Quickstart - Browser

This page will introduce the primary operations provided by Request Network's SDK while using the `Web3SignatureProvider` to sign requests with a private key stored inside a wallet.

This approach works well for Browser environments *with* access to a web3 wallet.

You will learn:

- How to create a request
- How to update a request (coming soon...)
- How to pay a request
- How to detect a payment
- How to retrieve a user's requests

Create a request

To create an unencrypted ERC-20 request, first connect to an `ethers` v5 `Provider` and `Signer` or `wagmi` / `viem` `WalletClient.`

Unfortunately, the Request Network SDK does not yet support ethers v6.

ethers v5 | wagmi | viem

```
import { providers } from "ethers";

let provider;
if (process.env.WEB3_PROVIDER_URL === undefined) {
  // Connect to Metamask and other injected wallets
  provider = new providers.Web3Provider(window.ethereum);
} else {
  // Connect to your own Ethereum node or 3rd party node provider
  provider = new providers.JsonRpcProvider(process.env.WEB3_PROVIDER_URL);
}
// getDefaultProvider() won't work because it doesn't include a Signer.
```

Then, construct a `Web3SignatureProvider`, passing in the `ethers` `Provider` or `viem` `WalletClient.`

```
import { Web3SignatureProvider } from "@requestnetwork/web3-signature";

const web3SignatureProvider = new Web3SignatureProvider(provider);
```

Then, construct a `RequestNetwork`, passing in the:

- Request Node URL. In this example, we use the Sepolia Request Node Gateway.
- `Web3SignatureProvider` constructed in the previous step.

```
import { RequestNetwork } from "@requestnetwork/request-client.js"

const requestClient = new RequestNetwork({
  nodeConnectionConfig: {
    baseURL: "https://sepolia.gateway.request.network/",
  },
  signatureProvider: web3SignatureProvider,
});
```

Then, prepare the Request creation parameters:

```
import { Types, Utils } from "@requestnetwork/request-client.js";

const payeeIdentity = '0x7eB023BFbAeE228de6DC5B92D0BeEB1eDb1Fd567';
const payerIdentity = '0x519145B771a6e450461af89980e5C17Ff6Fd8A92';
const paymentRecipient = payeeIdentity;
const feeRecipient = '0x0000000000000000000000000000000000000000';

const requestCreateParameters = {
  requestInfo: {

    // The currency in which the request is denominated
    currency: {
      type: Types.RequestLogic.CURRENCY.ERC20,
      value: '0x370DE27fdb7D1Ff1e1BaA7D11c5820a324Cf623C',
      network: 'sepolia',
    },

    // The expected amount as a string, in parsed units, respecting `decimals`
    // Consider using `parseUnits()` from ethers or viem
    expectedAmount: '1000000000000000000',

    // The payee identity. Not necessarily the same as the payment recipient.
    payee: {
      type: Types.Identity.TYPE.ETHEREUM_ADDRESS,
      value: payeeIdentity,
    },

    // The payer identity. If omitted, any identity can pay the request.
    payer: {
      type: Types.Identity.TYPE.ETHEREUM_ADDRESS,
      value: payerIdentity,
    },

    // The request creation timestamp.
    timestamp: Utils.getCurrentTimestampInSecond(),
  },

  // The paymentNetwork is the method of payment and related details.
  paymentNetwork: {
    id: Types.Extension.PAYMENT_NETWORK_ID.ERC20_FEE_PROXY_CONTRACT,
    parameters: {
      paymentNetworkName: 'sepolia',
      paymentAddress: payeeIdentity,
      feeAddress: feeRecipient,
      feeAmount: '0',
    },
  },

  // The contentData can contain anything.
  // Consider using rnf_invoice format from @requestnetwork/data-format
  contentData: {
    reason: '🍕',
```

```
    dueDate: '2023.06.16',
  },

  // The identity that signs the request, either payee or payer identity.
  signer: {
    type: Types.Identity.TYPE.ETHEREUM_ADDRESS,
    value: payeeIdentity,
  },
};
```

Then, call `createRequest()` to prepare a `Request` object.

