсутність необхідності створювати та запам’ятовувати пароль
- Простий та зрозумілий сценарій входу
- Широке застосування у сервісах, де номер телефону виступає основним ідентифікатором (таксі, доставка, банківські застосунки)
- Обмежений час дії коду підвищує рівень безпеки

Недоліки:

- Вразливість до перехоплення SMS (наприклад, атаки на мережу оператора або SIM-swapping)
- Залежність від стабільності доставки повідомлень
- Необхідність доступу до мобільного зв’язку або електронної пошти
- Недостатній рівень захисту як єдиного методу для критично важливих систем

Необхідні бібліотеки

Сервіси для побулови простої авторизації

Firebase - це набір хмарних сервісів від Google для створення бекенду без необхідності власного сервера. Один із ключових модулів - **Firebase Authentication**, який надає повноцінну систему керування користувачами.

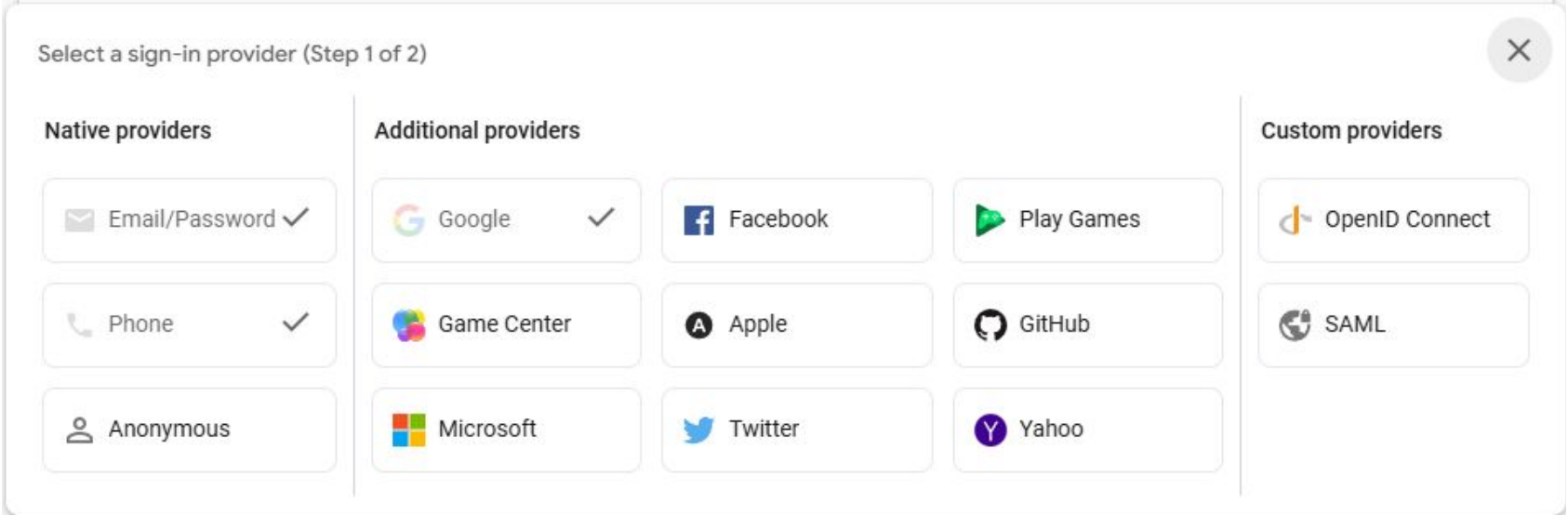
Переваги:

- Підтримка **Google, Facebook, Apple, GitHub** авторизації "з коробки"
- Простий API для **реєстрації та логіну через email**
- Механізми **refresh-токенів, валідації сесій**, захисту
- Реальне **продакшн-рішення**, яке працює в масштабі
- Консоль адміністратора для керування користувачами

Для мобільного застосунку на React Native це означає що повноцінний бекенд авторизації можна підключити за лічені хвилини - і не турбуватись про безпеку токенів, зберігання паролів чи побудову OAuth-флоу вручну.

Firebase провайдери

Firebase Authentication підтримує широкий спектр провайдерів авторизації.

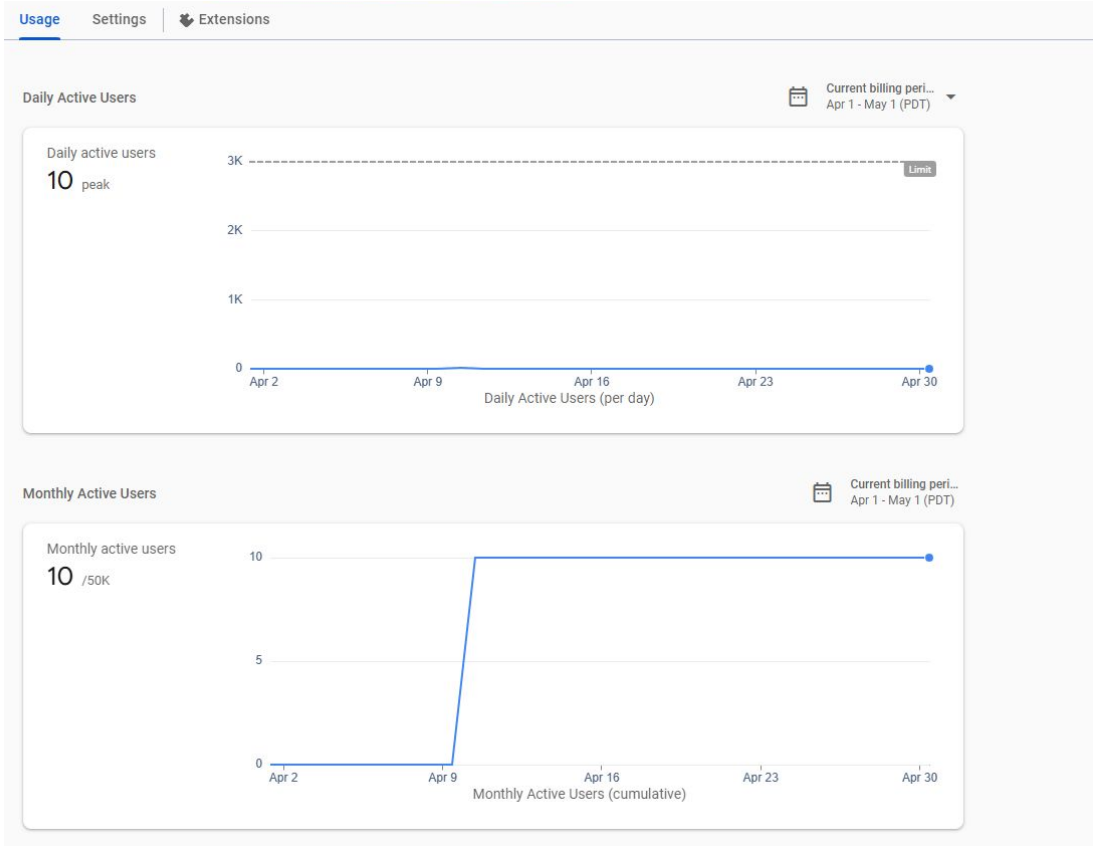


Моніторинг активності користувачів у **Firebase Console**

Firebase Authentication надає вбудований інструмент для відстеження активності користувачів у реальному часі. Це дозволяє розробнику контролювати навантаження на сервіс та своєчасно реагувати на зміни у поведінці аудиторії.

Daily Active Users (DAU) — кількість унікальних користувачів які здійснили вхід протягом одного дня. На безкоштовному плані Spark ліміт складає 3 000 активних користувачів на день.

Monthly Active Users (MAU) — кількість унікальних користувачів за календарний місяць. Безкоштовний план Spark включає до 50 000 активних користувачів на місяць для Email/Password автентифікації.

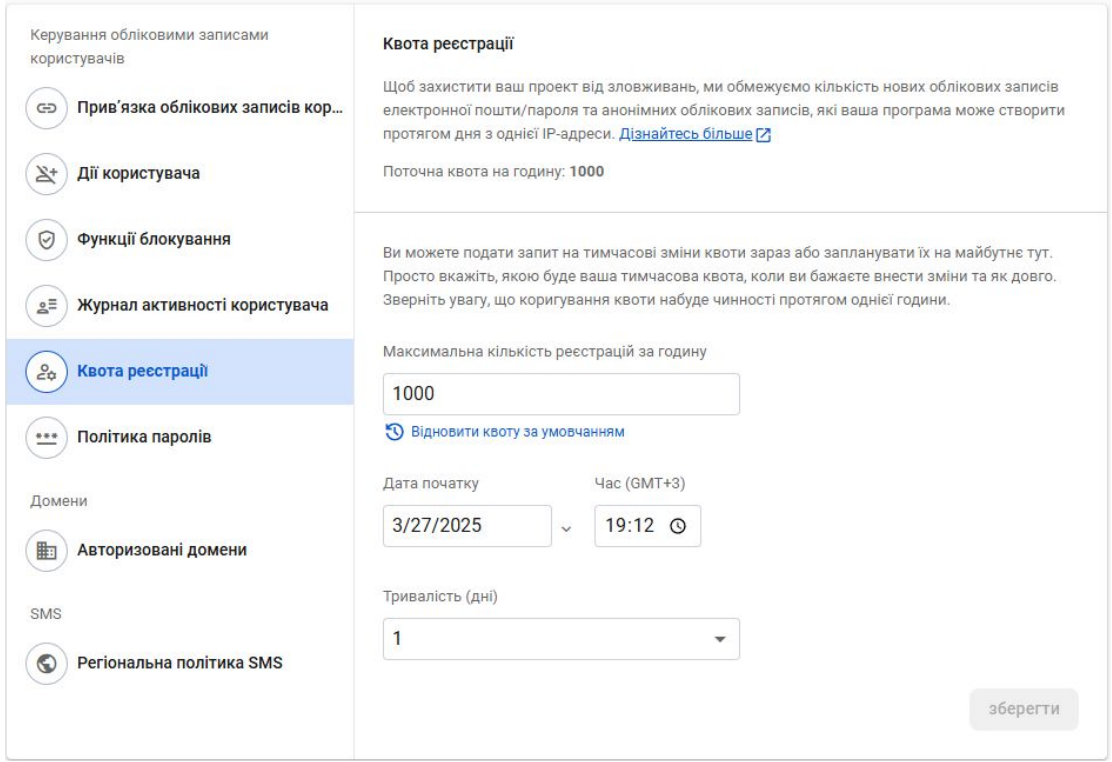


Квота реєстрації та захист від зловживань

Квота реєстрації та захист від зловживань Firebase Authentication має вбудований механізм захисту від автоматичних атак та масової реєстрації фейкових акаунтів.

