0x9fE46736679d2D9a65F0992F2272dE9f3c7fa6e0

BACKEND
----
configure backend folder
yarn add hardhat --dev
yarn hardhat init
create the SC in /contract
rename the file in /ignition/modules
update infos inside this file (Bank.js)
``
const { buildModule } = require("@nomicfoundation/hardhat-ignition/
modules");

module.exports = buildModule("BankModule", (m) => {
  const bank = m.contract("Bank");

  return { bank };
});
``
run the local node
yarn hardhat node
deploy with ignition
yarn hardhat ignition deploy ./ignition/modules/Bank.js --network
localhost
alternative au deploiement
creer folder scripts et file Bank.s.js
``
const hre = require("hardhat");

async function main() {

  const Bank = await hre.ethers.deployContract("Bank");

  await Bank.waitForDeployment();

  console.log(
    `Bank deployed to ${Bank.target}`
  );
}

// We recommend this pattern to be able to use async/await
everywhere
// and properly handle errors.
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
``
deployer avec cette methode
yarn hardhat run ./scripts/Bank.s.js --network localhost

FRONTEND

```
cd frontend
# create NEXT app
npx create-next-app@latest .
(no no yes no yes no) (tailwind & app router)
# install rainbowkit (https://www.rainbowkit.com/docs/installation)
npm install @rainbow-me/rainbowkit wagmi viem@2.x @tanstack/react-
query
# install chadcn-UI (https://ui.shadcn.com/)
npx shadcn-ui@latest init
(default, slate, yes)
# create a file in the /frontend/app folder named
RainbowKitAndWagmiProvider.js
# in this new file -> rafce (for generate a default template)
# 'use client' at the beginning of the file
# import list of imports from rainbowkit (https://
www.rainbowkit.com/docs/installation#import)
# change the data networks and replace with hardhat, sepolia (remove
mainnet,polygon,optimism,arbitrum,base,)
# copy and past bellow the configuration (https://
www.rainbowkit.com/docs/installation#configure)
# and update the key from wallet connect (https://
cloud.walletconnect.com/)
# projectId: 'YOUR_PROJECT_ID',
# update chains (cf previous imports)
# chains: [hardhat, sepolia],
# surround {children} with the different elements specified in the
documentation
```
```
const queryClient = new QueryClient();

const RainbowKitAndWagmiProvider = ({ children }) => {
  return (
    <WagmiProvider config={config}>
      <QueryClientProvider client={queryClient}>
        <RainbowKitProvider>{children}</RainbowKitProvider>
      </QueryClientProvider>
    </WagmiProvider>
  );
};

# Update the file /app/Layout.js (https://ui.shadcn.com/docs/
installation/)
# 4 <-> (Import the font in the root layout)
# /app/layout.js :
# copy/paste the code source then
(remove ``: RootLayoutProps``if no TypeScript)
(change ``import "@/styles/globals.css"``by "import ./globals.css")
# import RainbowKitAndWagmiProvider from ...
# add and change the metadata for the SEO
``
export const metadata = {
  title: "Bank DApp",
  description: "Description...",
};
```

```
#surround my app with this provider
#replace the ... in the file just upon the </body>
```
```
<RainbowKitAndWagmiProvider>{children}</RainbowKitAndWagmiProvider>
```
```
# launch the localhost network for check
npm run dev
# dark mode of the background
# add dark inside "min-h-screen bg-background DARK font-sans
antialiased",
# Change the dark theme of the button
# RainbowKitAndWagmiProvider.js :
# add darkTheme inside the imports
import { getDefaultConfig, RainbowKitProvider, darkTheme } from
"@rainbow-me/rainbowkit";
# add the style to the component
<RainbowKitProvider theme={darkTheme()}>
# add jsx components header, footer and layout (./components)
# create a folder 'components' if it doesnt exists
cd ./components
# create the files
touch Header.jsx Footer.jsx Layout.jsx
# Header.jsx :
(rafce)
```
```
import { ConnectButton } from "@rainbow-me/rainbowkit";
const Header = () => {
  return (
    <nav className="navbar flex justify-between items-center p-5 w-
full">
      <div className="grow">Logo</div>
      <div>
        <ConnectButton />
      </div>
    </nav>
  );
};

export default Header;
```
```
# Footer.jsx :
```
```
const Footer = () => {
  return (
    <footer className="footer flex justify-center items-center p-5
w-full">
      All rights reserved &copy; Alyra {new Date().getFullYear()}
    </footer>
  );
};

export default Footer;
```

```
# Layout.jsx :
```
```
import Header from "./Header";
import Footer from "./Footer";

const Layout = ({ children }) => {
  return (
    <div className="app flex flex-col h-screen">
      <Header />
      <main className="main grow">{children}</main>
      <Footer />
    </div>
  );
};

export default Layout;
```
```
# /app/page.js :
```
```
export default function Home() {
  return <div>welcome</div>;
}
```
```
# update the file /app/layout.js
# /app/layout.js
# import Layout from "@/components/shared/Layout";
# surround the children with this layout
```
```
<RainbowKitAndWagmiProvider>
        <Layout>{children}</Layout>
</RainbowKitAndWagmiProvider>
```
```
# test : npm run dev

# create a component NotConnected.jsx (rafce)
# create a component Bank.jsx (rafce)

# update the page.js
# page.js :
# create a display when connected or not
```
```
"use client";
import { useAccount } from "wagmi";
import Bank from "@/components/Bank";
import NotConnected from "@/components/NotConnected";

export default function Home() {
  const isConnected = useAccount();
  return <>{isConnected ? <Bank /> : <NotConnected />}</>;
}