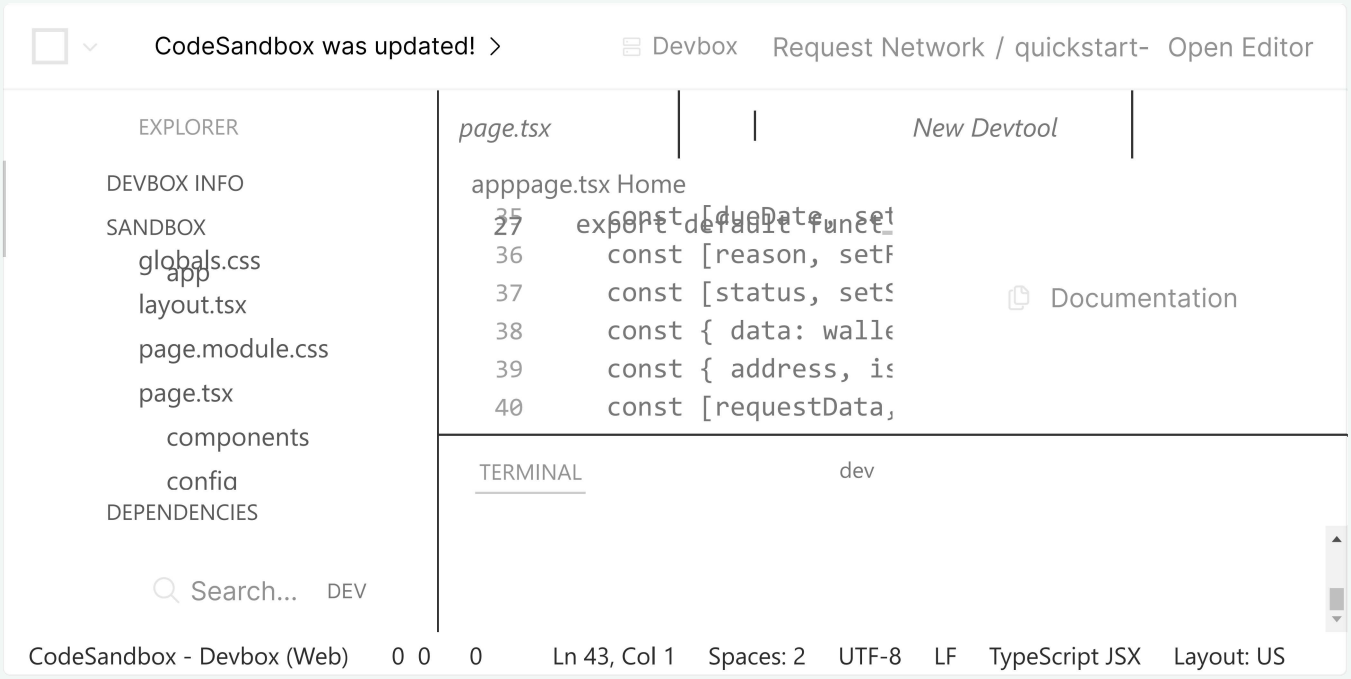
```
const request = await requestClient.createRequest(requestCreateParameters);
```

Finally, call `request.waitForConfirmation()` to wait until:

- The request contents are persisted in IPFS
- The Content-addressable ID (CID) is stored on-chain
- The resulting on-chain event is indexed by the storage subgraph.

```
const confirmedRequestData = await request.waitForConfirmation();
```

CodeSandBox: Create a request



https://codesandbox.io/p/sandbox/create-a-request-shffng?file=/app/page.tsx:43,1

Pay a request

First, construct a `RequestNetwork` object and connect it to a Request Node. In this example, we use the Sepolia Request Node Gateway:

> Note that paying a request doesn't require a `SignatureProvider` be passed into the `RequestNetwork` object.

```
import { RequestNetwork, Types } from "@requestnetwork/request-client.js";

const requestClient = new RequestNetwork({
  nodeConnectionConfig: {
    baseURL: "https://sepolia.gateway.request.network/",
  }
});
```

Then, retrieve the request and get the request data. Take note of the current request balance, to be used later for payment detection.

```
const request = await requestClient.fromRequestId(
  '019830e9ec0439e53ec41fc627fd1d0293ec4bc61c2a647673ec5aaaa0e6338855',
);
const requestData = request.getData();
```


> Unfortunately, the Request Network SDK does not yet support ethers v6.