Квота реєстрації обмежує кількість нових облікових записів які можна створити з однієї IP-адреси протягом певного проміжку часу.

Як це працює: За замовчуванням встановлено ліміт у 1000 реєстрацій на годину з однієї IP-адреси. Це захищає проєкт від ботів та спам-реєстрацій не впливаючи на реальних користувачів.



Expo Router - файлова навігація для **React Native**

Expo Router - це сучасна бібліотека навігації для React Native та Expo яка базується на файловій структурі проєкту. На відміну від React Navigation де маршрути описуються вручну в коді, Expo Router автоматично створює навігацію на основі розташування файлів у папці app/.

Ключові переваги Expo Router:

- Файловий роутинг — кожен файл у папці app/ автоматично стає маршрутом без додаткового налаштування
- Групи маршрутів — папки у круглих дужках (auth) та (app) дозволяють логічно розділяти публічні та захищені екрани
- Вкладена навігація — підтримує Stack, Tabs та Drawer навігацію через спеціальні файли _layout.tsx - Глибокі посилання

Умовна навігація в Expo Router: Замість ручного перемикання між GuestStack та AppStack, Expo Router використовує redirect та захищені групи маршрутів. Залежно від стану авторизації користувач автоматично перенаправляється до потрібної групи екранів.

Підготовка проєкту та структура авторизації

Архітектура застосунку

```
auth-app/
├── app/                          — основна папка маршрутів Expo Router
│   ├── (auth)/                   — група публічних екранів для гостей
│   │   ├── _layout.jsx           — layout для групи auth
│   │   ├── login.jsx             — екран входу
│   │   ├── register.jsx          — екран реєстрації
│   │   └── reset-password.jsx    — екран відновлення пароля
│   ├── (app)/                    — група захищених екранів
│   │   ├── _layout.jsx           — layout для групи app
│   │   └── index.jsx             — головний екран додатку
│   └── _layout.jsx               — кореневий layout з AuthProvider
├── contexts/
│   └── AuthContext.jsx           — глобальний стан авторизації
├── firebase/
│   └── config.js                 — ініціалізація Firebase
└── app.json                      — конфігурація Expo проєкту
```

Project Setup

Створення нового Expo проєкту з Expo Router:

npx create-expo-app@latest

Встановлення Firebase:

npx expo install firebase

Встановлення AsyncStorage для збереження сесії:

npx expo install @react-native-async-storage/async-storage

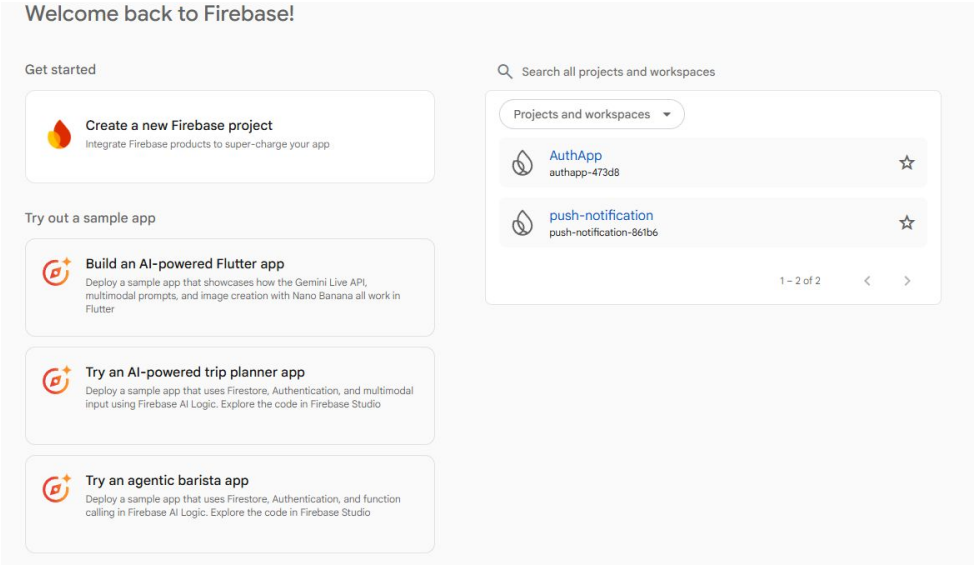
Налаштування Firebase

Firebase Console — це веб-інтерфейс для управління всіма сервісами Firebase.

Саме тут ми створюємо проєкт, підключаємо додатки та налаштовуємо

Authentication перед початком розробки.

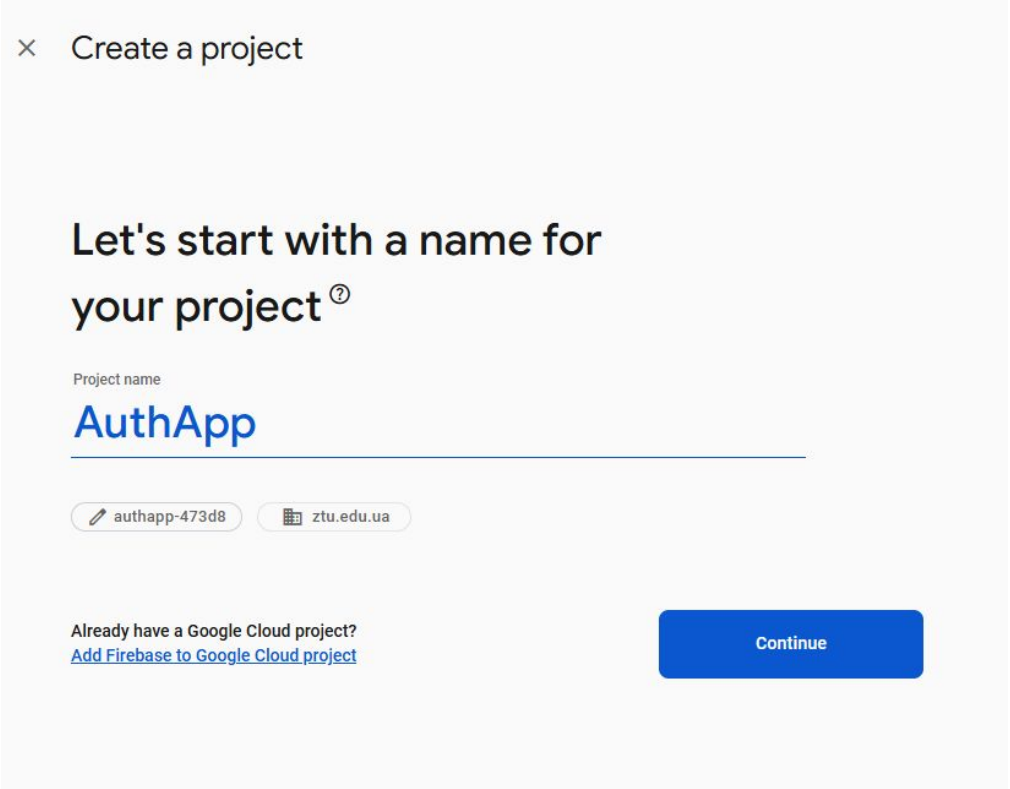
[Firebase Console]



Створення **Firebase** проєкту

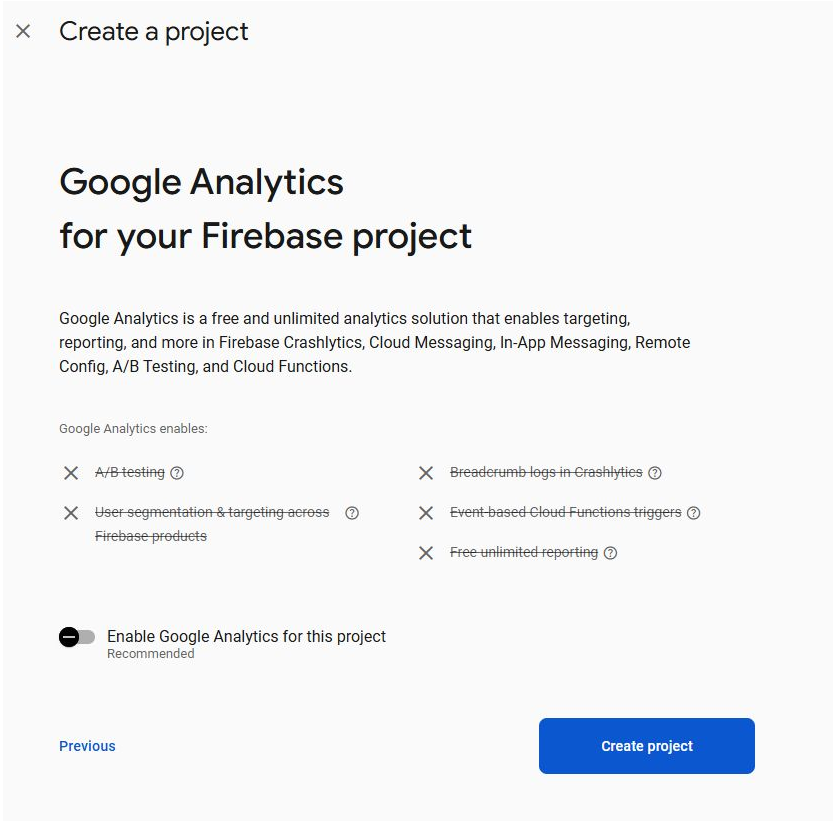
Введіть назву проєкту — наприклад AuthApp.

Firebase автоматично згенерує унікальний ідентифікатор проєкту на основі введеної назви. Ідентифікатор використовується у URL та конфігураційних файлах і не може бути змінений після створення проєкту.



