


# • Облікові записи та транзакції в Algorand



# План

- 1) Обліковий запис в блокчейні Algorand. Типи облікових записів
- 2) Гаманці в Algorand. Типи криптогаманців
- 3) Транзакції. Типи транзакцій в Algorand
- 4) Групові транзакції
- 5) Внутрішні транзакції
- 6) Створення транзакцій з використанням JS AlgoSDK

The background features a dark blue, almost black, field filled with intricate, flowing patterns of lighter blue lines. These lines create a sense of depth and movement, resembling a complex network or a topographical map. Scattered throughout the scene are numerous small, bright white and light blue dots, which appear to be particles or data points, adding a digital or scientific feel to the overall aesthetic.

Algorand — це блокчейн-  
платформа на основі  
облікових записів





# Blockchain account

**Обліковий запис** у блокчейні (account) - це цифровий запис, який містить інформацію про стан власника у блокчейн-мережі. Кожен обліковий запис має унікальну адресу, яка використовується для ідентифікації користувача та взаємодії з блокчейном.

Користувачі використовують обліковий запис Blockchain, щоб:

- тримати криптовалюту та токени в мережі блокчейн;
- відправляти та отримувати криптовалюту або токени;
- взаємодіяти з блокчейн додатками;
- підписувати транзакції.

# Структура облікового запису

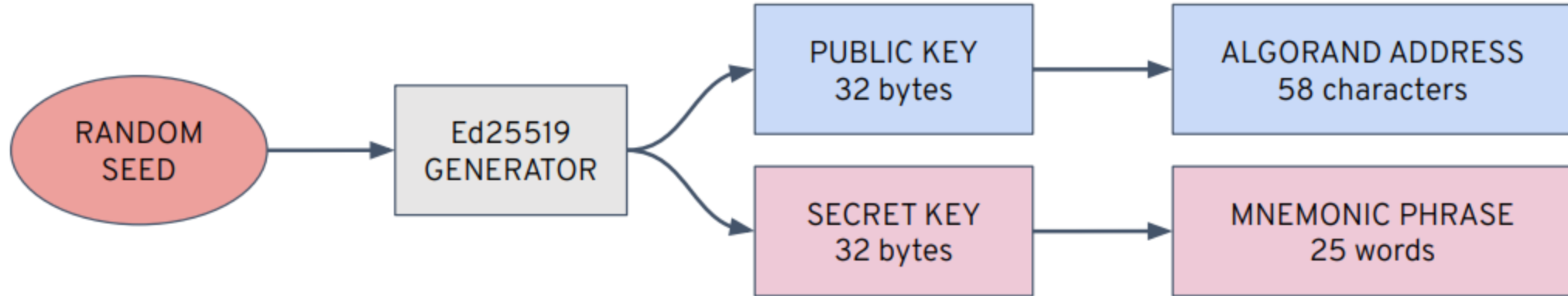
Під час створення нового облікового запису генерується пара ключів: приватний і публічний.

- **Відкритий ключ** — ключ, який генерується на основі закритого ключа. Відкритий ключ використовується для створення адреси облікового запису і може бути публічно доступний.
- **Закритий ключ** — секретний ключ, який використовується для підписання транзакцій. Закритий ключ повинен залишатися конфіденційним, оскільки будь-хто, хто має доступ до нього, може керувати обліковим записом.



# Мнемонічна фраза

- Це набір з 25 BIP39 англійських слів, які використовуються для відновлення облікового запису. Мнемонічна фраза генерується на основі закритого ключа та може використовуватись для відновлення облікового запису в разі втрати доступу до нього.
- Вона використовується для резервного копіювання і відновлення облікового запису. Збереження мнемонічної фрази є критично важливим, оскільки втратити доступ до неї означає втратити доступ до облікового запису.



