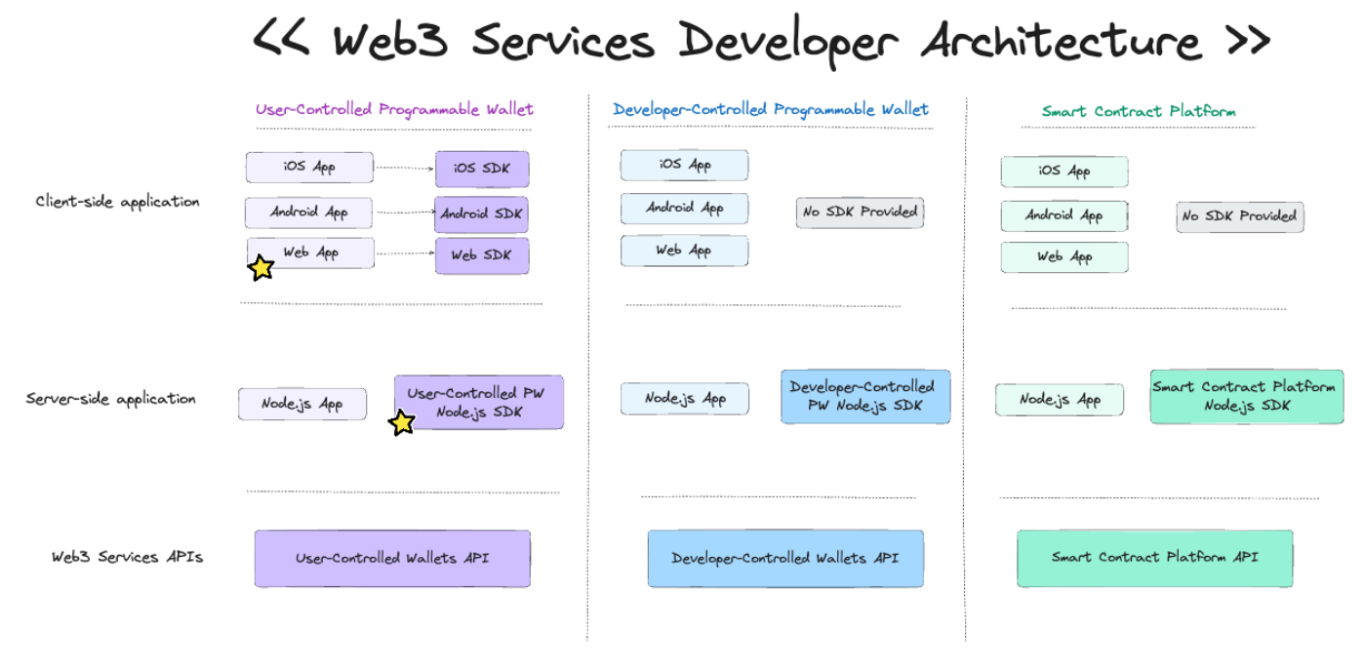
* Now that you're familiar with the concept of Programmable Wallets, it's time to dive into hands-on interaction with them!

In this quest, we'll be focusing on Circle’s recently launched Web3 Services SDKs, specifically the Server-side and Web SDKs. These SDKs are designed to simplify the development process by providing developers with direct, programmatic access to Circle's Web3 Services APIs. They also facilitate the seamless integration of these services into web applications, improving the developer experience and promoting the adoption of Web3 technologies.



Moreover, you'll have the chance to create your very first Smart Contract Account (SCA) User-Controlled Programmable Wallet. This wallet introduces gas abstraction, allowing you to sponsor gas fees on behalf of your users with Circle’s gas station. This eliminates the need for users to hold native tokens to pay for gas, thereby enhancing the overall user experience by offering a smooth, gas-free transaction experience.

Let’s proceed to the project setup!

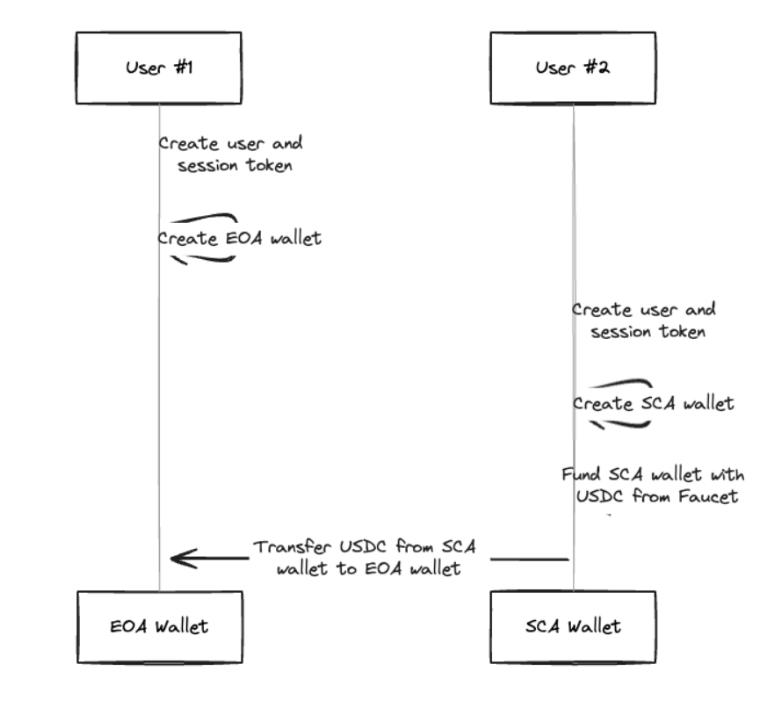
**Project Setup**

Before diving into the coding aspect, let's pause for a moment to understand the high-level workflow of what we're about to undertake in this quest.