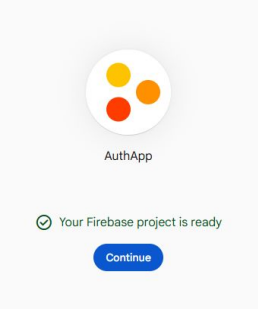
Створення **Firebase** проєкту

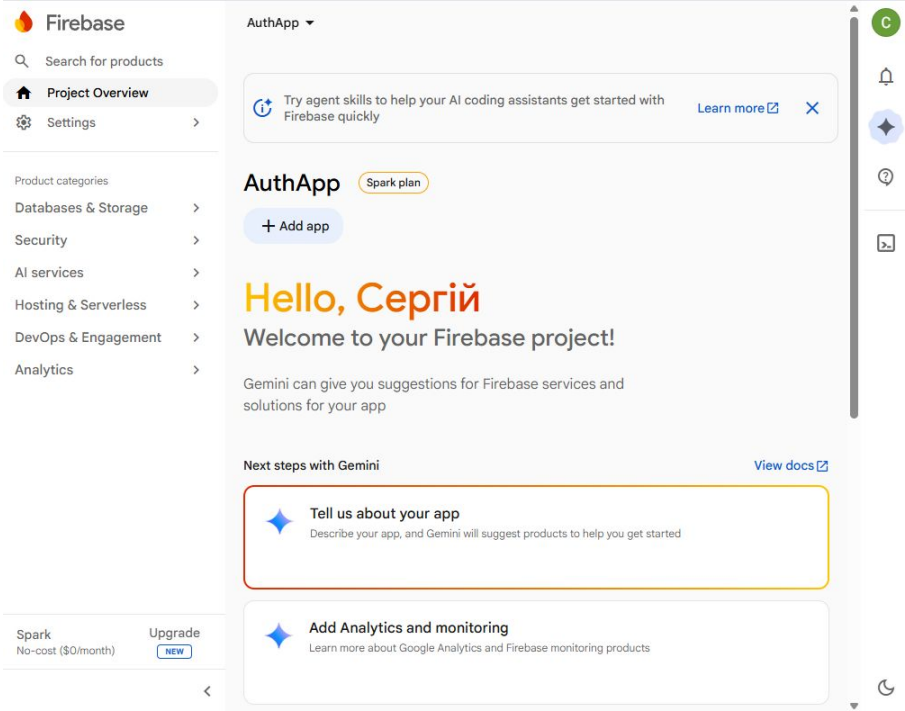
Firebase пропонує підключити Google Analytics для отримання детальної статистики використання додатку. Для навчального проєкту цю опцію можна вимкнути — аналітика не впливає на роботу Authentication та інших сервісів Firebase.



Firebase Console - огляд проєкту

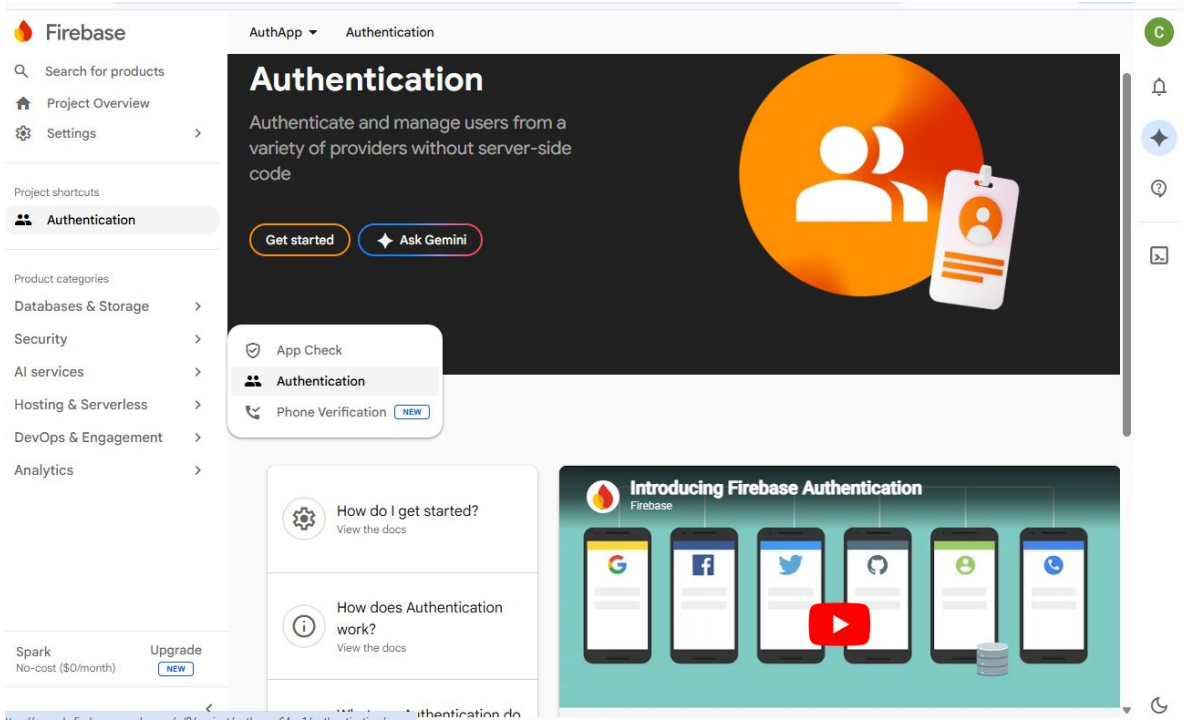
Після створення проєкту ви потрапляєте на головну сторінку Firebase Console. Тут зосереджені всі інструменти необхідні для розробки повноцінного мобільного або веб-додатку.





Firebase Authentication сторінка

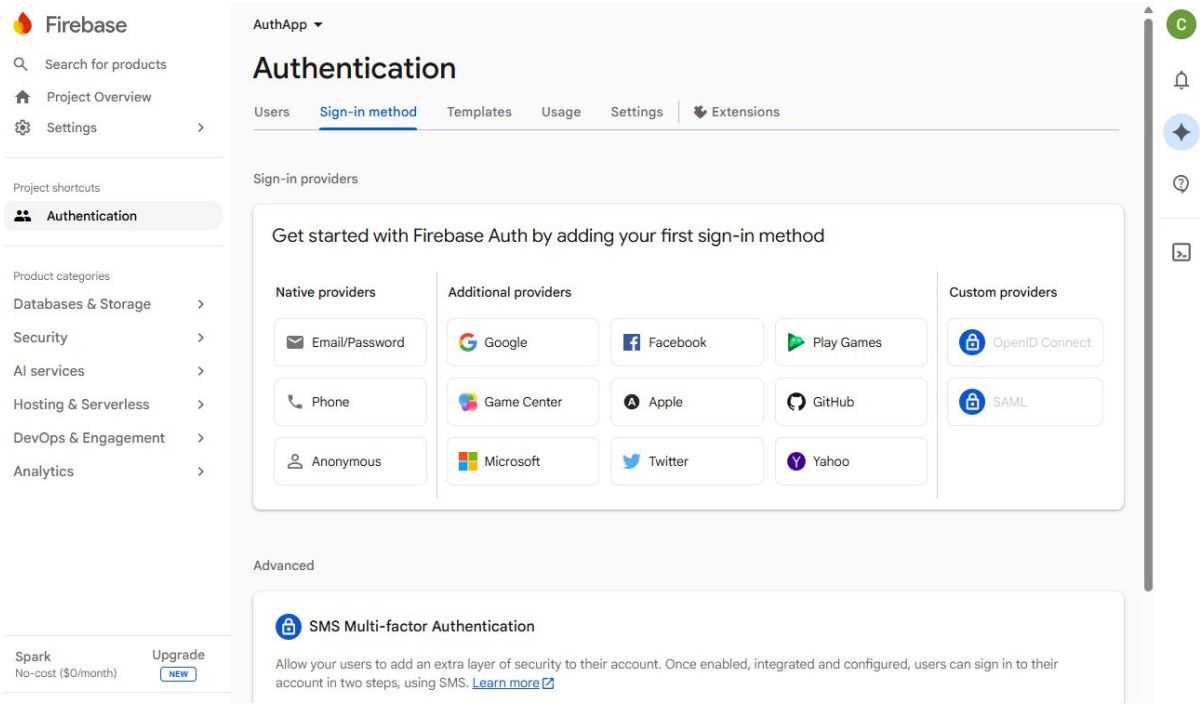
Firebase Authentication - це готовий до використання сервіс управління користувачами який не потребує власного сервера. Він бере на себе всю складну логіку автентифікації і надає розробнику простий та зрозумілий API.



Sign-in providers

Після активації Firebase Authentication перейдіть до вкладки **Sign-in method** — тут відображається список всіх доступних провайдерів авторизації та їх поточний статус.

За замовчуванням всі провайдери вимкнені. Для активації потрібного методу достатньо натиснути на нього та увімкнути відповідний перемикач.

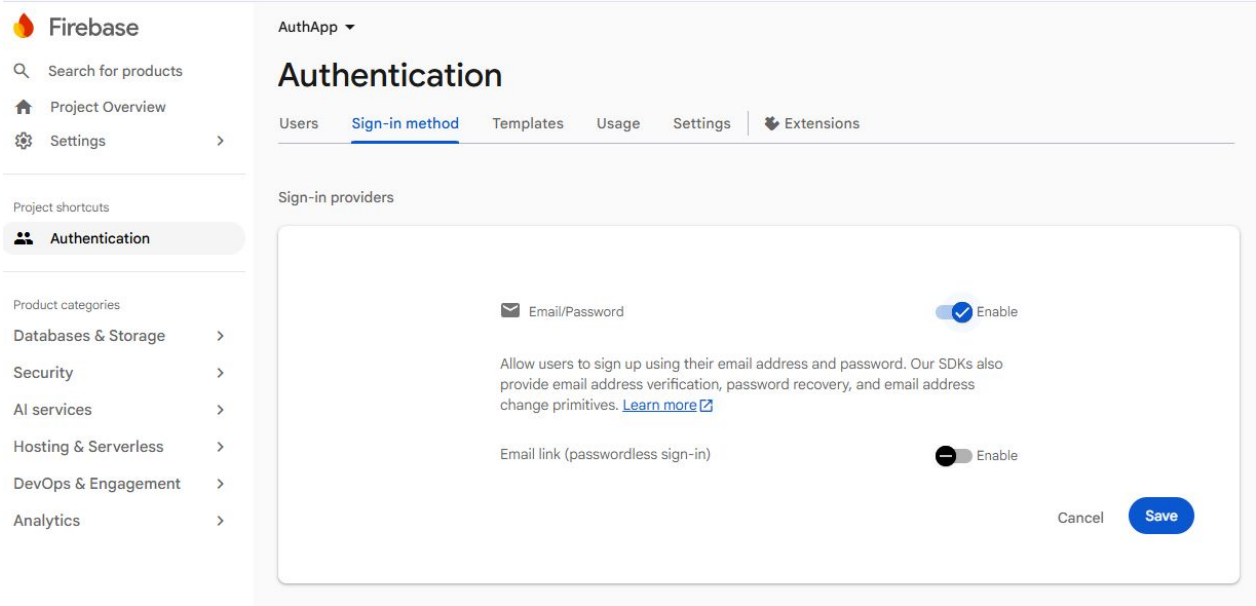


Sign-in providers

Для нашого проєкту активуємо Email/Password - метод авторизації:

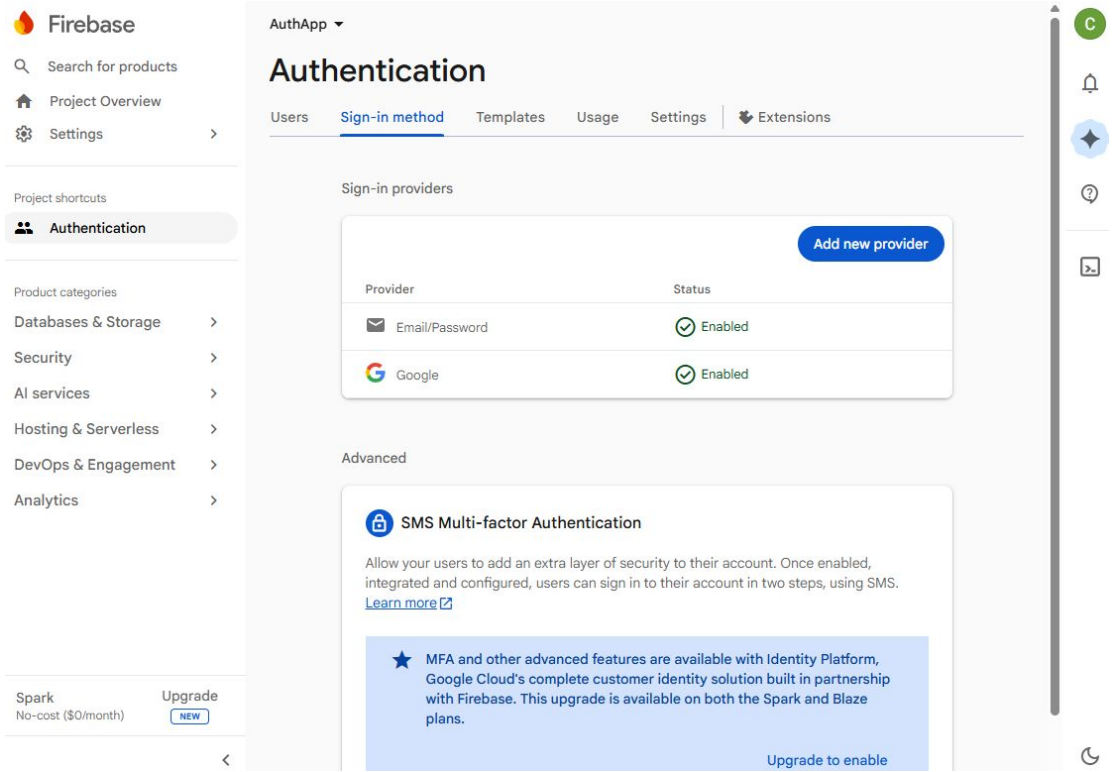
- Дозволяє користувачам реєструватись через електронну пошту та пароль

- Підтримує верифікацію email та відновлення пароля



Email/Password увімкнено

Після збереження налаштувань у списку провайдерів з'являється запис Email/Password зі статусом Enabled. Це означає що сервіс готовий до роботи і ви можете починати реалізацію авторизації у своєму додатку.



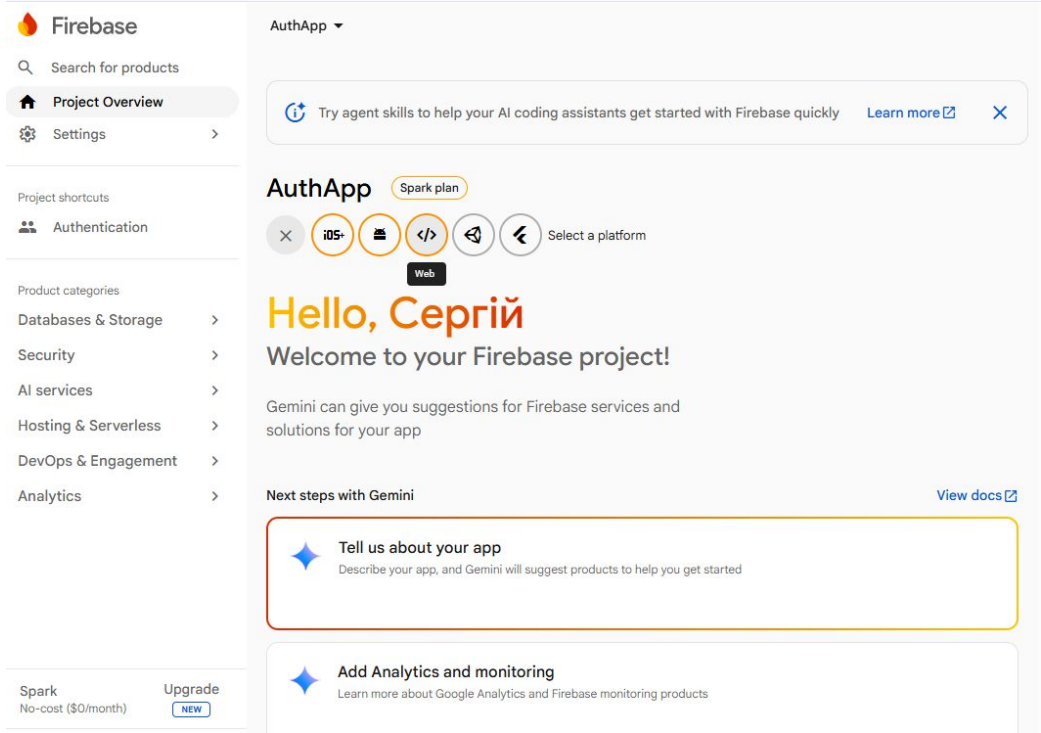
Реєстрація **Web** додатку у **Firebase**

Для підключення Firebase SDK до React Native проєкту необхідно зареєструвати додаток у Firebase Console. Незважаючи на те що ми розробляємо мобільний додаток - для Expo та React Native використовується Web конфігурація Firebase оскільки JavaScript SDK працює однаково на всіх платформах.

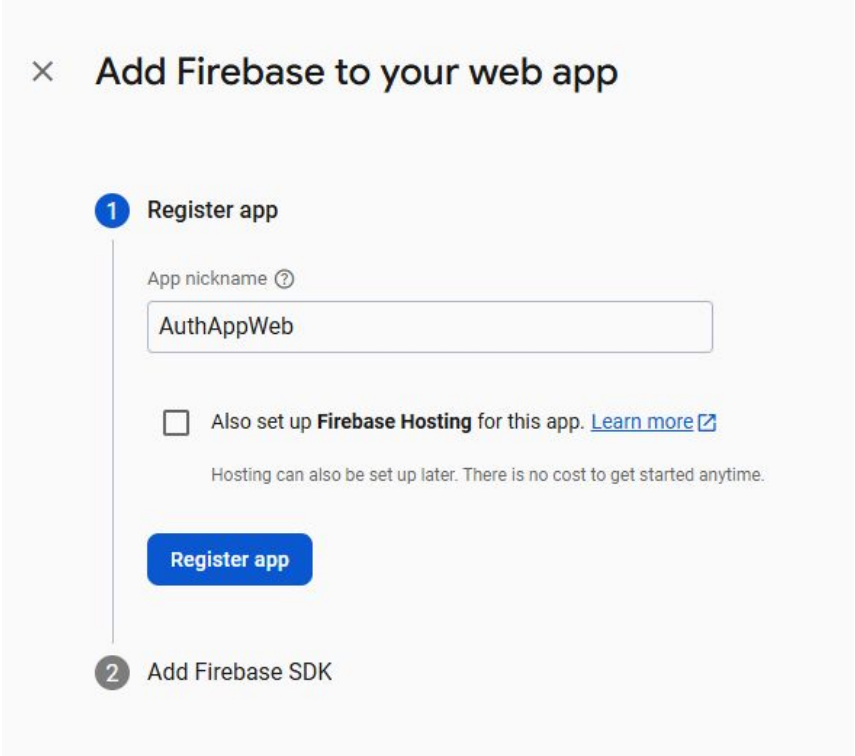
Чому Web конфігурація для React Native: Firebase надає окремі нативні SDK для Android та iOS які використовуються у чисто нативних додатках. Проте React Native з Expo використовує JavaScript оточення — тому Web SDK є правильним та рекомендованим вибором для нашого проєкту.

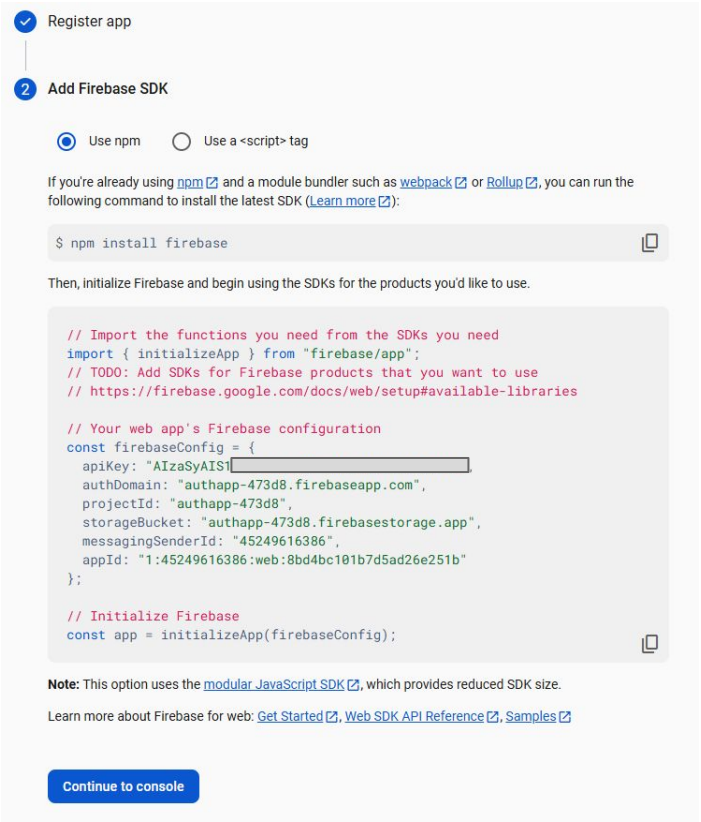
Реєстрація **Web** додатку

У головному меню Firebase Console натисніть іконку Web </> - щоб розпочати процес реєстрації веб-додатку. Введіть назву додатку - наприклад AuthAppWeb — це лише внутрішня назва для ідентифікації у консолі і не впливає на роботу додатку.