```
```
```

```
# install components from shadcn-UI (https://ui.shadcn.com/)
npx shadcn-ui@latest add alert toast input button
# install RocketIcon/Terminal from lucide-react
npm install lucide-react
# copy paste import and insert component on the pages depends of the
use case (NotConnected.jsx / Bank.jsx)
# update the NotConnected.jsx page with components and imports
``
"use client";
import { Alert, AlertDescription, AlertTitle } from "@/components/
ui/alert";
import { RocketIcon, Terminal } from "lucide-react";

const NotConnected = () => {
  return (
    <div className="h-full w-full flex items-center justify-center">
      <Alert>
        <Terminal className="h-4 w-4" />
        <AlertTitle>Warning</AlertTitle>
        <AlertDescription>
          Please connect your wallet to use the app !
        </AlertDescription>
      </Alert>
    </div>
  );
};

export default NotConnected;
``
# update the Bank.jsx page with components and imports
``

``
# Fix the error in the next config file
# next.config.mjs:
``
/** @type {import('next').NextConfig} */
const nextConfig = {
  webpack: (config) => {
    config.externals.push("pino-pretty", "lokijs", "encoding");
    return config;
  },
};

export default nextConfig;
`
# INFO : you can update the color with className="bg-[#F00]">

# WARNING : the toaster component must be directly in the layout.js
# import + component
``
<RainbowKitAndWagmiProvider>
<Layout>{children}</Layout>
</RainbowKitAndWagmiProvider>
```

```
<Toaster />
```

```
# For interact with the smart contract we need (address and ABI)
# create folder constants
# create file index.js
```
```
export const contractAddress =
"0x5fbdb2315678afecb367f032d93f642f64180aa3";
export const contractABI =[find it in => backend/artifacts/contract/
NameOfTheContract.json]
```
```
# Create the components who will be used in the Bank component
cd components
touch Balance.jsx Withdraw.jsx Deposit.jsx Events.jsx
# rafce in each for the moment
# update them imports in Bank.jsx
```
```
import Balance from "./Balance";
import Withdraw from "./Withdraw";
import Deposit from "./Deposit";
import Events from "./Events";
```
```
//--- Deposit.jsx : ---\\
# Deposit.jsx :
# Will be a button for interact so we need to 'use client' and a
useState
# Import the shadcn components (Input & Button)
# create a 'state' for manage
# create the deposit function
# retrieve the entered value and use it for update the
state(onChange={(e) => setDepositValue(e.target.value)})
# trigger the Deposit function when the button is clicked
(onClick={function})
# import wagmi's functions useWriteContract,
useWaitForTransactionReceipt, useAccount
# const { address } = useAccount();
# import parseEther
# import { parseEther } from "viem";
# INFOS : WAGMI = React's framework for interact with blockchains
https://wagmi.sh/react/getting-started
# INFOS to read => https://wagmi.sh/react/guides/write-to-
contract#write-to-contract
# useWriteContract() => Retrieve the details (hash etc) of the
transaction after write on the smart contract (writeContract (= the
function))
# const { data: hash, isPending, writeContract } =
useWriteContract();
# useWaitForTransactionReceipt() = manage the status of the
transaction and retrieve the hash when it is done
# const { isSuccess: isConfirmed, isLoading: isConfirming }
=useWaitForTransactionReceipt({
hash,});
# trigger the writeContract function in the deposit function
```

```