We will be setting up two users. **User #1** will establish a User-Controlled Wallet, while **User #2** will set up a Smart Contract Account (SCA) User-Controlled Wallet.

**User #2** will then proceed to fund their SCA wallet with USDC, utilizing a faucet for this purpose. Following this, they will execute an outbound transfer of USDC from their SCA wallet to the External Owned Account (EOA) wallet of **User #1**.

This step is designed to showcase the seamless, gas-free experience enabled by the SCA User-Controlled Wallet.

Quest Preview

Now that you have a better understanding of the sequence of events, let’s set up the project!

To begin, run the following command on your terminal in your desired directory to clone the GitHub repository.

git clone https://github.com/clement-stackup/circle-bounty.git

Next, proceed to open the project folder on your IDE and run the following command to install the necessary dependencies!

npm install

Once you have installed the necessary dependencies, proceed to create a new **.env** file under the root directory of the project and paste the following code snippet to the **.env** file.

API\_KEY="ENTER API KEY HERE"

USER\_TOKEN\_1="ENTER USER TOKEN HERE FOR USER #1"

ENCRYPTION\_KEY\_1="ENTER ENCRYPTION KEY HERE FOR USER #1"

WALLET\_ID="ENTER WALLET ID HERE"

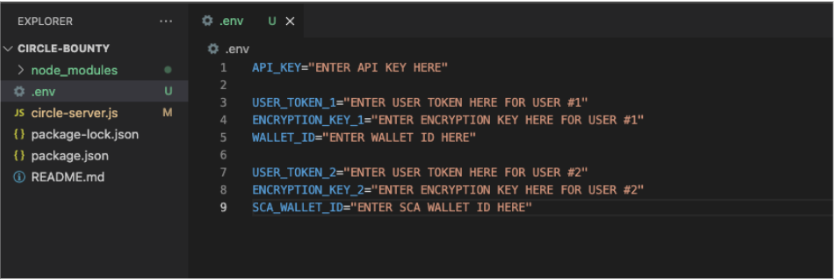
USER\_TOKEN\_2="ENTER USER TOKEN HERE FOR USER #2"

ENCRYPTION\_KEY\_2="ENTER ENCRYPTION KEY HERE FOR USER #2"

SCA\_WALLET\_ID="ENTER SCA WALLET ID HERE"

JavaScript

After creating the **.env** file, here’s how your working directory should look like!

Preview of Working Directory

Next, head over to your [**Circle Developer Console Account**](https://console.circle.com/) and under the **“API Keys”** section, proceed to port over your API key value to the **API\_KEY** environment variable in the **.env** file!

ℹ️ **Important**: If you've misplaced your API key or haven't set one up previously, you can create a new API key by selecting “Create a Key”.

Now that we have filled in the **API\_KEY** environment variable, let’s move on to the next section.

**Understanding the Code**

Navigate to the **circle-server.js** file, where you'll find that we've imported the **@circle-fin/user-controlled-wallets** library.

This npm library allows us to set up a client that allows us to interact with the suite of Programmable Wallets API, facilitating operations such as user creation, session token generation, and more!

Additionally, the **yargs** and **dotenv** libraries are included in our setup.

**yargs** enables the development of interactive command-line interfaces, facilitating a range of operations such as user creation, session token generation, and challenge creation.

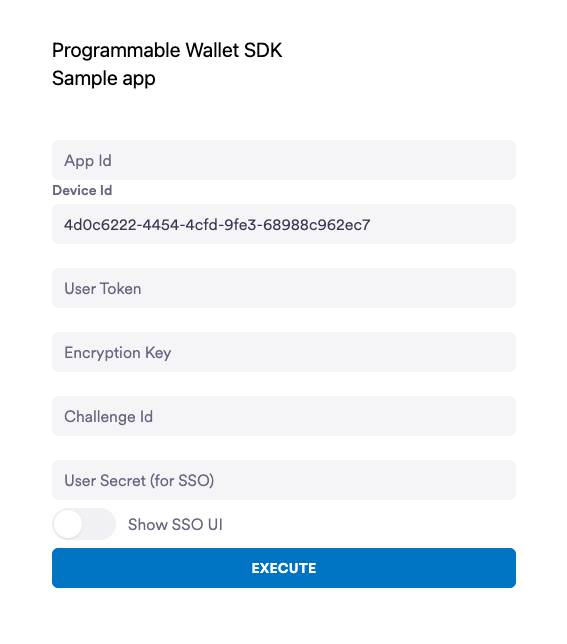
**dotenv**, meanwhile, is essential for securely and efficiently loading environment variables, like our API key, from the .env file. This ensures that our configurations are easily accessible yet secure.

**Run the Web Application**

As mentioned earlier, we will also be interacting with Circle’s sample web application built with Circle’s recently released Web SDK designed to allow developers to easily integrate the Web SDK into applications!

In this quest, we will be using the sample web application to initialize our wallets and execute the necessary challenges!

Proceed to head over [**here**](https://pw-auth-example.circle.com/) to launch the sample web application and you should observe the following as shown below.



ℹ️ **Note**: You can leave the web application running in the background as we will be referencing it in the upcoming steps!

⚠️ **Important:**When filling in the fields in the upcoming steps, you can leave the **User Secret (for SSO)** field empty and toggle off the **'Show SSO UI'**option.

With that, we are all set up!