Реєстрація **Web** додатку та **SDK**





Реєстрація **Web** додатку та **SDK**

Після реєстрації додатку Firebase надає готовий код ініціалізації з усіма необхідними ключами та ідентифікаторами. Це найважливіший крок — без цих даних підключення SDK до проєкту неможливе.

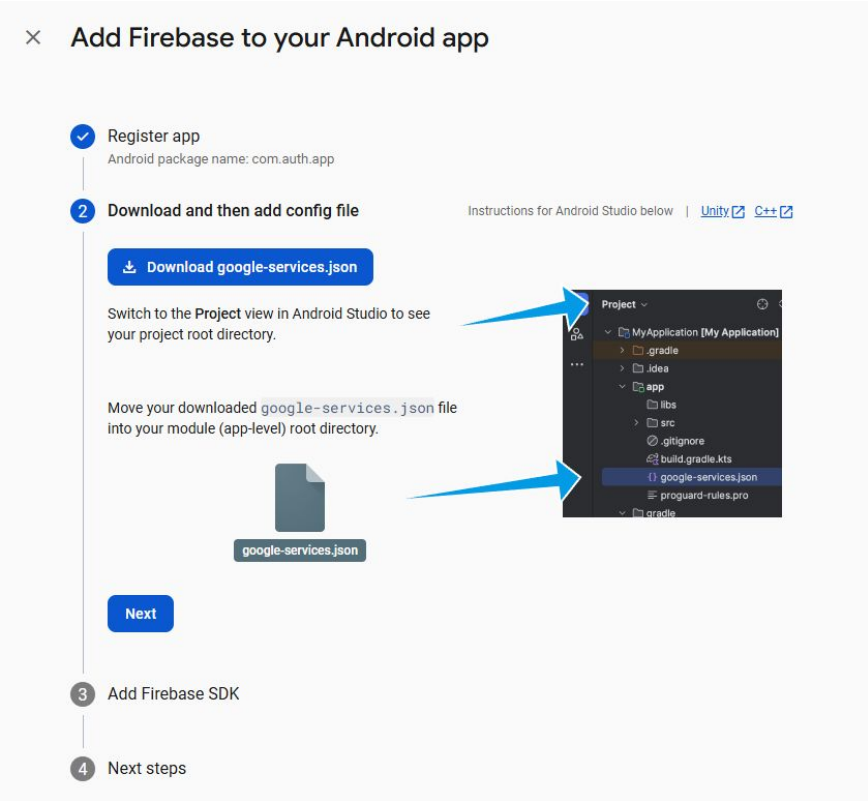
Важливі правила безпеки:

- Ніколи не публікуйте firebaseConfig у відкритих репозиторіях
- Зберігайте конфігурацію у файлі .env та додайте його до .gitignore
- Обмежте використання API ключа через Firebase Console та Google Cloud Console - дозвольте лише необхідні API сервіси
- Налаштуйте правила безпеки Firestore та Storage для додаткового захисту даних на рівні бази даних

Після отримання конфігурації натисніть "Continue to console" - дані завжди доступні у налаштуваннях проєкту Firebase Console у розділі Project Settings → General → Your apps.

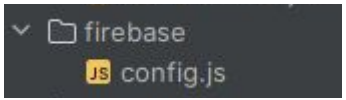
Завантаження **google-services.json**

Після реєстрації Android додатку та додавання SHA-1 відбитку Firebase генерує файл google-services.json - конфігураційний файл який містить всі необхідні ідентифікатори для нативної інтеграції Firebase сервісів з Android платформою.



Ініціалізуємо **Firebase** у проєкті

Цей код виконує ініціалізацію Firebase у веб-проєкті, використовуючи модульний підхід. Він підключає основні функції Firebase (`initializeApp`) та сервіс авторизації (`getAuth`), задає конфігурацію проєкту через об'єкт `firebaseConfig`, а далі ініціалізує застосунок за допомогою `initializeApp(firebaseConfig)`. Після цього створюється об'єкт `authentication`, який дозволяє використовувати функціонал Firebase Auth (реєстрацію, вхід користувачів тощо). Наприкінці цей об'єкт експортується, щоб його можна було легко використовувати в інших частинах застосунку.




```
import { initializeApp } from "firebase/app";
import { initializeAuth, getAuth } from "firebase/auth";

const firebaseConfig = {
    apiKey: process.env.EXPO_PUBLIC_FIREBASE_API_KEY,
    authDomain: process.env.EXPO_PUBLIC_FIREBASE_AUTH_DOMAIN,
    projectId: process.env.EXPO_PUBLIC_FIREBASE_PROJECT_ID,
    storageBucket: process.env.EXPO_PUBLIC_FIREBASE_STORAGE_BUCKET,
    messagingSenderId: process.env.EXPO_PUBLIC_FIREBASE_MESSAGING_SENDER_ID,
    appId: process.env.EXPO_PUBLIC_FIREBASE_APP_ID,
};

const app = initializeApp(firebaseConfig);

initializeAuth(app);

Show usages  Sergey
const authentication = getAuth(app);

export { authentication };
```

Попередження: Збереження стану авторизації

Проблема: За замовчуванням Firebase SDK у React Native використовує збереження в пам'яті. Це означає, що токен авторизації живе лише доти, доки додаток відкритий. Як тільки користувач закриє додаток або оновить його в Expo Go - сесія зникне, і він вийде з акаунта.

Технічна причина: У звичайному браузері Firebase автоматично використовує `localStorage`. Проте в мобільних середовищах (React Native/Expo) немає стандартного `localStorage`, тому SDK не знає, куди надійно записати дані про вхід.

Рішення: Необхідно явно вказати Firebase використовувати **AsyncStorage** - локальне сховище пристрою, яке працює як аналог бази даних «ключ-значення».

```
You are initializing Firebase Auth for React Native without providing
AsyncStorage. Auth state will default to memory persistence and will not
persist between sessions. In order to persist auth state, install the package
"@react-native-async-storage/async-storage" and provide it to
initializeAuth:

import { initializeAuth, getReactNativePersistence } from 'firebase/auth';
import ReactNativeAsyncStorage from '@react-native-async-storage/async-storage';
const auth = initializeAuth(app, {
  persistence: getReactNativePersistence(ReactNativeAsyncStorage)
});
```

Виправлення

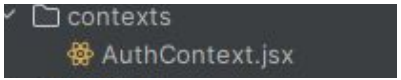
firebase\config.js

```
1  import { initializeApp } from "firebase/app";
2 -import { initializeAuth, getAuth } from "firebase/auth";
2 +import { initializeAuth, getAuth, getReactNativePersistence } from "firebase/auth";
3 +import ReactNativeAsyncStorage from "@react-native-async-storage/async-storage";
4
5  const firebaseConfig = {
6      apiKey: process.env.EXPO_PUBLIC_FIREBASE_API_KEY,
...
13
14  const app = initializeApp(firebaseConfig);
15
15 -initializeAuth(app);
16 +initializeAuth(app, {
17 +    persistence: getReactNativePersistence(ReactNativeAsyncStorage),
18 +});
19
20  const authentication = getAuth(app);
21
```

Створення **AuthContext**

```
import React, { useState, useEffect } from "react";
import { ActivityIndicator, View } from "react-native";
import { authentication } from "../firebase/config";
import { onAuthStateChanged } from "firebase/auth";
```

Цей код створює **контекст авторизації (AuthContext)** у React-застосунку, щоб централізовано зберігати й передавати інформацію про авторизованого користувача. За допомогою `React.createContext()` створюється контекст, далі компонент `AuthProvider` огортає інші компоненти і надає їм доступ до стану `firebaseUser` (користувач, який увійшов у систему) та функції `setFirebaseUser` для його зміни. Хук `useAuth` спрощує доступ до цього контексту з будь-якого компонента, дозволяючи легко отримувати або змінювати дані про авторизованого користувача. Такий підхід забезпечує зручне управління станом авторизації у всьому застосунку.




```
Show usages  Sergey
export const useAuth = () => React.useContext(AuthContext);
```

```
export const AuthProvider = ({ children }) => {
    const [firebaseUser, setFirebaseUser] = useState(null);
    const [isLoading, setIsLoading] = useState(true);

    useEffect(() => {
        const unsubscribe = onAuthStateChanged(authentication, (user) => {
            setFirebaseUser(user);
            setIsLoading(false);
        });
        return unsubscribe;
    }, []);

    if (isLoading) {
        return (
            <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
                <ActivityIndicator size="large" color="#3d5af1" />
            </View>
        );
    }

    return (
        <AuthContext.Provider value={{ firebaseUser, setFirebaseUser }}>
            {children}
        </AuthContext.Provider>
    );
};
```

Кореневий **layout** з **Expo Router**

У Expo Router головним файлом навігації є кореневий _layout.jsx розташований безпосередньо у папці app/. Саме тут відбувається підключення AuthProvider та реалізація умовної навігації — перенаправлення користувача до потрібної групи екранів залежно від стану авторизації.

Як працює умовна навігація:

- useSegments - повертає масив поточних сегментів маршруту
- inAuthGroup - перевіряє чи знаходиться користувач у групі (auth)
- Якщо користувач не авторизований і не у групі (auth)
- перенаправляємо на екран входу
- Якщо користувач авторизований і знаходиться у групі (auth) - перенаправляємо до захищеної групи (app)
- Slot - рендерить поточний активний екран маршруту

```jsx
import { AuthProvider, useAuth } from "../contexts/AuthContext";

Show usages  Sergey *
const InitialLayout = () => {
  const { firebaseUser } = useAuth();
  const router = useRouter();
  const segments = useSegments();

  useEffect(() => {
    const inAuthGroup = segments[0] === "(auth)";

    if (!firebaseUser && !inAuthGroup) {
      router.replace("/(auth)/login");
    } else if (firebaseUser && firebaseUser.emailVerified && inAuthGroup) {
      router.replace("/(app)");
    }
  }, [firebaseUser]);

  return <Slot />;
};

no usages  Sergey
export default function RootLayout() {
  return (
      <AuthProvider>
        <InitialLayout />
      </AuthProvider>
  );
}
```

Layout групи **(auth)**

Файл app/(auth)/_layout.jsx визначає навігаційну структуру для групи публічних екранів - тих які доступні користувачу до входу в систему. Це екрани входу, реєстрації та відновлення пароля.