**Ed25519** - це алгоритм, який використовується для генерації ключів і підписання транзакцій в Algorand.

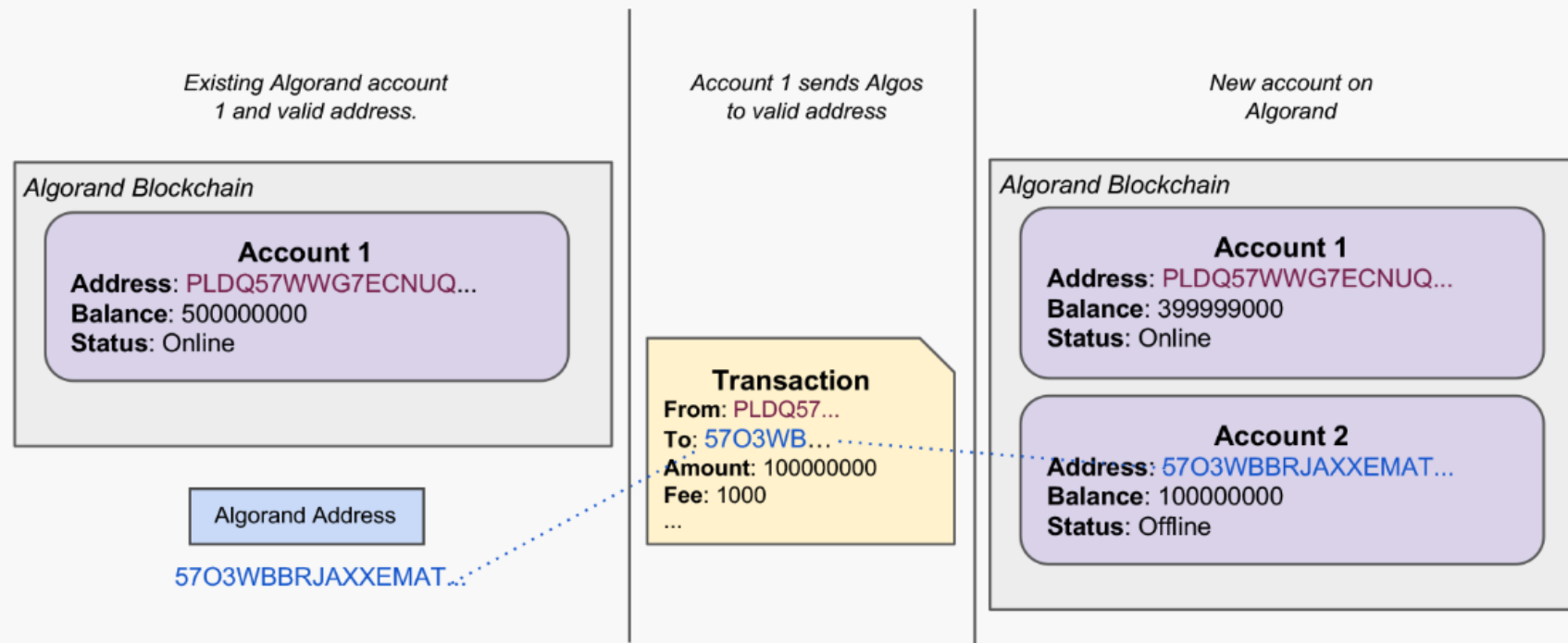
**ADDRESS: відкритий ключ** перетворюється на **адресу Algorand** шляхом додавання 4-байтової контрольної суми в кінець відкритого ключа та **кодування її в base32**.

**MNEMONIC:** мнемоніка з 25 слів створюється шляхом перетворення байтів приватного ключа в 11-бітні цілі числа, а потім відображення цих цілих чисел у список англійських слів bip-0039.

# Algorand Accounts

**Облікові записи** – це об'єкти в блокчейні Algorand, пов'язані з певним локальним станом у мережі, наприклад з балансом, створені додатками та активами.

**Адреса Algorand** – це унікальний ідентифікатор облікового запису Algorand. Вона складається з 58 символів і генерується на основі відкритого ключа.



Після генерації закритого ключа та відповідної адреси, надсилання Algos на адресу в Algorand ініціалізує її стан у блокчейні Algorand.



# Інформація про акаунт в Dappflow

Адреса облікового запису

Поточний баланс в Algo

Активи, створені даним обліковим записом

Історія транзакцій облікового запису

Account : FXA6NGWY6MR2Y5Q7ZPFH7CC7OWHX5JAIED22235KH6RPK55GHUB5JH4O54

Show QRLocal statesView Raw JSON

Offline

Balance999,999,995.971▲

Minimum balance4.983▲








Holding assets0

Created assets0

Created applications19

Opted applications0

TransactionsAssetsCreated assetsCreated applicationsOpted applications

Txn ID	Block	Age	From	To	Amount	Fee	Type
 <a href="#">SJMLCJDCIK5456O2TN...</a>	<a href="#">12828</a>	1 day, 4 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT <a href="#">Application: 1078</a>		▲ 0.001	App call
 <a href="#">BCODZMWSER76J2EJAL...</a>	<a href="#">12793</a>	1 day, 4 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT <a href="#">Application: 1078</a>		▲ 0.001	App call
 <a href="#">LOAKGN2K4JWSPFY5EK...</a>	<a href="#">12646</a>	1 day, 4 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT <a href="#">Application: 1078</a>		▲ 0.001	App call
  <a href="#">AG6SPP4DB232YIJJV...</a>	<a href="#">10838</a>	1 day, 10 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT <a href="#">CWP3BUY5KM553OL...</a>	▲ 2	▲ 0.001	Payment
  <a href="#">LDTHLAIMOJ3QFAXA3E...</a>	<a href="#">10838</a>	1 day, 10 hours ago	FXA6NGWY6MR2Y5Q7ZPFH...	OUT <a href="#">Application: 1068</a>		▲ 0.001	App call



## Мінімальний баланс для акаунтів Algorand

Кожен обліковий запис на Algorand повинен мати **мінімальний** баланс **100 000 microAlgos**.

Мінімальний баланс **збільшується** з кожним активом, який є в обліковому записі (незалежно від того, чи був актив створений або належить обліковому запису), а також з кожним додатком, який обліковий запис створив або у який був включений.

# Збільшення мінімального балансу


## Активи:

- Для кожного активу, створеного або яким володіє обліковий запис, його мінімальний баланс **збільшується на 0,1 Algos** (100 000 microAlgos).
- Обліковий запис може будь-коли відмовитися від активу. Це означає, що обліковий запис більше не зберігатиме актив, і обліковий запис більше не зможе отримати актив. Також **зменшується** вимога до мінімального балансу на **0,1 Algo**.

## Смарт-контракти:

- Під час створення чи підключення до смарт-контракту мінімальний баланс для облікового запису буде збільшено. Сума, на яку він буде збільшений, залежатиме від обсягу сховища в мережі, яке використовується. (Лекція 2. Слайд: Вимоги до мінімального балансу для смарт-контракту<sup>¶</sup>)

\*Якщо буде здійснено будь-яку транзакцію, яка порушить вимоги до мінімального балансу, транзакція буде відхилена.