In the next step, we will be using the Circle Server SDK to create 2 users and generate the necessary session tokens!

* **Step 2:Create Users & Generate Session Token**

In this step, we will interact with Circle’s Server SDK to create 2 users and generate the necessary session tokens!

**Create Users**

To begin, head over to your **circle-server.js** file and head over to the **createNewUser()** function below the comment **// #Step 1 - Create a new user** where we will be creating the first user!

Under the **userId** variable, proceed to replace the placeholder “ENTER USERNAME HERE” with “User\_1” and hit save!

Here’s how your function should look!

async function createNewUser() {

let response = await client.createUser({

userId: "User\_1",

});

console.log(response);

}

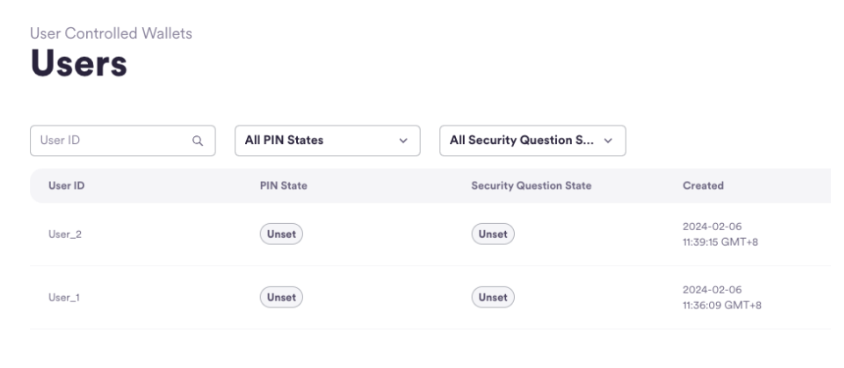
JavaScript

Next, proceed to run the following command on your terminal.

node circle-server.js create-user

Next, proceed to replace the **userId** variable with “User\_2” and run the above command again!

To verify if our 2 users have indeed been created, proceed to head over to your [**Circle Developers Console Users**](https://console.circle.com/wallets/user/users) tab and you should observe that your 2 Users have indeed been successfully created as shown below!

Users Created Successfully!

With our users successfully created, up next we will create the necessary session tokens to authenticate our subsequent requests to create a wallet, fetch wallet balance and initiate an outbound transfer!

**Create Session Tokens**

To begin, head over to the **createSessionToken()** function below the comment **// #Step 2 - Create session token** and replace the **userId** placeholder field with “User\_1”.

Here’s how your function should look!

async function createSessionToken() {

let response = await client.createUserToken({

userId: "User\_1",

});

console.log(response.data);

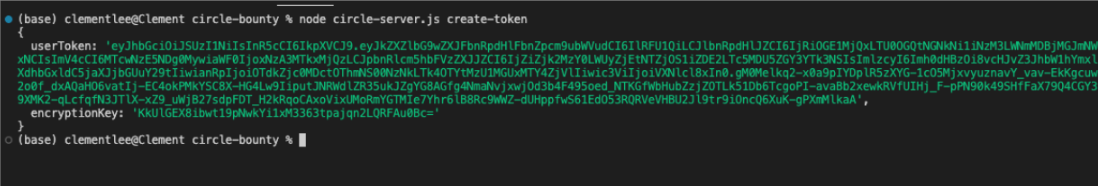
}

JavaScript

Next, proceed to run the following command on your terminal.

node circle-server.js create-token

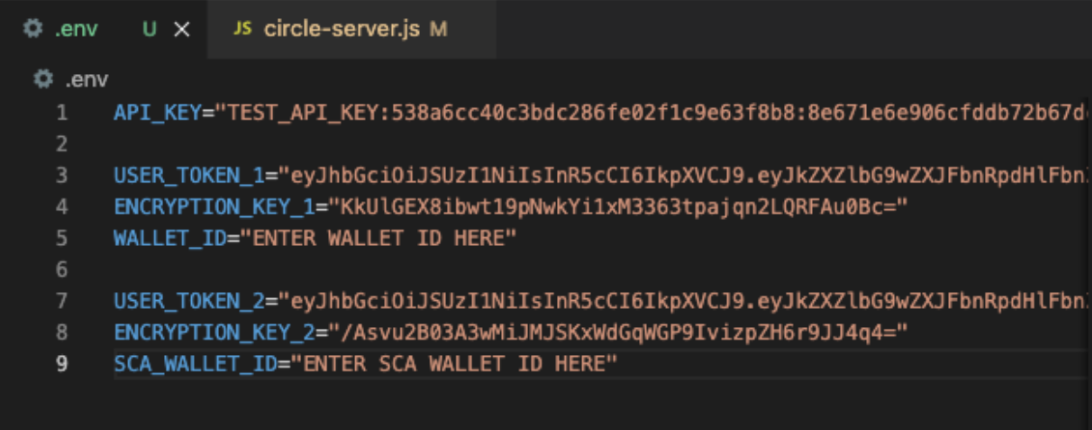
After running the command, you should observe that the userToken and encryption key values have been successfully generated as shown below!

Preview of "Create Session Token" Command

ℹ️ **Important**: Proceed to copy your userToken and encryption key values to the **USER\_TOKEN\_1**and **ENCRYPTION\_KEY\_1** environment variables in the .env file.

Once you have done so, proceed to repeat the same steps by replacing the **userId** variable with “User\_2” and generate a new session token and fill in the environment variables for **USER\_TOKEN\_2** and **ENCRYPTION\_KEY\_2**.

Here’s how your **.env** file should look like!