ethers v5 | wagmi | viem

```
import { providers } from "ethers";

let provider;
if (process.env.WEB3_PROVIDER_URL === undefined) {
  // Connect to Metamask and other injected wallets
  provider = new providers.Web3Provider(window.ethereum);
} else {
  // Connect to your own Ethereum node or 3rd party node provider
  provider = new providers.JsonRpcProvider(process.env.WEB3_PROVIDER_URL);
}
// getDefaultProvider() won't work because it doesn't include a Signer.

const signer = await provider.getSigner();
```

Then, check that the payer has sufficient funds using `hasSufficientFunds()`

```
import { hasSufficientFunds } from "@requestnetwork/payment-processor";

const _hasSufficientFunds = await hasSufficientFunds(
  requestData,
  payerAddress,
  {
    provider: provider,
  },
);
```

Then, in the case of an ERC-20 request, check that the payer has granted sufficient approval using `hasErc20Approval()`. If not, submit an approval transaction using `approveErc20`. Wait for an appropriate number of block confirmations. On Sepolia or Ethereum, 2 block confirmations should suffice. Other chains may require more.

```
import { approveErc20, hasErc20Approval } from "@requestnetwork/payment-proces

const _hasErc20Approval = await hasErc20Approval(
  requestData,
  payerAddress,
  provider
);
if (!_hasErc20Approval) {
  const approvalTx = await approveErc20(requestData, signer);
  await approvalTx.wait(2);
}
```

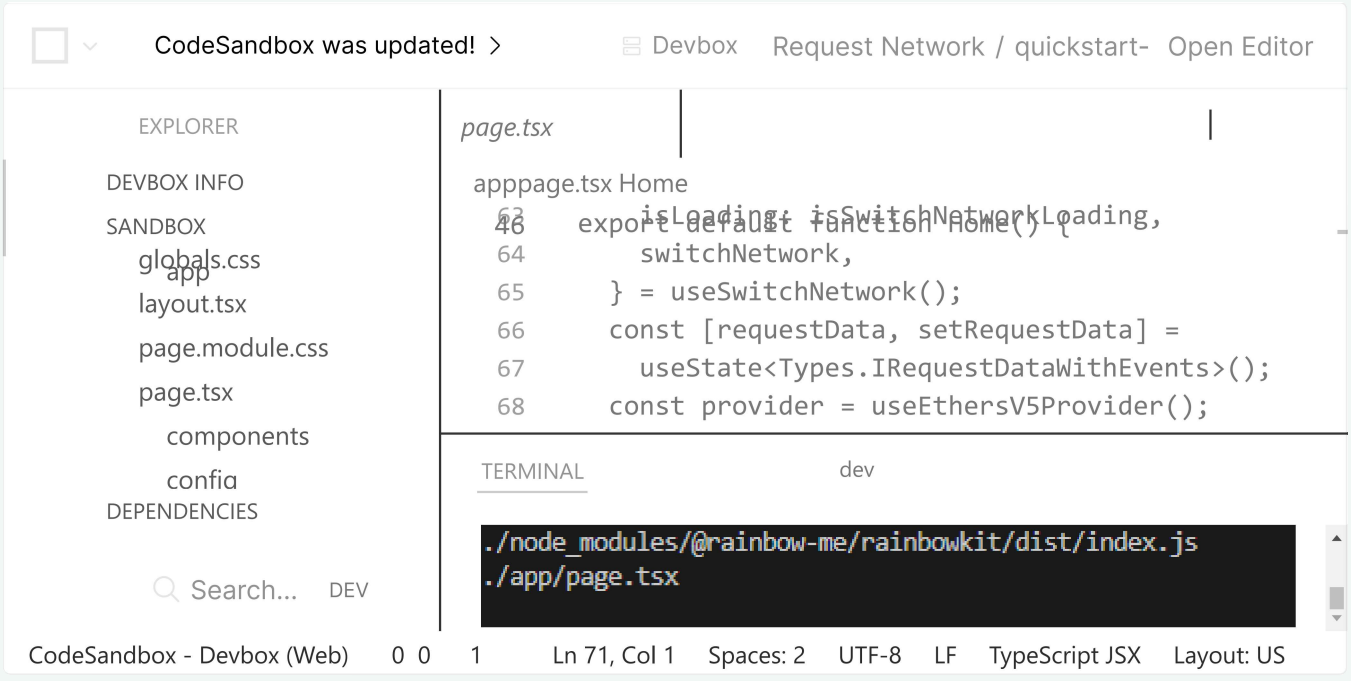
Finally, pay the request using `payRequest()`

```
import { payRequest } from "@requestnetwork/payment-processor";

const paymentTx = await payRequest(requestData, signer);
await paymentTx.wait(2);
```

You can detect that the payment was successful by polling the request and waiting until the request balance is greater than or equal to the expected amount.

