Phone Auth API 完整說明 - Firebase 前後端驗證流程

📌 更新摘要

我已經完成 Phone Auth API 的修正，主要解決 Kevin 提到的幾個問題：

✅ **verify-otp 不再接受任意輸入**（改用真實 Firebase 驗證）
✅ **LOCKED 狀態加入 retry_after 欄位**
✅ **釐清前後端驗證職責**

🔐 Firebase 驗證機制說明

核心概念：前端驗證 OTP，後端驗證憑證

Firebase Phone Auth 使用**雙重驗證機制**，這不是重複驗證，而是分別驗證不同東西：

```
前端驗證：驗證「使用者輸入的 OTP 是否正確」
          ↓
        取得 idToken（Firebase 簽發的憑證）
          ↓
後端驗證：驗證「idToken 是否由 Firebase 真實簽發」（防偽造）
          ↓
        綁定手機號碼到使用者帳號
```

🔄 完整驗證流程

只有前端可以發送 OTP ～

```
❌ 後端無法發送 OTP
   - Firebase Admin SDK（後端）只能驗證，不能發送
   - send-otp API 不會真的發送 SMS

✅ 前端才能發送 OTP
   - 使用 Firebase JS SDK 的 signInWithPhoneNumber
   - Firebase 自動發送 SMS 到使用者手機
```

步驟 1：前端發送 OTP（Firebase JS SDK）

> OTP 只能由前端發送，後面會再說明是否需要 call send_otp API

```
import { getAuth, RecaptchaVerifier, signInWithPhoneNumber } from
'firebase/auth';
```

```
const auth = getAuth();

// 1.1 設定 reCAPTCHA
window.recaptchaVerifier = new RecaptchaVerifier(auth, 'recaptcha-
container', {
  'size': 'invisible'
});

// 1.2 發送 OTP（Firebase 自動發送 SMS）
const phoneNumber = '+886987654321';
const appVerifier = window.recaptchaVerifier;

const confirmationResult = await signInWithPhoneNumber(auth, phoneNumber,
appVerifier);
// ↑ 此時 SMS 已發送，使用者會收到驗證碼

// 儲存 confirmationResult，等待使用者輸入 OTP
window.confirmationResult = confirmationResult;
```

重點：

- ✅ **Firebase JS SDK 才能發送 SMS**（這是唯一方式）
- ✅ 返回 `confirmationResult`（包含 verificationId）
- ⚠ 後端 `send-otp` API 是可選的（見下方說明）

步驟 2：前端驗證 OTP（必須！）

```
// 使用者輸入 OTP 後
const otpCode = '123456'; // 使用者輸入的 6 位數驗證碼

try {
  // 2.1 前端驗證 OTP（驗證碼是否正確）
  const result = await window.confirmationResult.confirm(otpCode);

  // 2.2 驗證成功！取得 idToken
  const idToken = await result.user.getIdToken();

  console.log('前端驗證成功，取得 idToken');

  // 2.3 呼叫後端 API
  await callBackendVerifyOTP(idToken, otpCode);

} catch (error) {
  // 前端驗證失敗
  if (error.code === 'auth/invalid-verification-code') {
    alert('驗證碼錯誤，請重新輸入');
  } else if (error.code === 'auth/code-expired') {
    alert('驗證碼已過期，請重新發送');
  } else {
```

```
    alert('驗證失敗：' + error.message);
  }
  // 如果前端驗證失敗，不需要呼叫後端
}
```

重點：

- ✅ `confirm(otpCode)` 驗證 OTP 是否正確
- ✅ 成功後取得 `idToken`（Firebase 簽發的憑證）
- ⚠ 這一步**不能省略**！沒有 confirm 就沒有 idToken