Preview of .env file

🎉 How easy was that! With Circle’ Server SDK, interacting with the suite of Programmable Wallets API suite is much more seamless and intuitive!

In the next step, we will use our generated session tokens to create the respective wallets for Users 1 and 2!

* **Step 3:Create a User-Controlled EOA and SCA Wallet**

In this step, we're going to set up two wallets.

User 1 will establish a default External Owned Account (EOA) User-Controlled Wallet, while User 2 will create a Smart Contract Account (SCA) User-Controlled Wallet. Both wallets will be initialized on the Ethereum Sepolia test network.

**Create EOA Wallet**

To begin, head over to the **createChallengeForWalletCreation()**function below the comment**// #Step 3 - Create Challenge for Wallet Creation** and replace the **userId** field with “User\_1”.

Here’s how your function should look!

async function createChallengeForWalletCreation() {

let response = await client.createUserPinWithWallets({

userId: "User\_1",

blockchains: ["ETH-SEPOLIA"],

userToken: process.env.USER\_TOKEN\_1,

});

console.log(response.data?.challengeId);

}

JavaScript

Next, proceed to run the following command on your terminal to generate a challenge ID to create an EOA wallet.

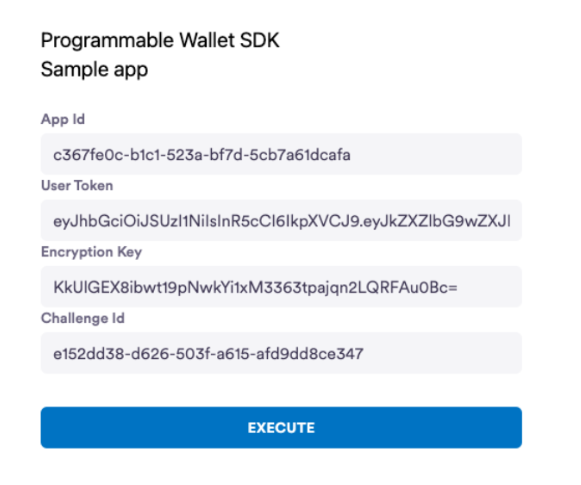
node circle-server.js create-challenge-wallet

After running the following command, you should observe that a challenge ID has been successfully generated as shown below! Proceed to save your challenge ID as we will be needing it to initialize the wallet.

**Initialize EOA Wallet**

We are now ready to initialize the EOA wallet! Proceed to head over to the [**web application**](https://pw-auth-example.circle.com/)and fill in all fields.

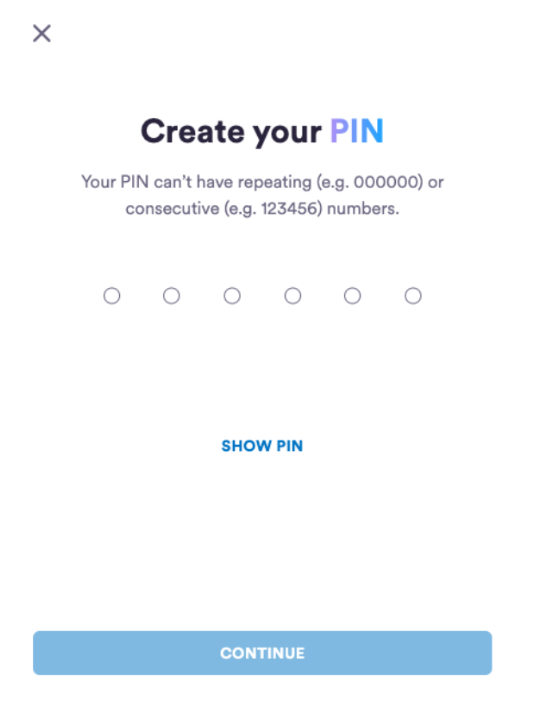
Your App ID can be found under the [**Circle Developers Console Configurator tab**](https://console.circle.com/wallets/user/configurator). Here’s how your fields should look!



🔔 **Reminder**: If your session token has expired, you are required to create a new session token by running the earlier commands and replace the userToken and encryptionKey values respectively on your **.env** file!

Next, click on “Execute”.

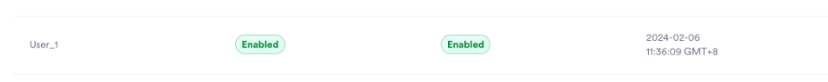
You should now observe the following as shown below.

Create your PIN

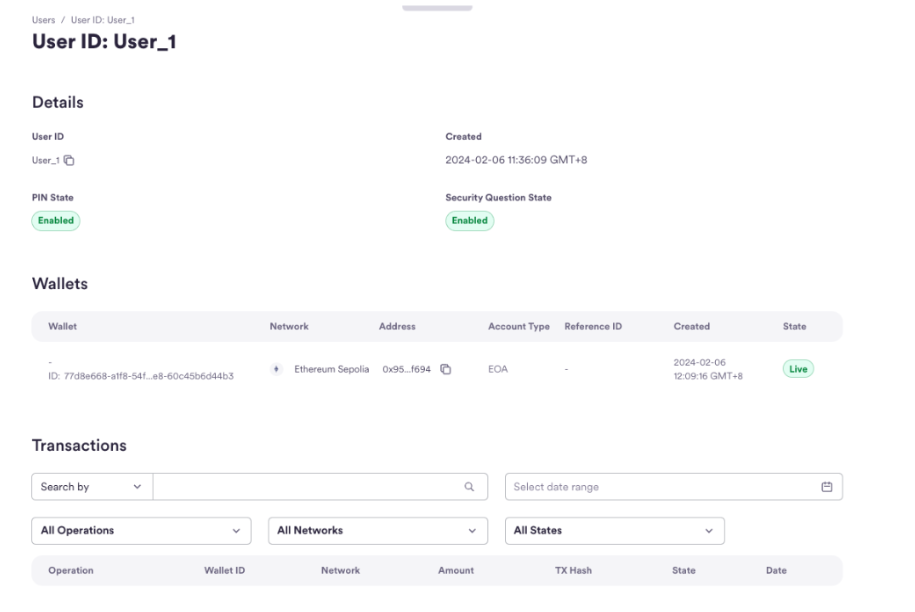
Proceed to create your PIN and re-enter your PIN to verify!