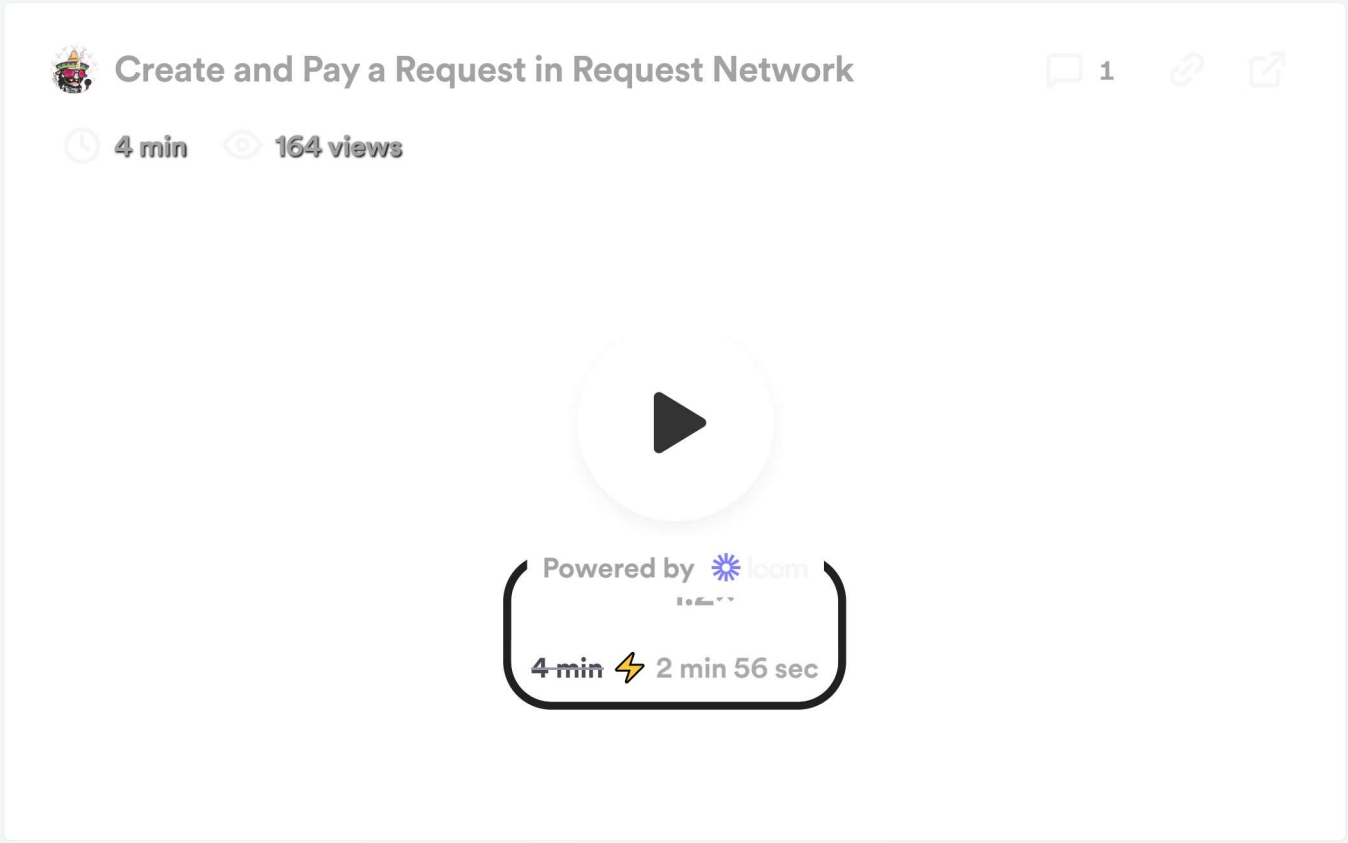
```
const request = await requestClient.fromRequestId(requestData.requestId);
let requestData = request.getData();

while (requestData.balance?.balance < requestData.expectedAmount) {
  requestData = await request.refresh();
  await new Promise((resolve) => setTimeout(resolve, 1000));
}
```