步驟 3：呼叫後端 verify-otp API

```
async function callBackendVerifyOTP(idToken, otpCode) {
  try {
    const response = await fetch('https://ai.akira-
dialog.com/auth/phone/verify-otp/', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        // Session Authentication 會自動帶 cookie
      },
      credentials: 'include', // 重要：帶上 session cookie
      body: JSON.stringify({
        verification_id: idToken,  // ← 使用 idToken，不是 verificationId！
        otp_code: otpCode          // ← 後端僅用於記錄，不用於驗證
      })
    });

    const data = await response.json();

    if (response.ok) {
      // 成功！
      console.log('手機號碼綁定成功：', data.phone_number);
      alert(`驗證成功！手機號碼：${data.phone_number}`);
    } else {
      // 失敗
      console.error('後端驗證失敗：', data);
      handleBackendError(data);
    }

  } catch (error) {
    console.error('API 呼叫錯誤：', error);
    alert('網路錯誤，請稍後再試');
  }
}
```

📩 API 規格：POST /auth/phone/verify-otp/

Request（前端需要傳入）

```
{
  "verification_id": "eyJhbGciOiJSUzI1N...",  // Firebase idToken
  "otp_code": "123456"                        // 使用者輸入的驗證碼
}
```

欄位說明：

欄位	類型	說明	來源
`verification_id`	string	Firebase ID Token（JWT）	`result.user.getIdToken()`
`otp_code`	string	6 位數驗證碼	使用者輸入

重點：

- ⚠ `verification_id` 是 **idToken**，不是 `confirmationResult.verificationId`
- ⚠ `otp_code` 後端僅用於記錄，實際驗證在前端完成

Response（前端會收到）

✅ 成功回應（200 OK）

```
{
  "status": "VERIFIED",
  "phone_number": "+886987654321",
  "message": "手機號碼驗證成功"
}
```

❌ 錯誤回應 1：驗證碼錯誤（400 Bad Request）

```
{
  "status": "INVALID_OTP",
  "remaining_attempts": 2,
  "message": "驗證碼錯誤，您還有 2 次機會"
}
```

❌ 錯誤回應 2：已鎖定（403 Forbidden）

```
{
  "status": "LOCKED",
  "message": "驗證失敗次數過多，請 60 秒後重新發送驗證碼",
```

```
  "retry_after": 60
}
```

❌ **錯誤回應 3：Token 無效（400 Bad Request）**

```
{
  "error": "驗證碼已過期或無效，請重新發送"
}
```

❌ **錯誤回應 4：手機號碼不符（400 Bad Request）**

```
{
  "error": "PHONE_MISMATCH",
  "message": "驗證的手機號碼與您的帳號不符，請確認"
}
```

🔧 後端實作：verify_otp 解釋

修改的邏輯（firebase_service.py）

```
def verify_otp(self, verification_id: str, otp_code: str) -> dict:
    """
    驗證 Firebase ID Token

    Args:
        verification_id: Firebase ID Token（前端驗證成功後取得）
        otp_code: 使用者輸入的驗證碼（用於記錄）

    Returns:
        dict: {'success': bool, 'phone_number': str, 'uid': str, 'error':
str}
    """
    try:
        # 1. 使用 Firebase Admin SDK 驗證 ID Token
        decoded_token = auth.verify_id_token(verification_id)

        # 2. 從 token 中取得已驗證的資訊
        uid = decoded_token.get('uid')                # Firebase User ID
        phone_number = decoded_token.get('phone_number')  # 已驗證的手機號碼

        # 3. 檢查 token 中是否包含手機號碼
        if not phone_number:
            return {
                'success': False,
                'error': 'ID Token 中沒有手機號碼資訊，請確認前端使用 Phone Auth
```

```
方式登入'
            }

        # 4. 驗證成功
        return {
            'success': True,
            'phone_number': phone_number,
            'uid': uid,
            'message': '手機號碼驗證成功'
        }

    except auth.InvalidIdTokenError:
        return {'success': False, 'error': '驗證碼已過期或無效，請重新發送'}
    except auth.ExpiredIdTokenError:
        return {'success': False, 'error': '驗證碼已過期，請重新發送'}
    except Exception as e:
        return {'success': False, 'error': f'驗證失敗：{str(e)}'}
```