Next, proceed to follow the prompts to set up the recovery methods for your wallet! Lastly, type “I agree” under the confirmation page and click on “Continue”.

To verify if your wallet has indeed been successfully created, head over to the [**Circle Developer Console Users Tab**](https://console.circle.com/wallets/user/users)and you should observe that “User\_1” PIN State and Security Question State has been enabled as shown below.



Next, click on “User\_1” and observe that your wallet has been successfully initialized on the Ethereum Sepolia network as shown below!

User-Controlled Wallet Successfully Created on Ethereum Sepolia

With that, we have successfully created an EOA User-Controlled wallet for User\_1! Proceed to paste the “Wallet ID” value to the **WALLET\_ID** environment variable in the .env file.

Next, we will be creating a SCA User-Controlled wallet for User\_2!

**Create SCA Wallet**

To begin, head over to the **createChallengeForSCAWalletCreation()** function below the comment **// #Step 4 - Create Challenge for SCA Wallet Creation** and replace the **userId** field with “User\_2”.

Here’s how your function should look!

async function createChallengeForSCAWalletCreation() {

let response = await client.createUserPinWithWallets({

userId: "User\_2",

blockchains: ["ETH-SEPOLIA"],

accountType: "SCA",

userToken: process.env.USER\_TOKEN\_2,

});

console.log(response.data?.challengeId);

}

JavaScript

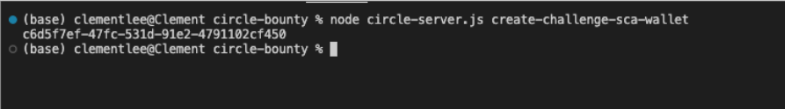
You should observe that we have defined the **“SCA”** account type to represent that we will be creating a SCA wallet!

⚠️ **Reminder**: If your session token has expired, you will need to create a new session token by running the earlier commands and replace the userToken and encryptionKey values on your **.env** file!

Proceed to run the following command on your terminal.

node circle-server.js create-challenge-sca-wallet

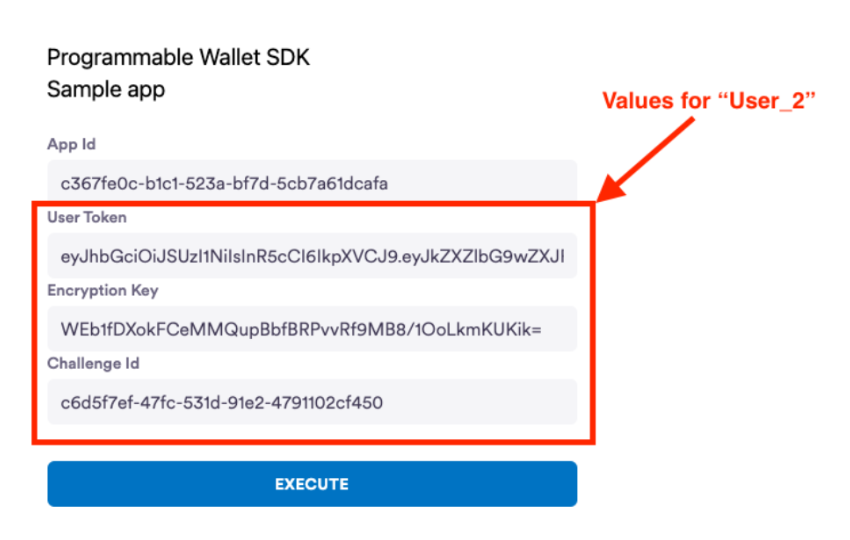
You should observe that a challenge ID is successfully generated on your terminal as shown below.

Challenge ID Successfully Generated!

**Initialize SCA Wallet**

We are now ready to initialize the SCA wallet! Proceed to head over to the web application and fill in all fields!

Here’s how your field should look like as shown below.

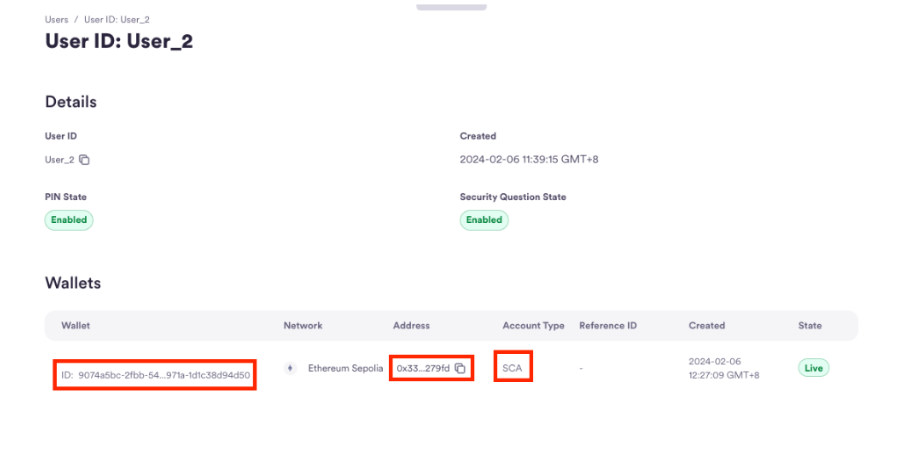


Next, click on “Execute”.