CodeSandBox: Create and pay a request and detect a payment



https://codesandbox.io/p/sandbox/pay-a-request-dn7kcf?file=/app/page.tsx:71,1

Video: Create and pay a request and detect a payment



Retrieve a user's requests

First, construct a `RequestNetwork` object and connect it to a Request Node. In this example, we use the Sepolia Request Node Gateway:

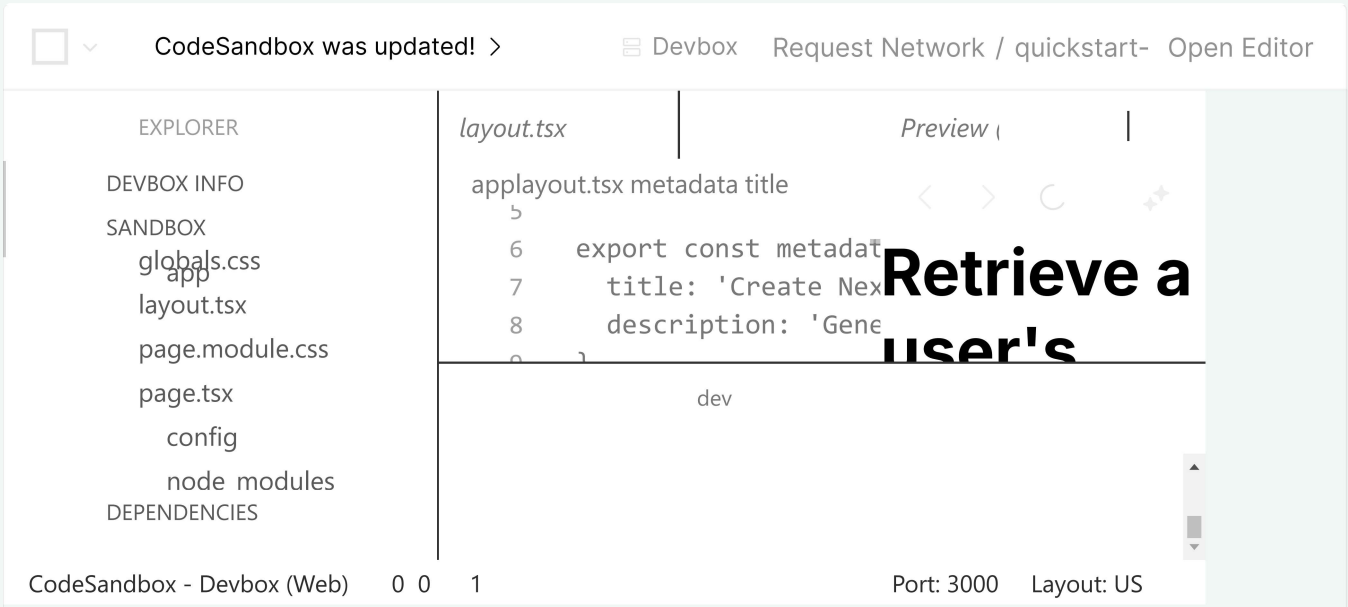
```
import { RequestNetwork, Types } from "@requestnetwork/request-client.js";
const requestClient = new RequestNetwork({
  nodeConnectionConfig: {
    baseURL: "https://sepolia.gateway.request.network/",
  },
});
```

Then, call `fromIdentity()` to get an array of `Request` objects or `fromRequestId()` to get a single `Request` object. This function retrieves the `Request`s stored in IPFS and queries on-chain events to determine the balances paid so far. Finally, call `getData()` on each `Request` to get the request contents.

fromIdentity() | fromRequestId()