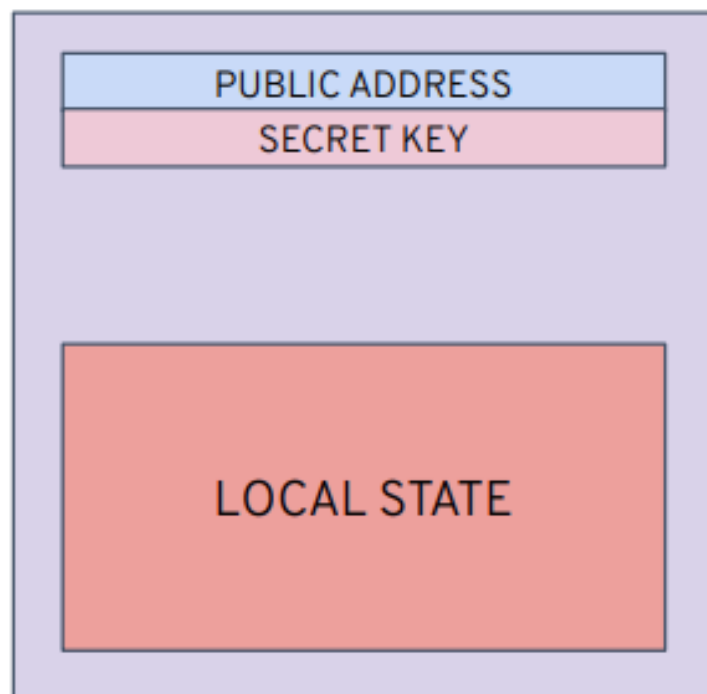


# Типи облікових записів

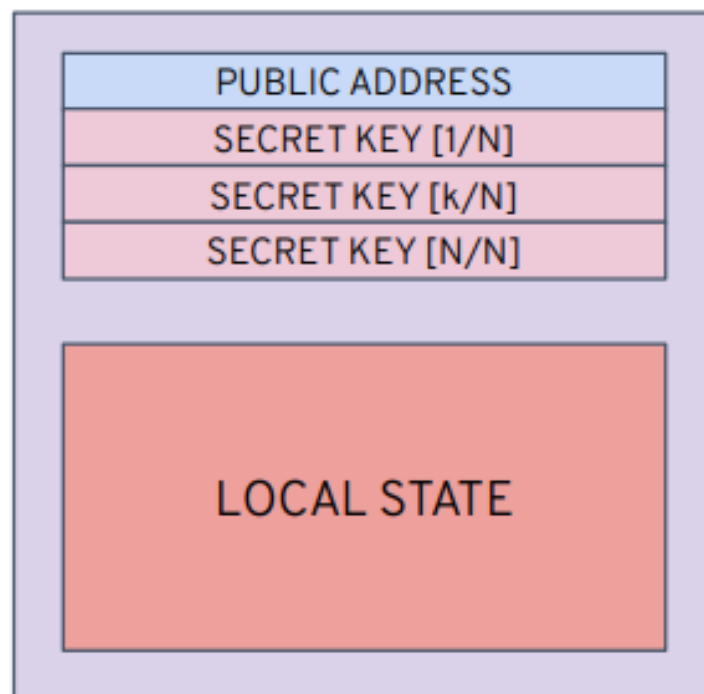
- **Стандартний обліковий запис:** використовується для зберігання ALGO та інших активів, а також для здійснення транзакцій.
- **Мультипідписний обліковий запис (Multisignature Account):** вимагають кілька підписів до виконання транзакції. Вони складаються зі списку адрес, версії та порогового значення, яке вказує, скільки підписів необхідно для підтвердження транзакції.
- **Обліковий запис смарт-контракту (Contract Account):** всі розгорнуті смарт-контракти мають власний обліковий запис додатку з пов'язаною публічною адресою Algorand. Ці облікові записи використовуються для здійснення внутрішніх транзакцій із смарт-контракту. .



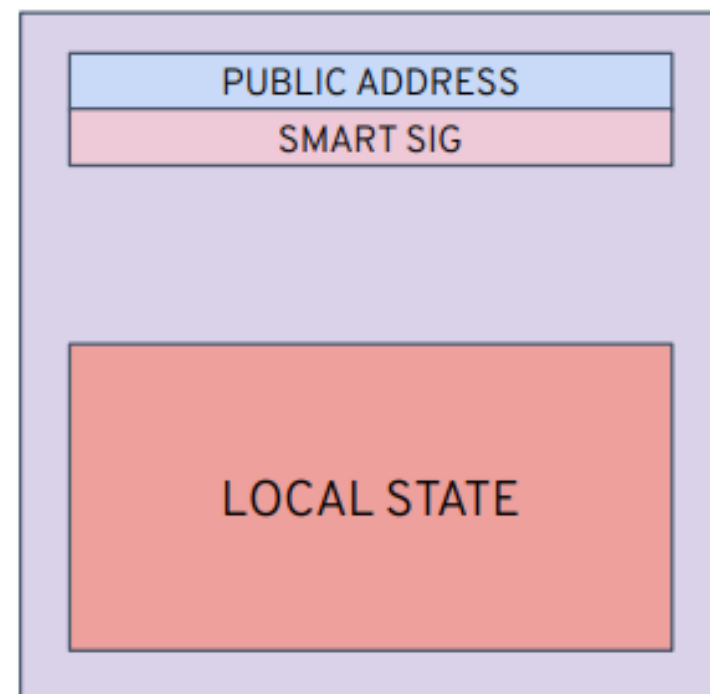
### STANDARD ACCOUNT



### MULTI SIGNATURE ACCOUNT



### CONTRACT ACCOUNT



# Мультипідписний обліковий запис

01

Облікові записи з мультипідписом можуть виконувати ті самі операції, що й інші облікові записи, включаючи надсилання транзакцій та участь у консенсусі.