後端驗證步驟

```
1️⃣ 驗證 idToken 是否由 Firebase 簽發（防偽造）
    ↓
2️⃣ 驗證 idToken 是否過期
    ↓
3️⃣ 從 idToken 中提取已驗證的手機號碼
    ↓
4️⃣ 檢查手機號碼是否與使用者帳號一致
    ↓
5️⃣ 更新資料庫：綁定手機號碼到使用者
    ↓
6️⃣ 記錄日誌
```

🎯 前端處理錯誤回應

```javascript
function handleBackendError(data) {
  switch (data.status) {
    case 'INVALID_OTP':
      alert(`驗證失敗，您還有 ${data.remaining_attempts} 次機會`);
      break;

    case 'LOCKED':
      alert(`驗證失敗次數過多，請等待 ${data.retry_after} 秒後重新發送`);
      // 可以顯示倒數計時
      showCountdown(data.retry_after);
      break;

    default:
      if (data.error === 'PHONE_MISMATCH') {
```

```
          alert('驗證的手機號碼與您的帳號不符');
        } else {
          alert(`驗證失敗：${data.message || data.error}`);
        }
    }
  }
```

📝 send-otp API 說明

send-otp API 不會發送 SMS！

```
# phone_auth/views.py - send_otp 函式
def send_otp(request):
    # 1. 驗證手機號碼格式 ✅
    # 2. 檢查 Rate Limiting（60秒內不可重複） ✅
    # 3. 檢查手機號碼是否已被其他使用者綁定 ✅
    # 4. 更新使用者狀態（phone_number, verification_status） ✅
    # 5. 記錄日誌 ✅
    # 6. 發送 SMS ❌ <- 不會做這個！
```

是否需要呼叫 send-otp API？

方案 A：不呼叫 send-otp

適合： 追求簡潔、快速開發

```
// 直接用 Firebase 發送
const confirmationResult = await signInWithPhoneNumber(auth, phoneNumber,
appVerifier);
const result = await confirmationResult.confirm(otp);
const idToken = await result.user.getIdToken();

// 直接呼叫 verify-otp
await fetch('/verify-otp/', {
  body: JSON.stringify({ verification_id: idToken, otp_code: otp })
});
```

優點：

- ✅ 流程簡單（少一次 API 呼叫）
- ✅ 減少後端負擔
- ✅ Firebase 本身有 Rate Limiting

缺點：

- ❌ 後端沒記錄發送狀態

- ❌ 要到 verify 時才知道手機號碼已被綁定

方案 B：呼叫 send-otp（完整）

適合： 需要完整記錄、更好的 UX

```
// 1. 先呼叫後端（檢查 + 記錄）
const response = await fetch('/send-otp/', {
  body: JSON.stringify({
    country_code: '+886',
    phone_number: '987654321'
  })
});

if (!response.ok) {
  const error = await response.json();
  if (error.error === 'PHONE_ALREADY_BOUND') {
    alert('此手機號碼已被其他帳號綁定');
    return; // 提前結束，不發送 OTP
  }
}

// 2. 後端檢查通過，才用 Firebase 發送
const confirmationResult = await signInWithPhoneNumber(auth, phoneNumber,
appVerifier);

// 3-4. 驗證流程...
```

優點：

- ✅ 提前檢查手機號碼是否已被綁定（UX 更好）
- ✅ 後端統一管理 Rate Limiting
- ✅ 完整的日誌記錄（方便追蹤和除錯）
- ✅ 記錄使用者狀態變化

缺點：

- ❌ 多一次 API 呼叫
- ❌ 流程稍微複雜

💡 可能會遇到的問題

Q1: 為什麼要驗證兩次？

A: 不是重複驗證，而是驗證不同東西：

- **前端：** 驗證使用者輸入的 OTP 是否正確
- **後端：** 驗證前端拿到的憑證是否真實（防止偽造）

這是安全機制，缺一不可。

Q2: 可以拿掉前端驗證嗎？

A: ❌ 不行！

- 沒有前端 `confirm(otp)` 就拿不到 idToken
- 後端需要 idToken 才能驗證
- 這是 Firebase 架構要求

Q3: verification_id 到底是什麼？

A: 有兩個容易混淆的東西：

名稱	是什麼	從哪來	給誰用
`verificationId`	Firebase 的 session ID	`confirmationResult.verificationId`	前端內部使用
`idToken`	Firebase 簽發的憑證（JWT）	`result.user.getIdToken()`	傳給後端