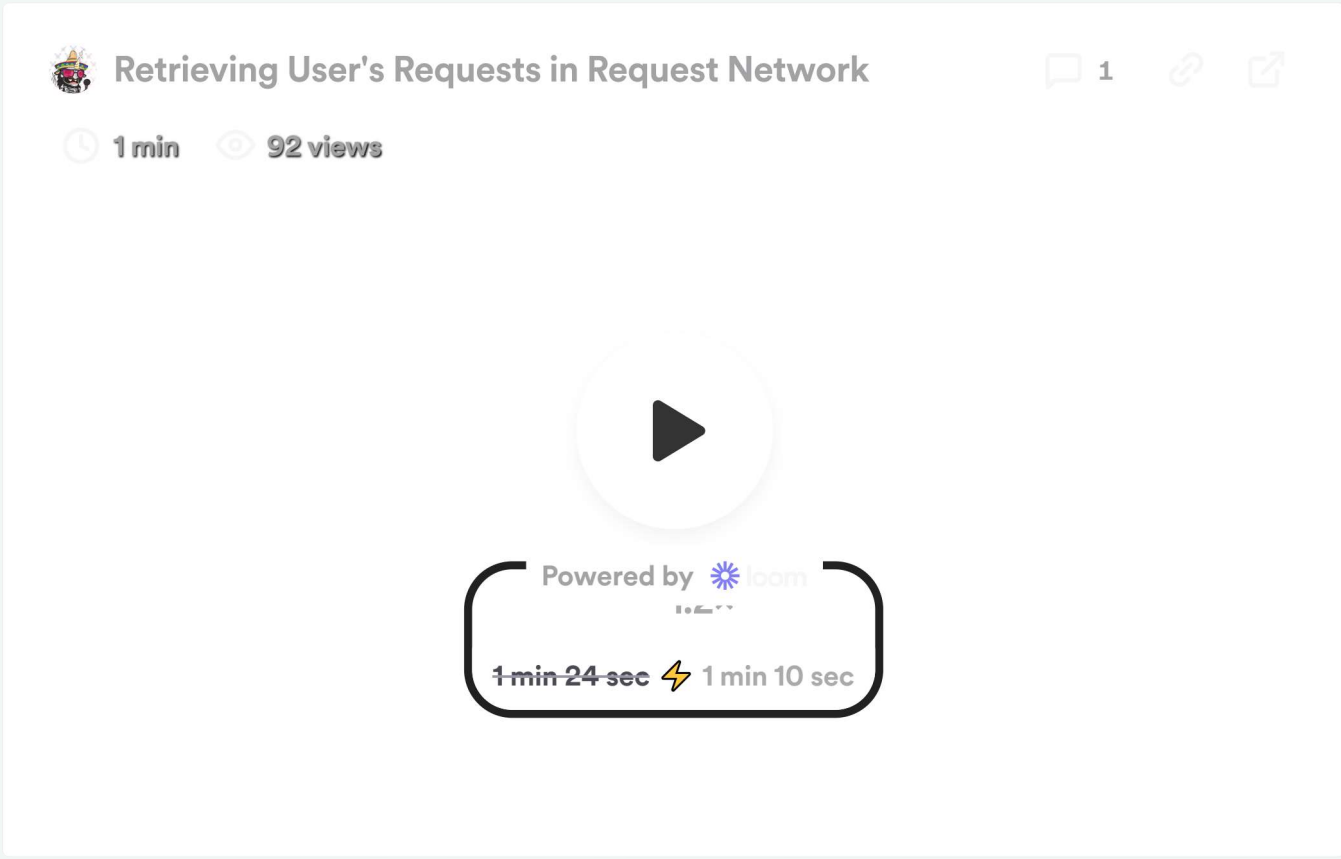
```
const identityAddress = "0x519145B771a6e450461af89980e5C17Ff6Fd8A92";
const requests = await requestClient.fromIdentity({
  type: Types.Identity.TYPE.ETHEREUM_ADDRESS,
  value: identityAddress,
});
const requestDatas = requests.map((request) => request.getData());
```

CodeSandBox: Retrieve a user's requests



https://codesandbox.io/p/sandbox/retrieve-a-users-requests-mqrjqy?file=/app/page.tsx:10,1

Video: Retrieve a user's requests



Previous
Lifecycle of a Request

Next
Quickstart - Node.js

Last updated 8 months ago