Similar to how you have created a PIN and set up your recovery methods, proceed to follow the on-screen prompts to initialize your SCA wallet!

Proceed to head over to the [**Developer Console Users Tab**](https://console.circle.com/wallets/user/users) and you should observe that “User\_2” PIN and Security Questions State has been enabled!

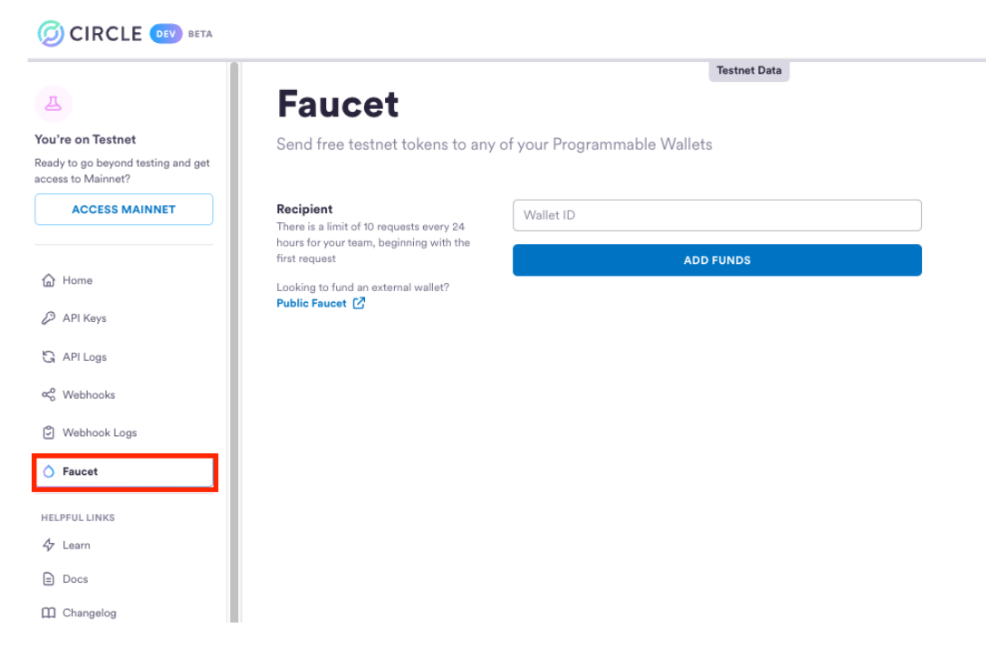
Proceed to click on “User\_2” and you should observe that your SCA wallet has been successfully created as shown below!

SCA Wallet Successfully Created on Ethereum Sepolia

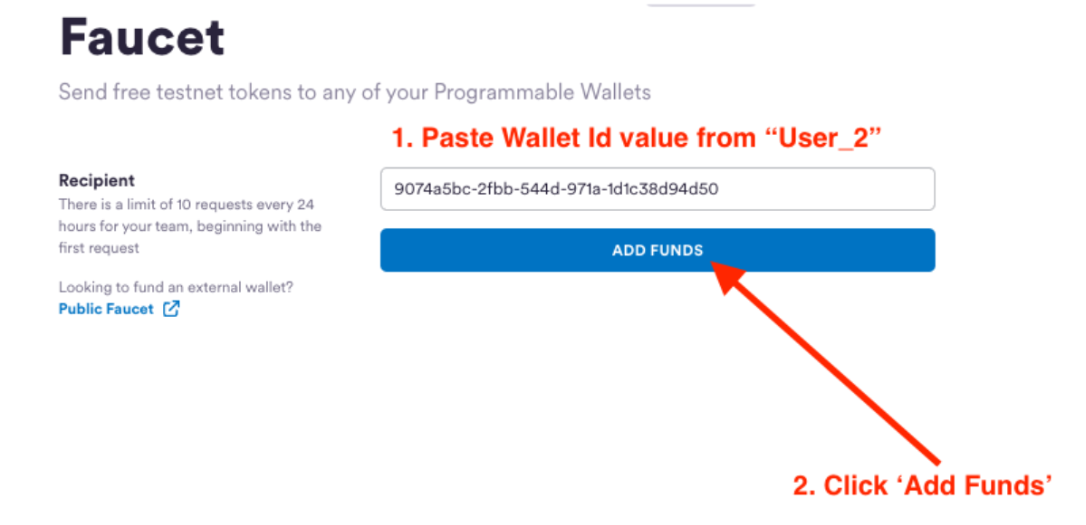
**Fund SCA Wallet with Faucet**

With our SCA wallet successfully created, up next, we will fund our SCA wallet with USDC from Circle’s developer console faucet!