後端 API 需要的是 `idToken`，不是 `verificationId`！

Q4: otp_code 在後端有用嗎？

A: 僅用於記錄，不用於驗證。

- 真正的驗證在前端完成（`confirm(otp)`）
- 後端只驗證 idToken 的真實性
- 後端的 `otp_code` 參數主要是記錄到 log 中

Q5: 如果前端驗證失敗，還要呼叫後端嗎？

A: ❌ 不需要！

```
try {
  const result = await confirmationResult.confirm(otp);
  const idToken = await result.user.getIdToken();
  // 前端驗證成功，才呼叫後端
  await callBackendAPI(idToken, otp);
} catch (error) {
  // 前端驗證失敗，直接顯示錯誤，不呼叫後端
  alert('驗證碼錯誤');
}
```

🧪 測試方式與設定

方式 1：使用 Firebase 測試手機號碼

不需要真的發送 SMS，可以無限次測試！

步驟 1：在 Firebase Console 設定測試號碼

操作步驟：

1. 前往 Firebase Console
2. 選擇專案
3. 左側選單：**Authentication** → **Sign-in method**
4. 找到 **Phone** → 點擊編輯
5. 展開 **Phone numbers for testing**
6. 加入測試號碼白名單和固定驗證碼：

```
測試號碼 1：+886987654321，驗證碼：123456
測試號碼 2：+886987654322，驗證碼：123456
測試號碼 3：+886987654323，驗證碼：123456
（可加入更多...）
```

7. 儲存

步驟 2：前端測試

```
// 1. 發送 OTP（不會真的發送 SMS）
const confirmationResult = await signInWithPhoneNumber(auth,
'+886987654321', appVerifier);

// 2. 使用你設定的固定驗證碼
const result = await confirmationResult.confirm('123456');  // ← 固定的測試驗
證碼

// 3. 取得 idToken
const idToken = await result.user.getIdToken();

// 4. 呼叫後端
const response = await fetch('/verify-otp/', {
  body: JSON.stringify({ verification_id: idToken, otp_code: '123456' })
});
```

優點：

- ✅ 不消耗 SMS 額度
- ✅ 可以無限次測試

- ✅ 測試碼固定，方便自動化測試

注意：

- ⚠ 測試號碼只能在開發環境使用
- ⚠ 部署到正式環境前要移除測試號碼

方式 2：使用真實手機號碼

如果要測試真實的 SMS 發送：

```
// 使用真實手機號碼
const confirmationResult = await signInWithPhoneNumber(auth,
'+886912345678', appVerifier);

// 等待收到 SMS，輸入真實的驗證碼
const result = await confirmationResult.confirm(realOtpCode);
```

注意：

- ⚠ 會消耗 Firebase SMS 額度
- ⚠ 有發送頻率限制
- ⚠ 需要真實手機接收

🔄 反覆測試的方法

問題：測試一次後，如何重新測試？

情況說明：

- **已驗證成功**：手機號碼已綁定到帳號 → 需要清除綁定狀態才能重新測試
- **驗證失敗**：還沒成功綁定 → 可以直接重試或重置嘗試次數

情況 1：已驗證成功（綁定成功）

方法 1-1：請在 Django Admin 重置

詳細操作步驟：

1. 登入 Django Admin
2. 找到 **Users** 或 **CustomUser**
3. 找到該的帳號，點擊進入編輯
4. **清除以下欄位**（重點！）：

```
✅ phone_verified: □（取消勾選，改為 False）
✅ phone_number: （完全清空）
✅ otp_attempts: 0（改為 0）
✅ verification_status: （選擇「----------」或清空）
✅ last_otp_sent_at: （清空，如果有的話）
```

5. 點擊「儲存」

就可以用同一個測試號碼重新測試驗證流程。

方法 1-2：使用不同的測試號碼

如果設定了多個測試號碼，直接換號碼測試：

- 第 1 次測試用 +886987654321
- 第 2 次測試用 +886987654322（不需要重置！）
- 第 3 次測試用 +886987654323（不需要重置！）