02

Адреса мультипідписного облікового запису створюється шляхом хешування списку адрес, версії та порогового значення. Порядок адрес важливий – зміна порядку змінить адресу.

03

Поріг визначає, скільки підписів потрібно для обробки будь-якої транзакції з цього облікового запису з мультипідписом.

04

Як і для інших облікових записів, щоб активувати мультипідписну адресу в блокчейні, потрібно надіслати на неї кілька Algos.

# Обліковий запис смарт-контракту

01

Щоб використовувати програму TEAL як обліковий запис, необхідно надіслати Algos на його адресу, щоб перетворити його на обліковий запис на Algorand із балансом.

02

Зовні цей обліковий запис нічим не відрізняється від будь-якого іншого облікового запису Algorand, і кожен може надіслати йому Algos або Algorand Standard Assets, щоб збільшити його баланс.

03

Особливістю contract account є те, що саме логіка смарт-контракту визначає чи буде схвалено транзакцію.

04

Контрактні рахунки потребують Algos для сплати комісій за транзакції та участі в операціях. Збільшити його баланс можна надіславши Algos на адресу контрактного облікового запису.

**Крипто-гаманець** - це програма, яка функціонує як гаманець для криптовалюти. Він називається гаманцем, тому що він використовується аналогічно гаманцю, в який ви кладете готівку та карти. Замість зберігання цих фізичних предметів, він зберігає ключі доступу, які використовуються для підпису транзакцій з криптовалютою, і надає інтерфейс, який дозволяє користувачам отримати доступ до їх криптовалюти.



# Гаманці

## 01

Гаманці блокчейну не містять реальної валюти. Гаманці використовуються для зберігання закритих ключів та підтримки балансу транзакцій. Реальні дані або валюта зберігаються у блоках у блокчейні.

## 02

Гаманці містять адресу та закриті ключі, необхідні для підпису транзакцій. Будь-хто, хто знає закритий ключ, може контролювати монети, пов'язані з цією адресою.

# Типи криптовалютних гаманців

Залежно від принципу роботи, виділяють гарячі (програмні) та холодні (апаратні) криптогаманці.

- **Гарячі гаманці** — це мобільні, десктопні та вебдодатки.
- **Холодні гаманці** мають фізичну форму, це спеціальні пристрої, які можна підключити до ПК. Вони зберігають закриті ключі до криптовалюти користувача в автономному режимі.

За типом управління даними, поділяють гаманці на дві групи: кастодіальні та некастодіальні.

- **Кастодіальні сервіси**, такі як криптовалютні біржі, зберігають публічні та приватні ключі на власних серверах.
- **Некастодіальні гаманці** дозволяють користувачеві самостійно зберігати ключі на пристрої. Гаманці цього типу не покладаються на третю особу чи “кастодіана” для забезпечення безпеки криптовалюти користувача. Під час створення облікового запису в такому гаманці користувач вказує фразу відновлення, тому він може отримати доступ до криптовалюти, навіть якщо ключі були втрачені.



Гаманець Algorand – це програмне або апаратне забезпечення, яке надійно зберігає закриті та відкриті ключі, що дозволяє користувачам зберігати, отримувати доступ, надсилати та отримувати Algo та інші активи



# Програмні гаманець в Algorand

Основні характеристики гаманця:

- **Кілька облікових записів:** один гаманець може керувати кількома обліковими записами.
- **Керування ключами:** гаманець безпечно зберігає та керує криптографічними ключами (закритими та відкритими) для кожного облікового запису.
- **Інтерфейс користувача:** гаманці пропонують інтерфейси (веб-інтерфейси, десктоп або мобільні додатки), що дозволяють користувачам взаємодіяти з блокчейном, відправляти та отримувати монети, переглядати історію транзакцій та багато іншого.



# Гаманці Algorand



Android - iOS - Browser extension -  
Desktop



Android - iOS



Hardware wallet



Android - iOS - Web



## Транзакції. Зміна стану блокчейну

Транзакція є основним елементом блоків, яка визначає еволюцію стану розподіленого реєстру. Вона є записом, який додається до блокчейн-реєстру і включає в себе деталі про передачу цифрових активів між учасниками або інші дії в мережі (створення додатку, створення активів, виклик додатку тощо).

У протоколі Algorand є сім типів транзакцій:

- [Payment](#)
- [Key Registration](#)
- [Asset Configuration](#)
- [Asset Freeze](#)
- [Asset Transfer](#)
- [Application Call](#)
- [State Proof](#)

# Типи транзакцій

- **Платіжна транзакція** (Payment): відправляє Algos з одного рахунку на інший.
- **Транзакція передачі активів** (Asset Transfer): використовується для згоди на отримання певного типу ASA (Algorand Standard Asset), передачі ASA з одного рахунку на інший.
- **Транзакція конфігурації активів**: AssetConfigTx використовується для створення активу, зміни певних параметрів активу або знищення ASA.
- **Транзакція заморожування активів** (Asset Freeze) : дозволяє заморозити або розморозити активи для певного рахунку. Коли актив для облікового запису заморожено, обліковий запис не може надіслати або отримати цей актив.
- **Транзакція реєстрації ключа** (Key Registration): використовується для реєстрації ключів участі та онлайн-статусу для облікового запису Algorand. Ця транзакція є важливою для користувачів, які хочуть взяти участь у механізмі консенсусу Algorand (Pure Proof of Stake), ставши валідатором.