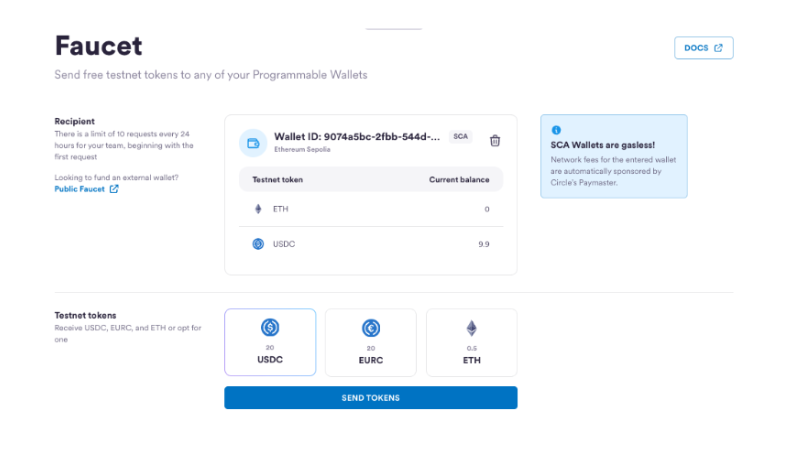
In order to do so, head over to the ‘Faucet’ tab within the developer console as shown below.



Next, paste the wallet id value from User\_2 and click on ‘Add Funds.



Next, do ensure that the ‘USDC’ option is selected and click on ‘Send Tokens’.



Once the transaction is complete, you should observe that the USDC tokens have been sent successfully to your SCA wallet!

Proceed to paste the “Wallet ID” value of your SCA wallet to the **SCA\_WALLET\_ID** environment variable in the **.env**file.

With that, we’re all set up to initiate an outbound transfer in the next step!

* **Step 4:Initiate an Outbound Transfer**

In this step, we will be initiating an outbound transfer of USDC from your SCA wallet to the EOA wallet!

Before we can initiate an outbound transfer, we will first need to fetch our wallet balance to get the token ID of USDC as it is a required parameter for the transfer!

**Fetch USDC Token ID**

To begin, head over to the **fetchWallet()** function below the comment **// #Step 5 - Fetch Wallet Balance** and replace the **userId** field with “User\_2”.

Here’s how your function should look!

async function fetchWallet() {

let response = await client.getWalletTokenBalance({

walletId: process.env.SCA\_WALLET\_ID,

userToken: process.env.USER\_TOKEN\_2,

userId: "User\_2",

});

console.log(response.data?.tokenBalances);

}

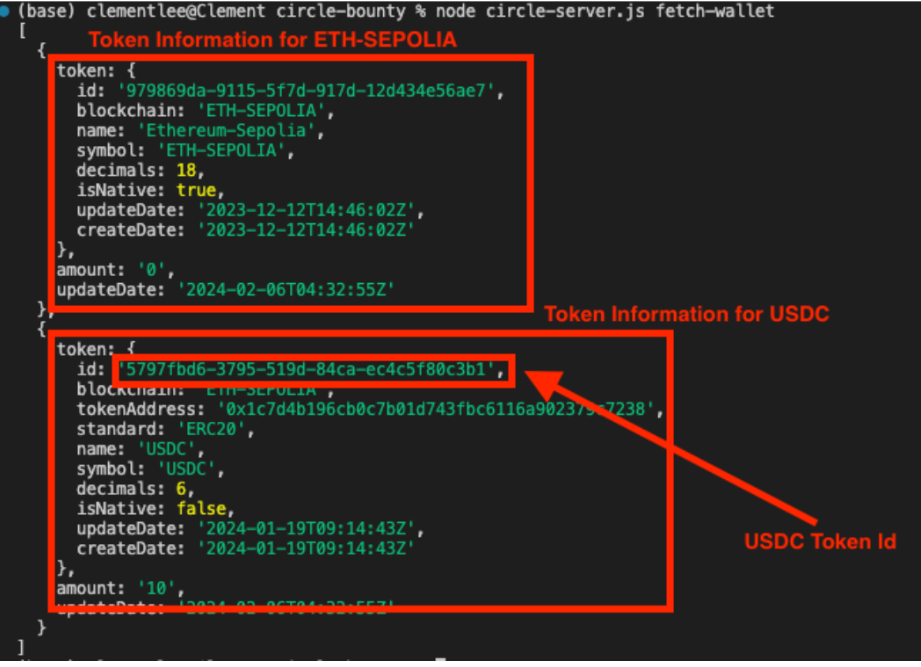
JavaScript

⚠️ **Reminder**: If your session token has expired, you will need to create a new session token by running the earlier commands and replace the userToken and encryptionKey values on your .env file!

Next, proceed to run the following command on your terminal.

node circle-server.js fetch-wallet

You should observe the following output on your terminal!

Token Balances of SCA Wallet for User\_2

Proceed to copy the USDC Token Id value as we will be referencing it in the next section!

**Initiate Outbound Transfer**

We are now ready to initiate an outbound transfer from our SCA wallet to the EOA wallet!

To begin, head over to the **createChallengeForOutboundTransfer()** function below the comment **// #Step 6 - Create Challenge for Outbound Transfer** and fill in the necessary fields of the function!

⚠️ **Important**: You are to generate a UUID value for the “**idempotencyKey**” field over [**here**](https://www.uuidgenerator.net/), replace the “**destinationAddress**” field with your “User\_1” EOA wallet address and enter “User\_2” under the “**userId**” field

Here’s how your function should look!

async function createChallengeForOutboundTransfer() {

let response = await client.createTransaction({

idempotencyKey: "c00c3c3d-5698-4280-b17c-0afaf8d63972",

amounts: ["0.1"],

destinationAddress: "0x9571e2bf50ed0ae1f514c15dc402c7096013f694",

tokenId: "5797fbd6-3795-519d-84ca-ec4c5f80c3b1",

walletId: process.env.SCA\_WALLET\_ID,

userId: "User\_2",

fee: {

type: "level",

config: {

feeLevel: "MEDIUM",

},

},

userToken: process.env.USER\_TOKEN\_2,

});

console.log(response.data?.challengeId);

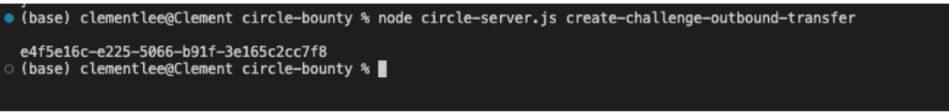
}

JavaScript

Next, proceed to run the following command on your terminal to generate a challenge ID for the outbound transfer transaction.