# set the results of writeFunction({
# address:contractAddress,
# abi:contractAbi,
# functionName:'deposit',
# value:parseEther(depositValue),
# account: address})
# and import them
# use a check for be sure the input is a number if(!
isNaN(depositValue){writeContract} else ERROR_MESSAGE)
# display an alert based on the status of the transaction
(isConfirming, isConfirmed..)
# {isConfirming && <Alert>etc...<Alert/>}
# {isConfirmed && <Alert>etc...<Alert/>}
# import the components for display the alert..

#INFOS: useReadContract() => read infos in the contract (address,
abi, functionName, account, refetch,..)
#INFOS: refetch : permet de relancer la fonction un peu plus tard
#comme utilisée dans plusieurs composants, on le met dans un
composant #supérieur donc Bank.jsx

//--- Bank.jsx : ---\\
``
import { useAccount, useReadContract } from "wagmi";
import { contractABI, contractAddress } from "@/constants";
``
const { address } = useAccount();

const {
data: balanceOfConnectedAddress,
error,
isPending, refetch,
} = useReadContract({
address: contractAddress,
abi: contractABI,
functionName: "getBalanceOfUser",
account: address,
});
`
# put the inputs in the <Balance /> component
<Balance isPending={isPending} balance={balanceOfConnectedAddress} /
>

//--- Balance.jsx : ---\\
# add the props in the Balance component
const Balance = ({ balance, isPending }) => {...}
# display result based on status and import the function formatEther
(wei to Eth)
``
import { formatEther } from "viem";

const Balance = ({ balance, isPending }) => {
  return (
    <div className="mb-2">
```

```
      {isPending ? (
        <p>Loading...</p>
      ) : (
        <p>
          Your balance is : <span>{formatEther(balance.toString())}
ETH</span>
        </p>
      )}
    </div>
  );
};

export default Balance;
```

```
//--- Withdraw.jsx : ---\\
# copy paste deposit.jsx in withdraw, they're very similar
# WARNING : writeContrat() haven't value but an argument because of
the function withdraw triggered
# deposit is payable and without args and withdraw isnt payable but
with an argument
args: [parseEther(withdrawValue)],

//--- Bank.jsx : ---\\
# make a refetch when deposit or withdraw isConfirmed
```
```
<Deposit refetch={refetch} />
<Withdraw refetch={refetch} />
```

```
//--- Deposit.jsx : ---\\
# insert the props {refetch}
const Deposit = ({ refetch }) => {...}
#update the useEffect
```
```
useEffect(() => {
    if (isConfirmed) {
      refetch();
    }
  }, [isConfirmed]);
```
```
//--- Withdraw.jsx : ---\\
# insert the props {refetch}
const Withdraw = ({ refetch }) => {...}
#update the useEffect
```
```
useEffect(() => {
    if (isConfirmed) {
      refetch();
    }
  }, [isConfirmed]);
```

```
# clean the input fiedl after the date is used
```

```
useEffect(() => {...
setDepositValue("");
...
}, [isConfirmed]);
# track directly in the imput the value of the state with
<Input ... value={depositValue}/>

# error management
# we'll use toasters
# must be in the layout.jsx for be common at all the components
<Toaster />
# use the function useToast in the component
``
const { toast } = useToast();
``
# use it when deposit failed
``
const Deposit = () => {
if (!isNaN(depositValue) && depositValue > 0) {
writeContract({
address: contractAddress,
abi: contractABI,
functionName: "deposit",
value: parseEther(depositValue),
account: address,
});
} else {
toast({ title: "Please enter a valid number", description: "Please
enter a valid number",className: "bg-red-500", });
}
};
 ``
# ALTERNATIVE
``
else {
alert("{address} Please enter a valid number");
}
``
# Manage the different <Alert /> components depending on the status
of the transaction
``
{hash &&