# Типи транзакцій

**Транзакція виклику додатку** (Application Call): надсилається в мережу з AppId і методом OnComplete. AppId визначає, яку програму потрібно викликати, а метод OnComplete використовується в контракті, щоб визначити, яку гілку логіки потрібно виконати.

Транзакції виклику програми можуть містити інші поля, необхідні логіці, наприклад:

- ApplicationArgs - для передачі довільних аргументів до смарт-контракту
- Accounts - для передачі облікових записів, для яких може знадобитися деяка перевірка балансу або статус підключення
- ForeignApps — для передачі програм та дозволу доступу до стану зовнішньої програми
- ForeignAssets – передача ASA для перевірки параметрів
- Boxes – для передачі посилань на коробки додатків, щоб AVM міг отримати доступ до вмісту



# Приклад платіжної операції

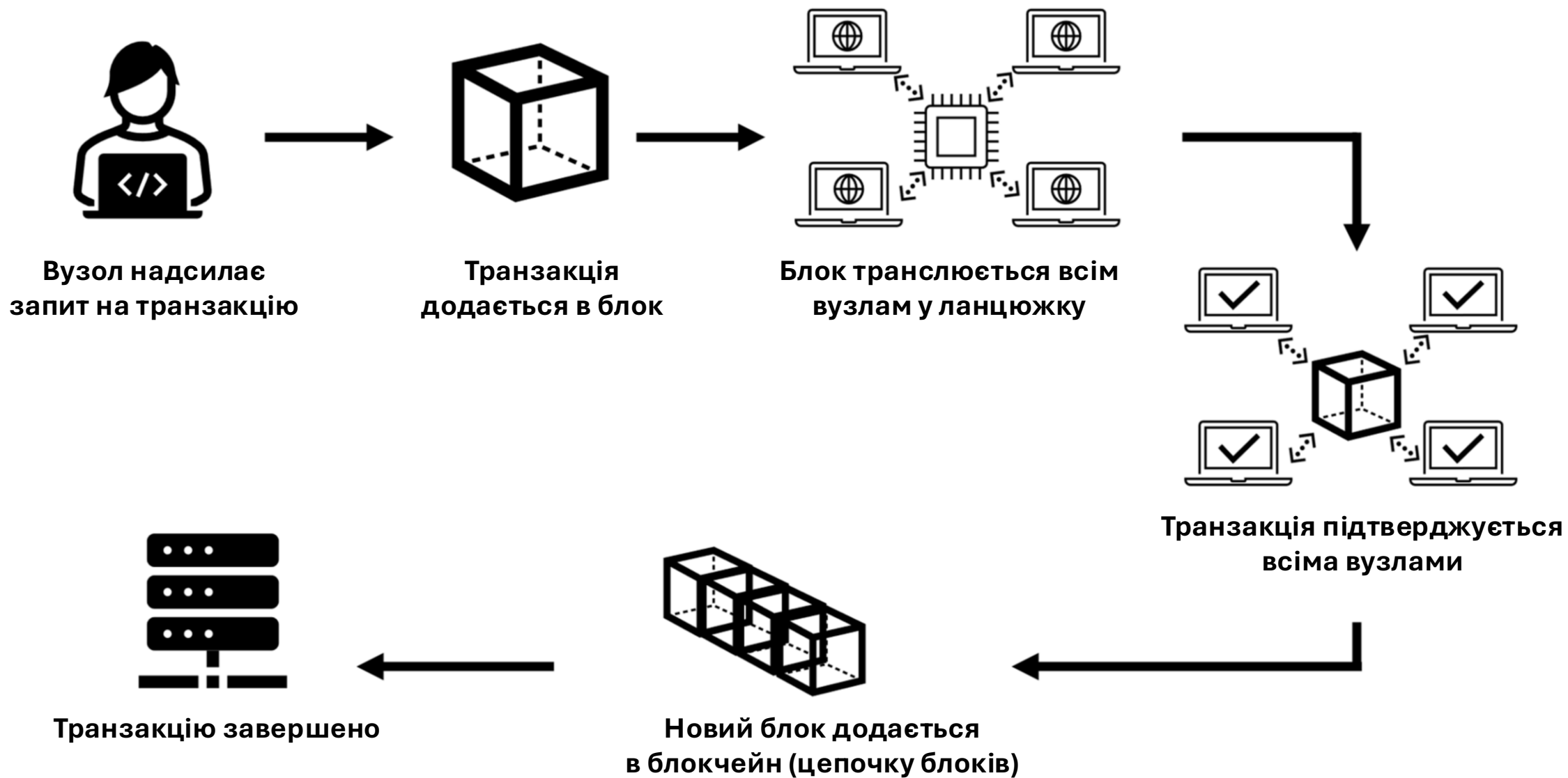
Приклад транзакції, яка надсилає 5 ALGO з одного облікового запису на інший у MainNet

```
{
  "txn": {
    "amt": 5000000,
    "fee": 1000,
    "fv": 6000000,
    "gen": "mainnet-v1.0",
    "gh": "wGHE2Pwvdvd7S12BL5Fa0P20EGYesN73ktiC1qzkkit8=",
    "lv": 6001000,
    "note": "SGVsbG8gV29ybGQ=",
    "rcv": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQ0IJVFDPPXWEG3FVOJCCDBBHU5A",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCLIHZU6TBEOC7XRSBG4",
    "type": "pay"
  }
}
```

First valid: Номер першого блоку, в якому транзакція може бути включена

Note: Додаткова інформація, пов'язана з транзакцією, закодована у Base64

Genesis ID: Ідентифікатор генезис-блоку, який визначає мережу, до якої належить ця транзакція







# Атомарні транзакції

---

Атомарна транзакція в Algorand — це спосіб згрупувати кілька транзакцій разом, щоб вони виконувалися як єдине ціле. Якщо будь-яка транзакція в групі завершується невдачею, усі транзакції в групі не вдаються. Це гарантує, що всі дії виконано успішно або жодна.