Що відбувається у цьому коді:

- Stack - навігація типу стек де екрани накладаються один на одний з можливістю повернення назад
- screenOptions={{ headerShown: false }} - вимикає стандартний заголовок для всіх екранів групи
- Stack.Screen - реєструє екран з відповідною назвою яка збігається з назвою файлу у папці (auth)

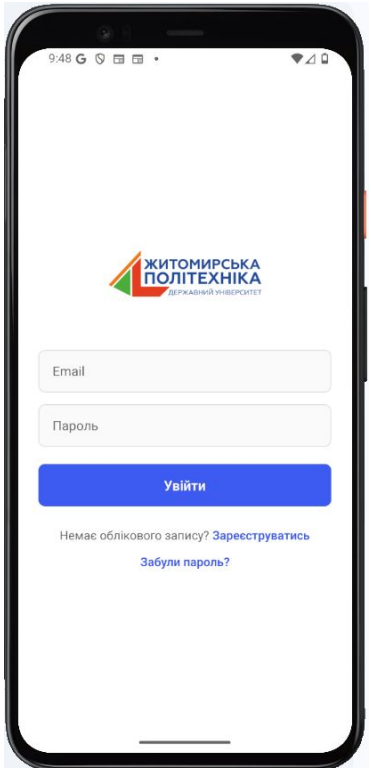
```jsx
import { Stack } from "expo-router";

no usages  new *
export default function AuthLayout() {
    return (
        <Stack screenOptions={{ headerShown: false }}>
            <Stack.Screen name="login" />
            <Stack.Screen name="register" />
            <Stack.Screen name="reset-password" />
        </Stack>
    );
}
```

login.jsx

```jsx
<Image
    source={{ uri: "https://ztu.edu.ua/img/logo/university-colored.png" }}
    style={styles.logo}
    resizeMode="contain"
/>
<TextInput
    style={styles.input}
    placeholder="Email"
    placeholderTextColor="#666"
    keyboardType="email-address"
    autoCapitalize="none"
    value={email}
    onChangeText={setEmail}
    returnKeyType="next"
    onSubmitEditing={() => passwordRef.current.focus()}
/>
<TextInput
    ref={passwordRef}
    style={styles.input}
    placeholder="Пароль"
    placeholderTextColor="#666"
    secureTextEntry
    value={password}
    onChangeText={setPassword}
    returnKeyType="done"
    onSubmitEditing={handleSignIn}
/>
```

```jsx
<TouchableOpacity
    style={[styles.button, isLoading && styles.buttonDisabled]}
    onPress={handleSignIn}
    disabled={isLoading}
>
    <Text style={styles.buttonText}>Увійти</Text>
    {isLoading && <ActivityIndicator color="#fff" style={styles.loader} />}
</TouchableOpacity>
<View style={styles.footer}>
    <Text style={styles.footerText}>Немає облікового запису? </Text>
    <TouchableOpacity onPress={() => router.replace("/(auth)/register")}>
        <Text style={styles.linkText}>Зареєструватись</Text>
    </TouchableOpacity>
</View>
<TouchableOpacity onPress={() => router.push("/(auth)/reset-password")}>
    <Text style={styles.forgotText}>Забули пароль?</Text>
</TouchableOpacity>
```



signInWithEmailAndPassword

Цей метод забезпечує перевірку облікових даних користувача та створення активної сесії в додатку.

Як це працює:

- **Запит:** SDK надсилає введений Email та Пароль на сервери Firebase Authentication через захищене з'єднання.
- **Перевірка:** Firebase порівнює хеш пароля з тим, що зберігається в базі.
- **Результат:** У разі успіху повертається об'єкт `UserCredential`, який містить унікальний `uid` користувача та токен доступу.

```
import { signInWithEmailAndPassword, signOut } from "firebase/auth";
import { authentication } from "../../firebase/config";
import { useAuth } from "../../contexts/AuthContext";
import { useRouter } from "expo-router";
```

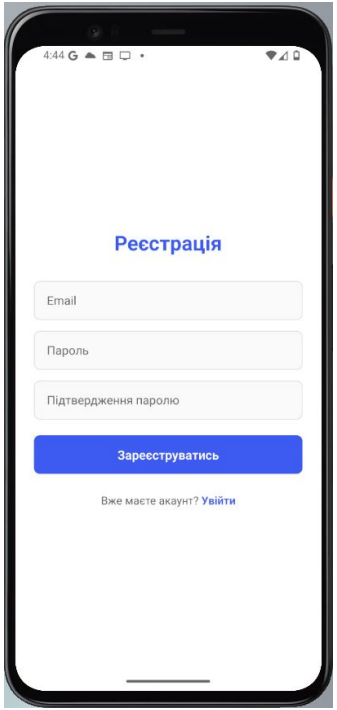
```
const [email, setEmail] = useState("");
const [password, setPassword] = useState("");
const [error, setError] = useState(null);
const [isLoading, setIsLoading] = useState(false);
const { setFirebaseUser } = useAuth();
const router = useRouter();
const passwordRef = useRef();

Show usages  Sergey *
const handleSignIn = async () => {
    if (!email || !password) {
        setError("Заповніть всі поля");
        return;
    }
    setError(null);
    setIsLoading(true);
    try {
        const res = await signInWithEmailAndPassword(authentication, email, password);
        const user = res.user;
        if (!user.emailVerified) {
            Alert.alert("Email не підтверджено", "Підтвердьте акаунт перед входом.");
            await signOut(authentication);
            setFirebaseUser(null);
            return;
        }
        setFirebaseUser(user);
    } catch {
        setError("Невірний Email або Пароль");
    } finally {
        setIsLoading(false);
    }
};
```

register.jsx

```
<Text style={styles.title}>Реєстрація</Text>
<TextInput
    style={styles.input}
    placeholder="Email"
    placeholderTextColor="#666"
    keyboardType="email-address"
    autoCapitalize="none"
    value={email}
    onChangeText={setEmail}
    returnKeyType="next"
    onSubmitEditing={() => passwordRef.current.focus()}
/>
<TextInput
    ref={passwordRef}
    style={styles.input}
    placeholder="Пароль"
    placeholderTextColor="#666"
    secureTextEntry
    value={password}
    onChangeText={setPassword}
    returnKeyType="next"
    onSubmitEditing={() => confirmRef.current.focus()}
/>
<TextInput
    ref={confirmRef}
    style={styles.input}
    placeholder="Підтвердження паролю"
    placeholderTextColor="#666"
    secureTextEntry
    value={confirmPassword}
    onChangeText={setConfirmPassword}
    returnKeyType="done"
    onSubmitEditing={handleSignUp}
/>
{error !== "" && <Text style={styles.errorText}>{error}</Text>}
```

```
<TouchableOpacity
    style={[styles.button, isLoading && styles.buttonDisabled]}
    onPress={handleSignUp}
    disabled={isLoading}
>
    <Text style={styles.buttonText}>Зареєструватись</Text>
    {isLoading && <ActivityIndicator color="#fff" style={styles.loader} />}
</TouchableOpacity>
<View style={styles.footer}>
    <Text style={styles.footerText}>Вже маєте акаунт? </Text>
    <TouchableOpacity onPress={() => router.replace("/(auth)/login")}>
        <Text style={styles.linkText}>Увійти</Text>
    </TouchableOpacity>
</View>
```



`createUserWithEmailAndPassword`

- **Призначення:** Створює новий запис у Firebase Authentication.
- **Процес:** SDK надсилає Email та Пароль; якщо формат коректний і такого користувача ще немає, створюється унікальний **UID**.
- **Результат:** Користувач автоматично стає авторизованим у додатку відразу після реєстрації.

`sendEmailVerification`

- **Призначення:** Надсилає автоматичний лист на вказану пошту з посиланням для підтвердження.
- **Безпека:** Дозволяє відфільтрувати ботів та переконатися, що користувач ввів свою реальну адресу.
- **Статус:** Після натискання на посилання в листі, властивість `emailVerified` у профілі користувача змінюється з `false` на `true`.

```
import { createUserWithEmailAndPassword, sendEmailVerification, signOut } from "firebase/auth";
import { authentication } from "../../firebase/config";
import { useRouter } from "expo-router";
```

```
const handleSignUp = async () => {
    setError("");
    if (!email || !password || !confirmPassword) {
        setError("Заповніть всі поля");
        return;
    }
    if (password !== confirmPassword) {
        setError("Паролі не співпадають");
        return;
    }
    if (password.length < 6) {
        setError("Пароль повинен містити мінімум 6 символів");
        return;
    }
    setIsLoading(true);
    try {
        const { user } = await createUserWithEmailAndPassword(authentication, email, password);
        await sendEmailVerification(user);
        await signOut(authentication);
        Alert.alert(
            "Підтвердьте пошту",
            "Ми надіслали лист на вашу пошту. Підтвердьте її перш ніж входити в систему.",
            [{ text: "OK", onPress: () => router.replace("/(auth)/login") }]
        );
    } catch (err) {
        if (err.code === "auth/email-already-in-use") {
            setError("Акаунт з таким email вже існує");
        } else if (err.code === "auth/invalid-email") {
            setError("Невірний формат email адреси");
        } else {
            setError("Помилка реєстрації. Спробуйте ще раз.");
        }
    } finally {
        setIsLoading(false);
    }
};
```

Templates

Email

- Email address verification
- Password reset
- Email address change
- Multi-factor enrollment notification
- SMTP settings

SMS

- SMS verification

Template language
English

Email address verification

When a user signs up using an email address and password, you can send them a confirmation email to verify their registered email address. Learn more

Sender name
not provided

From
noreply@authapp-473d8.firebaseapp.com

Reply to
noreply

Subject
Verify your email for %APP_NAME%

Message

Hello %DISPLAY_NAME%,

Follow this link to verify your email address.

https://authapp-473d8.firebaseapp.com/__/auth/action?mode=action&oobCode=code

If you didn't ask to verify this address, you can ignore this email.