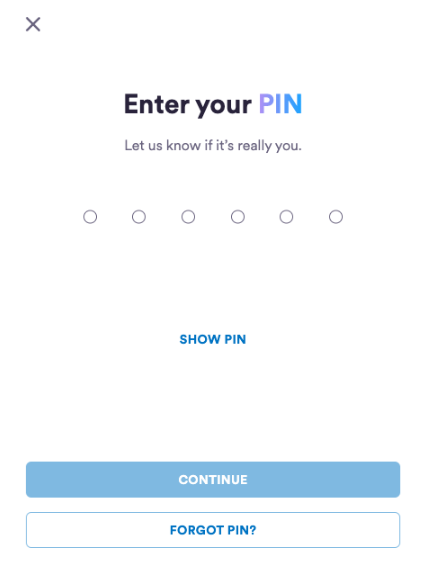
node circle-server.js create-challenge-outbound-transfer

You should observe that after running the command, the challenge ID is successfully generated as shown below.

Challenge ID Successfully Generated!

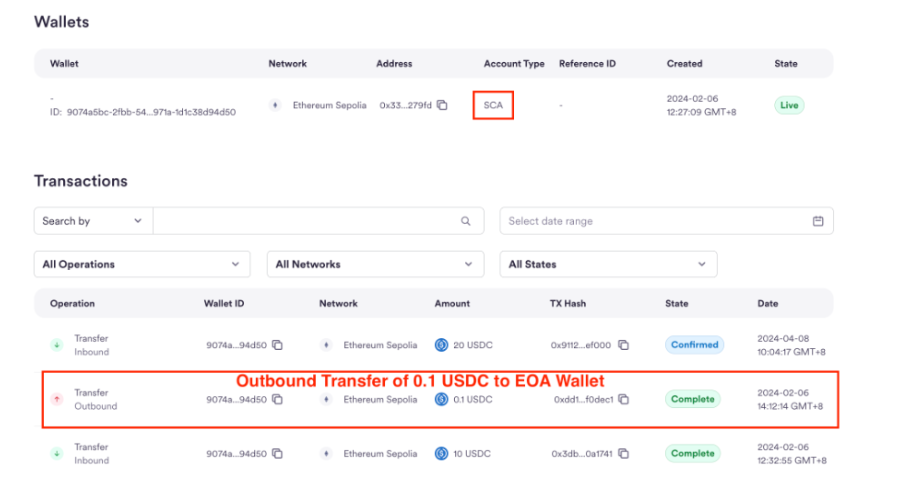
Next, proceed to port over the relevant fields to the web application to execute the challenge for the outbound transfer transaction and click “Execute”.

You should observe the following on your web application as shown below where you are required to enter the PIN during wallet initialization of your SCA wallet to execute the outbound transfer transaction!

Enter your PIN

Proceed to enter your PIN and click “Continue”.

To verify if your outbound transfer transaction was indeed successful, head over to your developer console account and under “User\_2”’s wallet balance, you should observe the following!



Congratulations Learners! 🎉🎉 You've just successfully executed an outbound transfer from a Smart Contract Account (SCA) wallet to an Externally Owned Account (EOA) wallet.

This is a significant achievement in enhancing user experience, as it demonstrates the potential and functionality of SCA wallets where creating these wallets on behalf of users allow these users to seamlessly interact with blockchain applications without needing to possess native tokens (like ETH) to cover transaction fees!

Please capture a screenshot of your wallet details page, as illustrated earlier. This screenshot will serve as the required deliverable for this quest.

* **Step 5:Let’s Ace Your Submissions! Preparing Your Submission!**

You have reached the end! Now to make sure you successfully complete this quest! There is 1 deliverable that is required for this quest, a screenshot.

**Screenshot**

In Step 4, you should have taken a screenshot of your SCA wallet details page!

Your screenshot should show:

* + your full screen, including your taskbar (for Windows and Linux) / dock (for MacOS)
  + Wallet Account Type shows SCA
  + A successful outbound transfer of 0.1 USDC to the EOA wallet
  + make sure that all parts visible in ‘Expected output’ are also visible in your screenshot!

When labeling your screenshot, make sure to follow the format provided (remember to replace “yourstackupname” with your StackUp username): C20\_Q2\_yourstackupname.png.

**Note**: You can retrieve your Campus Learn and Earn v2.0 username by logging into your StackUp Community account and clicking "Login to Campus". Your username can be found at the top of the profile icon or by accessing Account Information under Account Management. You can check out [**this article**](https://stackuphelpcentre.zendesk.com/hc/en-us/articles/25416574073625-How-to-obtain-your-Campus-Learn-and-Earn-v2-0-username-) for a reference on how to obtain it!

Refer to ‘Expected output’ if you are unsure.

By submitting the quest, please note that our StackUp Policy prohibits the use of multiple accounts by a single user and the submission of copied work.

**Expected Output**