# Випадки використання



Атомарні передачі можуть використовуватися в таких випадках:

- **Групові платежі:** платять усі чи не платить ніхто.
- **Розподілені платежі:** платежі кільком одержувачам. Коли потрібно здійснити виплати кільком одержувачам одночасно, забезпечуючи, що всі одержувачі отримають свої виплати або жоден з них не отримає.
- **Об'єднані комісії за транзакції:** одна транзакція оплачує комісії інших. Один з учасників угоди може покрити комісії за всі транзакції в групі, забезпечуючи, що всі операції будуть виконані одночасно без збоїв через недостатність коштів на оплату комісій.
- **Обмін активами:** обмін одного активу на інший між двома або більше сторонами. Групові транзакції забезпечують, що обмін відбудеться тільки в разі успішного виконання всіх транзакцій, пов'язаних з обміном.



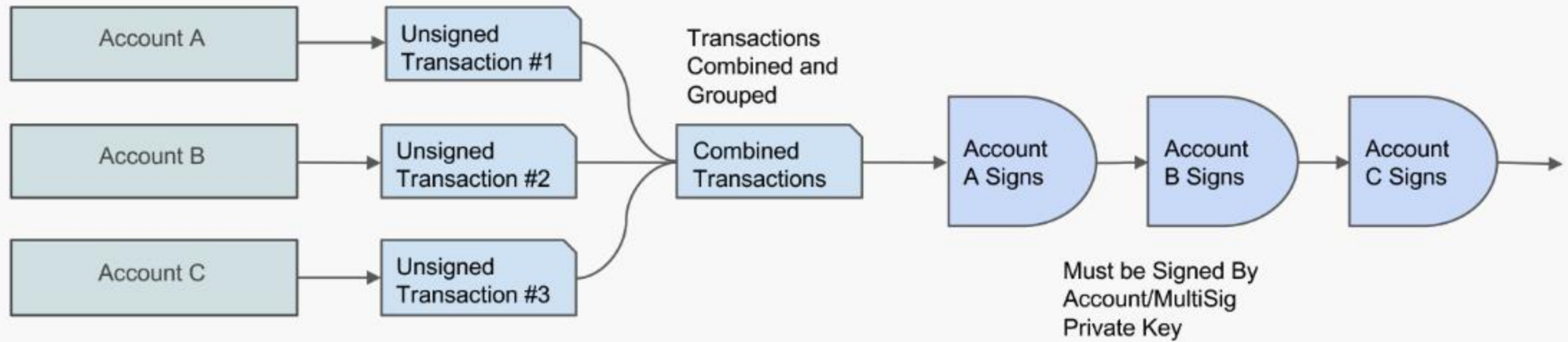
# Групові транзакції

Групові транзакції — це реалізація атомарних транзакцій в Algorand. Ось як вони працюють і що включають:

- 1. Створення групових транзакцій:** щоб створити групу транзакцій, вам потрібно вказати кілька окремих транзакцій (не більше 16), які ви хочете включити до групи. Кожна транзакція матиме унікальний ідентифікатор.
- 2. Призначення ідентифікатора групи:** усім транзакціям у групі призначається загальний ідентифікатор групи. Цей ідентифікатор гарантує, що транзакції пов'язані разом і повинні оброблятися разом.
- 3. Перевірка та надсилання:** коли група транзакцій надсилається в мережу Algorand, мережа перевіряє всю групу. Якщо будь-яка транзакція в групі недійсна або не виконується з будь-якої причини, уся група відхиляється.



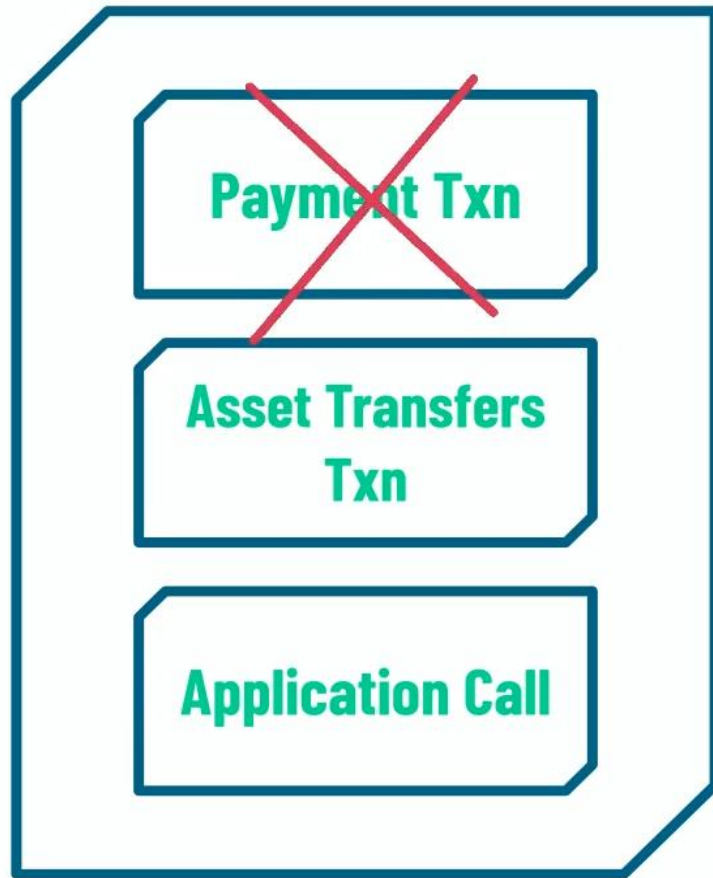
Create Unsigned Transactions



Initial Transactions  
can be created by 1  
or more Accounts or  
MultiSig Accounts