Thanks,

Your %APP_NAME% team

Templates

Email

- Email address verification
- Password reset
- Email address change
- Multi-factor enrollment notification
- SMTP settings

SMS

- SMS verification

Email address verification

When a user signs up using an email address and password, you can send them a confirmation email to verify their registered email address. Learn more

Sender name: noreply

From: noreply @authapp-473d8.firebaseapp

Customize domain

Reply to:

Subject: Verify your email for %APP_NAME%

Message

```
<p>Hello %DISPLAY_NAME%,</p>
<p>Follow this link to verify your email address.</p>
<p><a href='%LINK%'>%LINK%</a></p>
<p>If you didn't ask to verify this address, you can ignore this email.</p>
<p>Thanks,</p>
<p>Your %APP_NAME% team</p>
```

Action URL (%LINK% value)

https://authapp-473d8.firebaseapp.com/__/auth/action

Customize action URL

Пошук у пошті

Verify your email for project-45249616386

Вхідні ×

noreply@authapp-473d8.firebaseapp.com
кому мені

Hello,

Follow this link to verify your email address.

https://authapp-473d8.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=QqhlY8ixSt0GdC5kQCce5n-MVD7PqPco4H

If you didn't ask to verify this address, you can ignore this email.

Thanks,

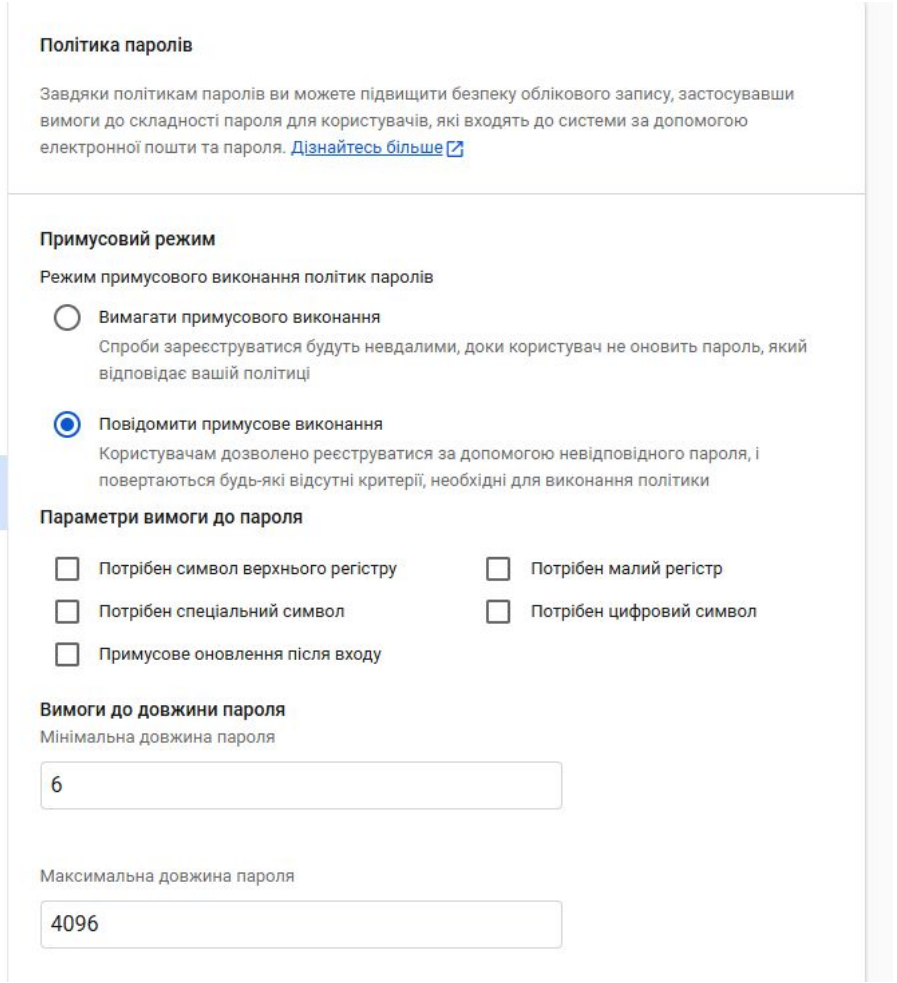
Your project-45249616386 team

Відповісти Переслати

Your email has been verified

You can now sign in with your new account

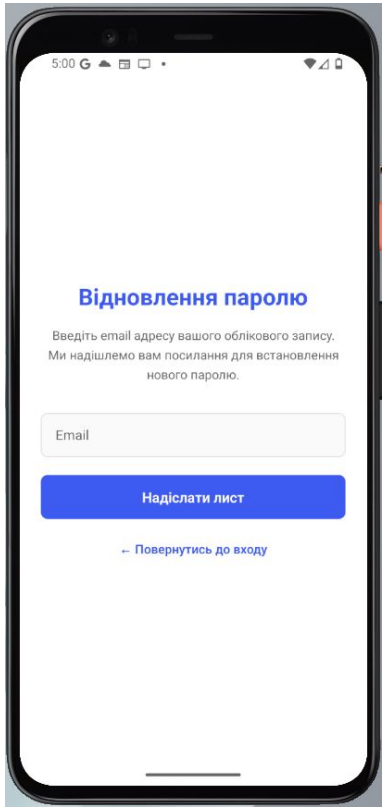
Password policy



reset-password.jsx

У будь-якому застосунку з авторизацією важливо надати користувачу можливість відновити доступ до свого облікового запису. Функція Forgot Password є стандартом безпеки та зручності, яку очікує кожен користувач.

```jsx
<Text style={styles.title}>Відновлення паролю</Text>
<Text style={styles.description}>
    Введіть email адресу вашого облікового запису.
    Ми надішлемо вам посилання для встановлення нового паролю.
</Text>
<TextInput
    style={styles.input}
    placeholder="Email"
    placeholderTextColor="#666"
    keyboardType="email-address"
    autoCapitalize="none"
    value={email}
    onChangeText={setEmail}
    returnKeyType="done"
    onSubmitEditing={handlePasswordReset}
/>
{error !== "" && <Text style={styles.errorText}>{error}</Text>}
<TouchableOpacity
    style={[styles.button, isLoading && styles.buttonDisabled]}
    onPress={handlePasswordReset}
    disabled={isLoading}
>
    <Text style={styles.buttonText}>Надіслати лист</Text>
    {isLoading && <ActivityIndicator color="#fff" style={styles.loader} />}
</TouchableOpacity>
<TouchableOpacity style={styles.backButton} onPress={() => router.back()}>
    <Text style={styles.backText}>← Повернутись до входу</Text>
</TouchableOpacity>
```



sendPasswordResetEmail

Цей метод дозволяє користувачеві безпечно відновити доступ до свого акаунта у разі втрати пароля.

Як це працює:

- **Запит:** Додаток передає лише Email користувача до Firebase Auth.
- **Лист:** Firebase автоматично генерує та надсилає на цю пошту лист із унікальним одноразовим посиланням.
- **Зміна:** Перейшовши за посиланням, користувач потрапляє на захищену сторінку (хостинг Google), де вводить новий пароль.

```
import { sendPasswordResetEmail } from "firebase/auth";
import { authentication } from "../../firebase/config";
import { useRouter } from "expo-router";
```

```
const [email, setEmail] = useState("");
const [error, setError] = useState("");
const [isLoading, setIsLoading] = useState(false);
const router = useRouter();

Show usages  Sergey
const handlePasswordReset = async () => {
    if (!email) {
        setError("Введіть email адресу");
        return;
    }
    setError("");
    setIsLoading(true);
    try {
        await sendPasswordResetEmail(authentication, email);
        Alert.alert(
            "Лист надіслано",
            "Перевірте свою пошту для відновлення паролю.",
            [{ text: "OK", onPress: () => router.back() }]
        );
    } catch (e) {
        if (e.code === "auth/user-not-found") {
            setError("Акаунт з таким email не знайдено");
        } else if (e.code === "auth/invalid-email") {
            setError("Невірний формат email адреси");
        } else {
            setError("Не вдалося надіслати лист. Спробуйте ще раз.");
        }
    } finally {
        setIsLoading(false);
    }
};
```

Templates Usage Settings Extensions

Templates

Email

- Email address verification
- Password reset
- Email address change
- Multi-factor enrollment notification
- SMTP settings

SMS

- SMS verification

Template language
English

Password reset

When a user forgets their password, a password reset email is sent to help them set up a new one.
Learn more

Sender name
not provided

From
noreply@authapp-473d8.firebaseapp.com

Reply to
noreply

Subject
Reset your password for %APP_NAME%

Message

Hello,

Follow this link to reset your %APP_NAME% password for your %EMAIL% account.

https://authapp-473d8.firebaseapp.com/__/auth/action?mode=action&oobCode=code

If you didn't ask to reset your password, you can ignore this email.

Thanks,

Your %APP_NAME% team

Reset your password for project-45249616386 Вхідні ×

noreply@authapp-473d8.firebaseapp.com
кому мені

Перекласти такою мовою: українська

Hello,

Follow this link to reset your project-45249616386 password for your nicebooyyy@gmail.com account.

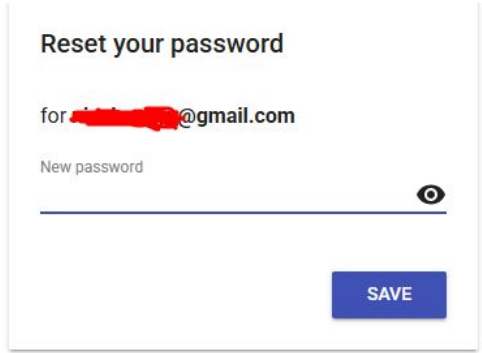
https://authapp-473d8.firebaseapp.com/__/auth/action?mode=resetPassword&oobCode=eoOa4lUm4zLFM-1NI1UXYIRsN3ruWT79jp

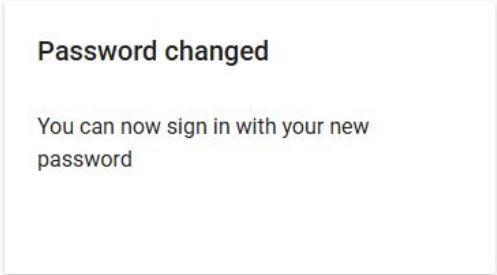
If you didn't ask to reset your password, you can ignore this email.