            <Alert className="bg-lime-400 text-black mb-2">
                <AlertTitle>Heads up!</AlertTitle>
                <AlertDescription>
                    Transaction Hash: {hash}.
                </AlertDescription>
            </Alert>
        }
        {isConfirming &&
            <Alert className="bg-orange-400 text-black mb-2">
                <AlertTitle>Heads up!</AlertTitle>
                <AlertDescription>
                    Waiting for confirmation...
```

```
                </AlertDescription>
            </Alert>
        }
        {error && (
            <Alert className="bg-red-600 text-white mb-2">
                <AlertTitle>Error</AlertTitle>
                <AlertDescription>
                    Error: {(error).shortMessage || error.message}
                </AlertDescription>
            </Alert>
        )}
        {isConfirmed &&
            <Alert className="bg-lime-400 text-black mb-2">
                <AlertTitle>Heads up!</AlertTitle>
                <AlertDescription>
                    The transaction has been confirmed.
                </AlertDescription>
            </Alert>
        }
```

```
# manage the alerts in a separate component
# create Informations.jsx component
# put the component with the props in deposit and withdraw
<Informations hash={hash} isConfirmed={isConfirmed}
isConfirming={isConfirming} error={error} />
```

```
// @ts-nocheck
import { Alert, AlertDescription, AlertTitle } from "@/components/
ui/alert";

const Informations = ({ hash, isConfirming, error, isConfirmed }) =>
{
  return (
    <>
      {hash && (
        <Alert className="bg-lime-400 text-black mb-2">
          <AlertTitle>Heads up!</AlertTitle>
          <AlertDescription>Transaction Hash: {hash}.</
AlertDescription>
        </Alert>
      )}
      {isConfirming && (
        <Alert className="bg-orange-400 text-black mb-2">
          <AlertTitle>Heads up!</AlertTitle>
          <AlertDescription>Waiting for confirmation...</
AlertDescription>
        </Alert>
      )}
      {error && (
        <Alert className="bg-red-600 text-white mb-2">
          <AlertTitle>Error</AlertTitle>
          <AlertDescription>
            Error: {error.shortMessage || error.message}
          </AlertDescription>
```

```
        </Alert>
      )}
      {isConfirmed && (
        <Alert className="bg-lime-400 text-black mb-2">
          <AlertTitle>Heads up!</AlertTitle>
          <AlertDescription>
            The transaction has been confirmed.
          </AlertDescription>
        </Alert>
      )}
    </>
  );
};

export default Informations;

``


# EVENTS
# for retrieve the events i need to create a new folder and a file
# create folder utils, file client.js
# client.js
``
import { createPublicClient, http } from 'viem'
import { sepolia } from 'viem/chains'

export const publicClient = createPublicClient({
  chain: sepolia,
  transport: http(process.env.RPC)
})
``
# we will do a getEvents() in the useEffect() and we need to make it
in the parent component
//--- Bank.jsx : ---\\
# imports publicClient and parseAbiItem)
``
import { publicClient } from "@/utils/client"
import { parseAbiItem } from "viem"
 ``
# retrieve the events in an array with the useState()
const [events, setEvents] = useState([])
# create the function getEvents()
# we need to use the getLogs() function of viem
# return a list of events logs matching parameters
# concat them and sort them and setEvents in the useState()
``
const getEvents = async() => {
    const depositEvents = await publicClient.getLogs({
      address: contractAddress,