## Atomic Group



## Atomic Group





## Внутрішні транзакції

Внутрішні транзакції в Algorand — це транзакції, які створюються і виконуються смарт-контрактами (або "додатки" в термінології Algorand) під час їхнього виконання. Вони дозволяють смарт-контрактам ініціювати передачу активів або здійснювати інші дії без необхідності зовнішнього втручання або створення окремих транзакцій поза контрактом.

Смарт-контракт може виконати до 256 внутрішніх транзакцій за один виклик.

Якщо будь-яку з цих транзакцій не буде виконано, смарт-контракт також не буде виконано.

Комісію за внутрішні транзакції оплачується смарт-контрактом та автоматично встановлюється на рівні мінімальної комісії за транзакцію. Внутрішні комісії за транзакцію можуть бути об'єднані в пул комісій, як і будь-яка інша транзакція. Це дозволяє або виклику програми, або будь-якій іншій транзакції групи транзакцій оплачувати комісію за внутрішні транзакції.

Внутрішні транзакції оцінюються під час виконання AVM, що дозволяє бачити зміни у контракті. Наприклад, якщо код операції «баланс» використовується до і після відправки транзакції «оплата», зміна балансу буде видно контракту, що виконує.

Одержувач внутрішньої транзакції має бути у масиві Accounts транзакції виклику додатку.

## Випадки використання

- Смарт-контракт може слугувати рахунком умовного депонування та утримувати кошти, а потім пересилати їх після виконання певних умов, визначених логікою смарт-контракту.
- Смарт-контракти можуть взаємодіяти між собою, використовуючи внутрішні транзакції для виклику функцій інших смарт-контрактів або передачі їм активів.
- Смарт-контракт може створювати нові активи і автоматично розподіляти їх між користувачами. Це може використовуватися для випуску нових токенів або NFT.



# Створення внутрішньої транзакції

Для створення внутрішньої транзакції використовуються коди операцій `itxn_begin`, `itxn_field`, `itxn_next` та `itxn_submit`.

- `itxn_begin` означає початок внутрішньої транзакції.
- `itxn_field` використовується для встановлення певних властивостей транзакції.
- `itxn_next` переходить до наступної транзакції в тій же групі, що й попередня, а код операції
- `itxn_submit` використовується для надсилання транзакції або групи транзакцій.

# Приклад

Далі наведено звичайну платіжну транзакцію, яка пересилає 5 Algo з рахунку смарт-контракту на рахунок визначеного отримувача.

```
# ...
InnerTxnBuilder.Begin(),
InnerTxnBuilder.SetFields(
    {
        TxnField.type_enum: TxnType.Payment,
        TxnField.amount: Int(5000),
        TxnField.receiver: Txn.sender(),
    }
),
InnerTxnBuilder.Submit(),
```

Відправником транзакції за замовчуванням є обліковий запис смарт-контракту.



# JavaScript AlgoSDK

---





# Algorand SDKs

Algorand надає SDK для різних мов програмування, що дозволяє розробникам вибирати найбільш зручний інструмент для своїх потреб. Основні Algorand SDK включають:

- JavaScript SDK
- Python SDK
- Java SDK
- Go SDK

JavaScript AlgoSDK — це бібліотека, що надає зручний інтерфейс для взаємодії з блокчейном Algorand. Використовуючи AlgoSDK, розробники можуть створювати, підписувати та надсилати транзакції, працювати зі смарт-контрактами, створювати та керувати активами, а також виконувати багато інших операцій

# Встановлення та налаштування AlgoSDK

Для початку роботи з AlgoSDK необхідно встановити бібліотеку через npm (Node Package Manager):

```
bash
```

```
npm install algosdk
```

Після встановлення бібліотеки можна імпортувати її в проект:

```
javascript
```

```
const algosdk = require('algosdk');
```

# Підключення до Algorand Node

Для взаємодії з блокчейном Algorand необхідно підключитися до вузла (node). Можна використовувати як власний вузол, так і сторонні сервіси.

Підключення до Algorand Sandbox:

```
const algodToken = 'a'.repeat(64);  
const algodServer = 'http://localhost';  
const algodPort = 4001;  
  
const algodClient = new algosdk.Algodv2(algodToken, algodServer, algodPort);
```

- **token:** Токен аутентифікації для доступу до sandbox node. Значення "a" \* 64 зазвичай використовується для локальних sandbox node.
- **server:** Адреса sandbox node. Для локального sandbox використовується <http://localhost>.
- **port:** Порт, на якому працює ваш sandbox node. Для локального sandbox зазвичай використовується порт 4001.