Thanks,

Your project-45249616386 team

Відповісти Переслати





Layout групи **(app)**

Файл app/(app)/_layout.jsx визначає навігаційну структуру для групи захищених екранів які доступні виключно авторизованим користувачам. Спроба неавторизованого користувача потрапити до цієї групи автоматично перенаправляє його на екран входу.

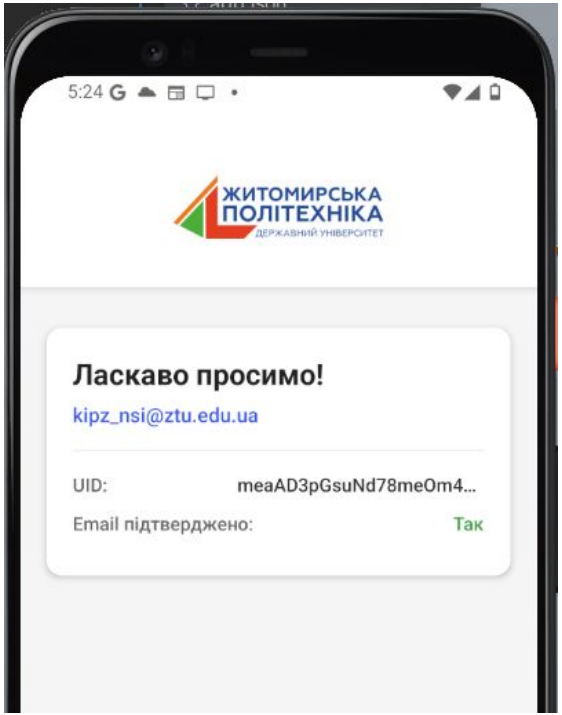
```jsx
import { Stack } from "expo-router";

no usages  new *
export default function AppLayout() {
    return (
        <Stack screenOptions={{ headerShown: false }}>
            <Stack.Screen name="index" />
            <Stack.Screen name="delete-account" />
        </Stack>
    );
}
```

index.jsx

```jsx
<View style={styles.header}>
    <Image
        source={{ uri: "https://ztu.edu.ua/img/logo/university-colored.png" }}
        style={styles.logo}
        resizeMode="contain"
    />
</View>

<View style={styles.content}>
    <View style={styles.card}>
        <Text style={styles.welcomeText}>Ласкаво просимо!</Text>
        <Text style={styles.emailText}>{firebaseUser?.email}</Text>
        <View style={styles.divider} />
        <View style={styles.infoRow}>
            <Text style={styles.infoLabel}>UID:</Text>
            <Text style={styles.infoValue} numberOfLines={1}>
                {firebaseUser?.uid}
            </Text>
        </View>
        <View style={styles.infoRow}>
            <Text style={styles.infoLabel}>Email підтверджено:</Text>
            <Text style={[
                styles.infoValue,
                { color: firebaseUser?.emailVerified ? "#43a047" : "#e53935" }
            ]}>
                {firebaseUser?.emailVerified ? "Так" : "Ні"}
            </Text>
        </View>
    </View>
</View>
```



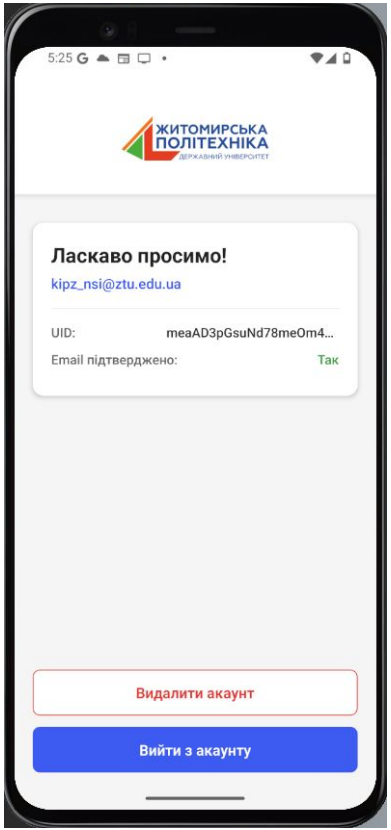
Видалення облікового запису

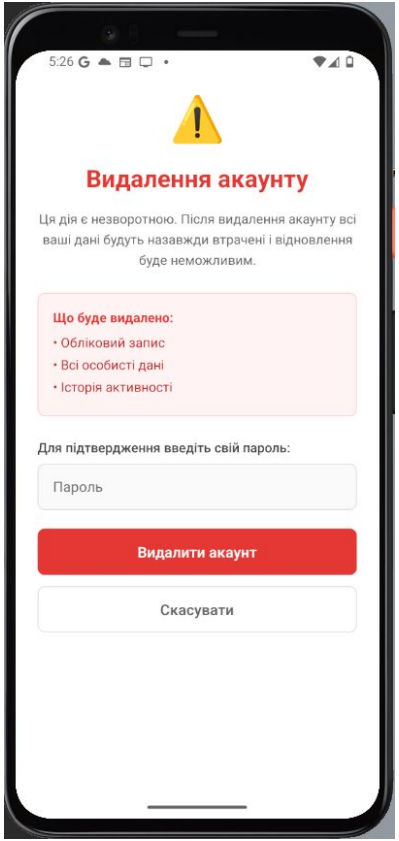
Інколи користувачі хочуть повністю **видалити свій акаунт** з додатку або сервісу. Це може бути зумовлено:

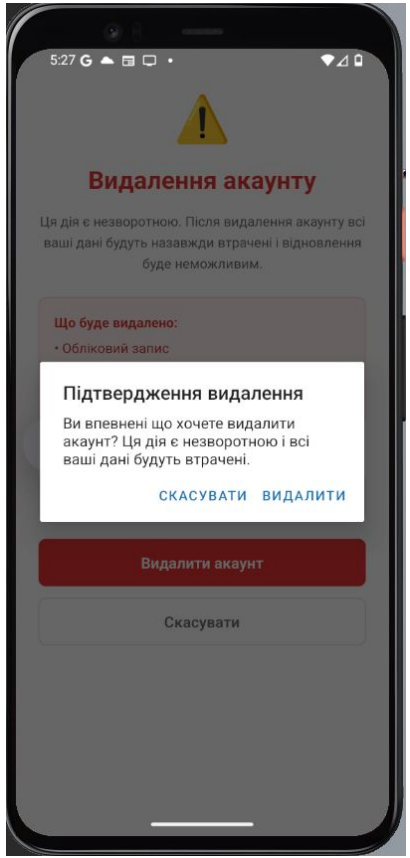
- **Зміною акаунта** - користувач зареєструвався з іншою поштою або через Google і більше не використовує старий профіль.
- **Питаннями конфіденційності** - бажанням видалити персональні дані.
- **Припиненням користування** - додаток більше не актуальний або не відповідає потребам.
- **Очищенням тестових/тимчасових акаунтів** - під час розробки чи тестування.

Видалення облікового запису - важлива частина **права на контроль над своїми даними**.
З точки зору розробника, реалізація цієї функції демонструє **відповідальне ставлення до безпеки та етики роботи з користувачами**.

Потрібно обов’язково забезпечити **повторну авторизацію перед видаленням**, щоб захистити обліковий запис від випадкового або несанкціонованого доступу.


5:25
ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Ласкаво просимо!
kipz_nsi@ztu.edu.ua
UID:
meaAD3pGsuNd78meOm4...
Email підтверджено:
Так
Видалити акаунт
Вийти з акаунту


5:26
Видалення акаунту
Ця дія є незворотною. Після видалення акаунту всі ваші дані будуть назавжди втрачені і відновлення буде неможливим.
Що буде видалено:
• Обліковий запис
• Всі особисті дані
• Історія активності
Для підтвердження введіть свій пароль:
Пароль
Видалити акаунт
Скасувати


5:27
Видалення акаунту
Ця дія є незворотною. Після видалення акаунту всі ваші дані будуть назавжди втрачені і відновлення буде неможливим.
Що буде видалено:
• Обліковий запис
Підтвердження видалення
Ви впевнені що хочете видалити акаунт? Ця дія є незворотною і всі ваші дані будуть втрачені.
СКАСУВАТИ
ВИДАЛИТИ
Видалити акаунт
Скасувати

`deleteUser`

- **Призначення:** Повне видалення облікового запису користувача з бази Firebase Authentication.
- **Наслідок:** Користувач миттєво втрачає доступ до додатка, а його унікальний **UID** стає недійсним.
- **Особливість:** Це не видаляє дані користувача з Firestore автоматично — про очищення бази даних розробник має подбати окремо.

`reauthenticateWithCredential`

- **Призначення:** Повторне підтвердження особи перед виконанням чутливих операцій.
- **Чому це потрібно:** Firebase захищає акаунти від ситуацій, коли хтось інший отримав доступ до розблокованого телефону. Якщо сесія триває довго, спроба видалити акаунт або змінити пароль видасть помилку `auth/requires-recent-login`.
- **Процес:** Користувачеві пропонується ще раз ввести свій поточний пароль. Після цього видається "свіжий" токен, який дозволяє завершити видалення або зміну даних.

```
import { deleteUser, EmailAuthProvider, reauthenticateWithCredential } from "firebase/auth";
import { authentication } from "../../firebase/config";
import { useRouter } from "expo-router";
```

```
Alert.alert(
    "Підтвердження видалення",
    "Ви впевнені що хочете видалити акаунт? " +
    "Ця дія є незворотною і всі ваші дані будуть втрачені.",
    [
        { text: "Скасувати", style: "cancel" },
        {
            text: "Видалити",
            style: "destructive",
            onPress: async () => {
                const user = authentication.currentUser;
                if (!user) {
                    setError("Сесія закінчилась. Перезайдіть в акаунт.");
                    return;
                }
                const credential = EmailAuthProvider.credential(
                    user.email, password
                );
                setIsLoading(true);
                try {
                    await reauthenticateWithCredential(user, credential);
                    await deleteUser(user);
                } catch (error) {
                    if (error.code === "auth/wrong-password" || error.code === "auth/invalid-credential") {
                        setError("Невірний пароль. Спробуйте ще раз.");
                    } else if (error.code === "auth/too-many-requests") {
                        setError("Забагато спроб. Спробуйте пізніше.");
                    } else {
                        setError("Помилка видалення акаунту. Спробуйте ще раз.");
                    }
                } finally {
                    setIsLoading(false);
                }
            },
        },
    ]
);
```