      event: parseAbiItem('event etherDeposited(address indexed
account, uint amount)'),
      fromBlock: 0n,
      toBlock: 'latest'
    })
```

```
    const withdrawEvents = await publicClient.getLogs({
      address: contractAddress,
      event: parseAbiItem('event etherWithdrawed(address indexed
account, uint amount)'),
      fromBlock: 0n,
      toBlock: 'latest'
    })

    const combinedEvents = depositEvents.map((event) => ({
      type: 'Deposit',
      address: event.args.account,
      amount: event.args.amount,
      blockNumber: Number(event.blockNumber)
    })).concat(withdrawEvents.map((event) => ({
      type: 'Withdraw',
      address: event.args.account,
      amount: event.args.amount,
      blockNumber: Number(event.blockNumber)
    })))

    combinedEvents.sort(function(a,b) {
      return b.blockNumber - a.blockNumber
    })

    setEvents(combinedEvents);
  }
``
# update the component with the props
<Events events={events} />
# useEffect
``
  useEffect(() => {
    const getAllEvents = async () => {
      if (address !== "undefined") {
        await getEvents();
      }
    };
    getAllEvents();
  }, [address]);
``
//--- Events.jsx : ---\\
``
import {
  Table,
  TableBody,
  TableCaption,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table";

import { Badge } from "@/components/ui/badge";
```

```
import { formatEther } from "viem";

const Events = ({ events }) => {
  return (
    <>
      <h2 className="text-4xl font-extrabold mt-4">Events</h2>
      <Table className="mt-4">
        <TableCaption>A list of your recent invoices.</TableCaption>
        <TableHeader>
          <TableRow>
            <TableHead className="w-[200px]">Type</TableHead>
            <TableHead>Address</TableHead>
            <TableHead className="text-right">Amount</TableHead>
          </TableRow>
        </TableHeader>
        <TableBody>
          {events.map((event) => (
            <TableRow key={crypto.randomUUID()}>
              <TableCell className="font-medium">
                {event.type === "Deposit" ? (
                  <Badge className="bg-lime-400">Deposit</Badge>
                ) : (
                  <Badge className="bg-red-400">Withdraw</Badge>
                )}
              </TableCell>
              <TableCell>{event.address}</TableCell>
              <TableCell className="text-right">
                {formatEther(event.amount.toString())} ETH
              </TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
    </>
  );
};

export default Events;

``
# add getEvents to deposit and withdraw
``
const Deposit = (refetch, getEvents) => {...
.....
.....
  useEffect(() => {
    if (isConfirmed) {
      refetch();
      setWithdrawValue("");
      getEvents();
    }
  }, [isConfirmed]);
 }
```

``

#INFOS:
parseEther (etherString) => BigNumber
Parse the etherString representation of ether into a BigNumber instance of the amount of wei.
formatEther (wei) => string
Format an amount of wei into a decimal string representing the amount of ether. The output will always include at least one whole number and at least one decimal place, otherwise leading and trailing 0’s will be trimmed.

DEPLOYMENT
cf video 1h43..

OLD jsconfig.json
```
{
  "compilerOptions": {
    "paths": {
      "@/*": ["./*"]
    },
  }
}
```

OLD jsconfig.json
Enable the auto-import
```
{
  "compilerOptions": {
    "jsx": "preserve",
    "paths": {
      "@/*": ["./*"]
    },
    "checkJs": true,
  }
}
```

la faille de sécurité est bien au niveau de la boucle for du tallyVotes ?
Fil d'Ariane (breadcrumb)