# Створення облікового запису

Щоб взаємодіяти з блокчейном Algorand, необхідно мати поповнений обліковий запис. Створення тестового облікового запису відбувається наступним чином:

```
const generatedAccount = algosdk.generateAccount();  
const passphrase = algosdk.secretKeyToMnemonic(generatedAccount.sk);  
console.log(`My address: ${generatedAccount.addr}`);  
console.log(`My passphrase: ${passphrase}`);
```



# Перевірка балансу облікового запису

```
const acctInfo = await algodClient.accountInformation(acct.addr).do();  
console.log(`Account balance: ${acctInfo.amount} microAlgos`);
```

# Створення платіжної транзакції

```
const suggestedParams = await algodClient.getTransactionParams().do();
const ptxn = algosdk.makePaymentTxnWithSuggestedParamsFromObject({
  from: acct.addr,
  suggestedParams,
  to: acct2.addr,
  amount: 10000,
  note: new Uint8Array(Buffer.from('hello world')),
});
```

- **getTransactionParams():** цей метод викликає API, щоб отримати рекомендовані параметри для нової транзакції, такі як поточний мінімальний fee, останній блок (first valid round) і т.д.
- **algosdk.makePaymentTxnWithSuggestedParamsFromObject:** ця функція створює нову платіжну транзакцію з використанням рекомендованих параметрів.
- **from:** Адреса відправника транзакції. Це той, хто ініціює транзакцію.
- **suggestedParams:** Параметри транзакції, отримані з попереднього кроку (suggestedParams).
- **to:** Адреса отримувача транзакції. Це той, хто отримує Algos.
- **amount:** Сума Algos, що буде надіслана в транзакції. У цьому випадку це 10000 microAlgos (0.01 Algos).
- **note:** Примітка до транзакції, яка може містити додаткову інформацію в форматі байтів. У цьому випадку це 'hello world', конвертоване в Uint8Array.

# Підписання транзакції

Перш ніж транзакція буде вважатися дійсною, вона повинна бути підписана закритим ключем відправника:

```
const signedTxn = ptxn.signTxn(acct.privateKey);
```

# Надсилання транзакції

Підписану транзакцію тепер можна надіслати в мережу. `WaitForConfirmation` викликається після відправки транзакції, щоб дочекатися, поки транзакція буде передана в блокчейн Algorand і підтверджена.

```
const { txId } = await algodClient.sendRawTransaction(signedTxn).do();
const result = await algosdk.waitForConfirmation(algodClient, txId, 4);
console.log(result);
console.log(`Transaction Information: ${result.txn}`);
console.log(`Decoded Note: ${Buffer.from(result.txn.txn.note).toString()}`);
```

# Відновлення облікового запису з мнемонічної фрази

```
const mnemonic =  
  'creek phrase island true then hope employ veteran rapid hurdle above liberty tissue connect alcohol timber idle ten frog bulb embody crunch taxi abstract month';  
const recoveredAccount = algosdk.mnemonicToSecretKey(mnemonic);  
console.log('Recovered mnemonic account: ', recoveredAccount.addr);
```

# Визначення схеми для глобального та локального станів

```
// define uint64s and byteslices stored in global/local storage
const numGlobalByteSlices = 1;
const numGlobalInts = 1;
const numLocalByteSlices = 1;
const numLocalInts = 1;
```



## Визначення вихідного TEAL з рядка або файлу

```
const approvalProgram = fs.readFileSync(  
  path.join(__dirname, '/application/approval.teal'),  
  'utf8'  
);  
const clearProgram = fs.readFileSync(  
  path.join(__dirname, '/application/clear.teal'),  
  'utf8'  
);
```

# Компіляція програми

```
const approvalCompileResp = await algodClient
  .compile(Buffer.from(approvalProgram))
  .do();
```

```
const compiledApprovalProgram = new Uint8Array(
  Buffer.from(approvalCompileResp.result, 'base64')
);
```

```
const clearCompileResp = await algodClient
  .compile(Buffer.from(clearProgram))
  .do();
```

```
const compiledClearProgram = new Uint8Array(
  Buffer.from(clearCompileResp.result, 'base64')
);
```

# Транзакція створення додатку

Створення транзакції з визначеними значеннями, підпис, надсилання та підтвердження транзакції:

```
const appCreateTxn = algosdk.makeApplicationCreateTxnFromObject({
  from: creator.addr,
  approvalProgram: compiledApprovalProgram,
  clearProgram: compiledClearProgram,
  numGlobalByteSlices,
  numGlobalInts,
  numLocalByteSlices,
  numLocalInts,
  suggestedParams,
  onComplete: algosdk.OnApplicationComplete.NoOpOC,
});

// Sign and send
await algodClient
  .sendRawTransaction(appCreateTxn.signTxn(creator.privateKey))
  .do();
const result = await algosdk.waitForConfirmation(
  algodClient,
  appCreateTxn.txID().toString(),
  3
);
// Grab app id from confirmed transaction result
const appId = result['application-index'];
console.log(`Created app with index: ${appId}`);
```

# Додаткові матеріали

- [Learn How to Manage Private / Public Algorand Account Keys \[Accounts Explained #2\]](#)
- [The 4 A's of Algorand](#): розглядаються примітиви рівня 1 блокчейну Algorand. Розробники дізнаються, що таке 4 A Algorand (Accounts, Assets, Atomic Transactions and Applications ), як вони пов'язані між собою та як складені в надійні децентралізовані рішення.
- [Algorand JavaScript SDK Examples](#): репозиторій Algorand JavaScript SDK містить приклади використання Algorand JavaScript SDK.
- [Interact with smart contracts](#): офіційна документація Algorand про взаємодію зі смарт-контрактами з використанням AlgoSDK.