每個號碼都是獨立的，不需要重置！

情況 2：驗證失敗（還沒成功綁定）

方法 2：使用不同的測試手機號碼

在 Firebase Console 加入多個測試號碼：

```
測試號碼 1：+886987654321，驗證碼：123456
測試號碼 2：+886987654322，驗證碼：123456
測試號碼 3：+886987654323，驗證碼：123456
```

每次測試用不同號碼即可。

🐛 測試錯誤訊息的方法

測試 1：前端 OTP 錯誤

```
try {
  // 故意輸入錯誤的 OTP
  await confirmationResult.confirm('000000');
} catch (error) {
  console.log('錯誤代碼：', error.code);
  // 預期：'auth/invalid-verification-code'

  console.log('錯誤訊息：', error.message);
  // 預期：'The SMS verification code used to create the phone auth
```

```
credential is invalid...'
}
```

測試 2：後端 Token 無效

```
// 故意傳假的 idToken
const response = await fetch('/verify-otp/', {
  method: 'POST',
  body: JSON.stringify({
    verification_id: 'fake-invalid-token',
    otp_code: '123456'
  })
});

const data = await response.json();
console.log(data);
// 預期：{ "error": "驗證碼已過期或無效，請重新發送" }
```

測試 3：連續錯誤 3 次被鎖定

```
// 第 1 次錯誤
let response = await fetch('/verify-otp/', {
  body: JSON.stringify({
    verification_id: 'fake-token-1',
    otp_code: '111111'
  })
});
console.log(await response.json());
// 預期：{ "status": "INVALID_OTP", "remaining_attempts": 2, ... }

// 第 2 次錯誤
response = await fetch('/verify-otp/', {
  body: JSON.stringify({
    verification_id: 'fake-token-2',
    otp_code: '222222'
  })
});
console.log(await response.json());
// 預期：{ "status": "INVALID_OTP", "remaining_attempts": 1, ... }

// 第 3 次錯誤
response = await fetch('/verify-otp/', {
  body: JSON.stringify({
    verification_id: 'fake-token-3',
    otp_code: '333333'
  })
});
```

```
console.log(await response.json());
// 預期：{ "status": "LOCKED", "retry_after": 60, ... }

// 第 4 次嘗試（已被鎖定）
response = await fetch('/verify-otp/', {
  body: JSON.stringify({
    verification_id: 'any-token',
    otp_code: '444444'
  })
});
console.log(await response.json());
// 預期：{ "status": "LOCKED", "retry_after": 60, ... }
```

測試 4：Rate Limiting（60 秒內重複發送）

```
// 第 1 次發送
await fetch('/send-otp/', {
  body: JSON.stringify({
    country_code: '+886',
    phone_number: '987654321'
  })
});
// 預期：成功

// 立即第 2 次發送
await fetch('/send-otp/', {
  body: JSON.stringify({
    country_code: '+886',
    phone_number: '987654321'
  })
});
// 預期：{ "status": "TOO_MANY_REQUESTS", "retry_after": xx, ... }
```

測試 5：手機號碼已被其他使用者綁定

```
// 假設 +886987654321 已被使用者 A 綁定
// 使用者 B 嘗試綁定同號碼

await fetch('/send-otp/', {
  body: JSON.stringify({
    country_code: '+886',
    phone_number: '987654321'  // 已被綁定
  })
});
// 預期：{ "error": "PHONE_ALREADY_BOUND", "message": "此手機號碼已被其他帳號綁
定" }
```

測試 6：手機號碼格式錯誤

```
// 測試：缺少國碼
await fetch('/send-otp/', {
  body: JSON.stringify({
    country_code: '886',  // ❌ 缺少 +
    phone_number: '987654321'
  })
});
// 預期：400 Bad Request

// 測試：號碼太短
await fetch('/send-otp/', {
  body: JSON.stringify({
    country_code: '+886',
    phone_number: '123'  // ❌ 太短
  })
});
// 預期：400 Bad Request
```