

ブロックチェーン公開講座 第11回

ブロックチェーンアプリケーションとWeb3 のためのWeb2技術

芝野恭平

東京大学大学院工学系研究科技術経営戦略学専攻

ブロックチェーンイノベーション寄付講座

特任研究員

shibano@tmi.t.u-tokyo.ac.jp



Agenda

- Web3開発者になるには，講義の位置づけ
- Web3システムのシステムデザイン：
 - バックエンドサーバーなし
 - バックエンドサーバあり
- サンプルアプリケーション：
 - システム構成
 - 実際に動かしてみよう
 - 演習.



Web3開発者になるには：今回の講義の位置づけ

技術積み上げ（イメージ）



- 開発者のプロとなるには長い道のりが必要。
 - 自身の試行錯誤の積み上げで技術を身につけている方が多いです。
 - DYOR (Do Your Own Research)はWeb3界隈に限らず、開発者の道を志す中でも重要なマインドセットです。
 - 職業プログラマーには、開発経験が何十年の方もいます。
 - 様々な専門分野ごとに専門家がいます。
 - 今回の講義は、開発経験のない方には内容の理解が困難な部分も含まれます。
- 今回の講義は、最新の実装環境を用い、比較的簡単なコード操作でサンプルプログラムを通じてWebブラウザからブロックチェーンに接続してコントラクトを操作する、という体験ができるように構成されています。

Web3開発者になるには：初学者の方へ

- 今回や前回・前々回の講義の内容の理解のみでプロのWeb3開発者になることは困難です。
 - 将来本格的に開発者になろうとしたときに、参考資料の一つとして活用ください。
- ソフトウェア開発者になるには？
 - （以下、私見になります。
 - とにかく開発業務（コードを書いたり，LinuxのCLI環境に慣れたり）を行う時間を増やしましょう。
 - 今回や前回の講義の中で何をやっているのか，わからなかったところをネット検索などで調べたりし，まずは目の前にあるわからない箇所を潰しましょう。（DYORです）
 - Web3に限らず，何かしらのアプリを自分で考えて最後まで作ってみる，ということをやってみることをおすすめします。
 - 一通りの工程を経験できて，最後まで作り上げるということは経験にもなりますし，自信にも繋がります。
 - Web3アプリの開発は様々な技術の組み合わせが必要な部分が多いのに加えて情報がまだまだ多くなくさらに開発環境も洗練されていないことが多いため，結構苦労すると思います。

Web3開発者になるには：初学者の方へ

- 今回の講義中に、コマンドを実行していく中で何かしらエラーが生じる場合もあります。
 - 個々人の環境が異なり、全ての方の環境上での動作確認はできていないことに起因します。
 - PCのOS, バージョン, ウイルス対策ソフト, ネットワーク環境, etc
- そのような場合は、焦らずに何かしらエラーメッセージが表示されている場合はそのメッセージを検索して解決策を探しましょう。
 - 講義には遅れるかもしれませんが、後々動画でキャッチアップすることも可能です。
- エラーはひとまずおいておき、講義の後に時間をかけて解決をしていくのもいいと思います。
- また、ソースコードの編集を行う場合も「正解じゃないと入力してはいけない」などと考えることなくトライアンドエラーの繰り返しで進める思考が重要です。動かなければ考え直してもう一回編集して進めていきましょう。
- 今回の講義の演習部分で特に重要なことは、サンプルプログラムが動くことではなく、アプリケーション開発というのは数々の多くのコンポーネントの組み合わせでできているため、結構動作させることは難しい、という部分を体験してもらうことでもあります。

前回の内容要約

初学者でもコントラクト開発の工程を体験できるやり方の紹介。

スキルが必要な各工程をAIを用いてカットし、本質的に作りたい機能を考えることに注力できる。

この工程をAIで省力化

どんなDAOにしよう
か考える



システムフローの構築



Solidityのコード作成



ビルド・デプロイ



eraser.io

eraser.ioを用いることでAIで簡単にシーケンス図を作成可能。

- 処理フローを確認しつつDAO設計に注力ができる。
- 特定のコード形式 (Mermaid like) で出力可能なため、情報の欠落を極力少なくChatGPTに伝えられる。



GPTs

ChatGPTでビジネスロジックを自動生成。↓の基盤上で処理されるため、比較的複雑な処理をコード生成したものを使用可能。



Meta Contractフレームワーク

ストレージとビジネスロジック(function)を分離し、function間での共通データを取り扱うことを実現。
開発者はfunctionをどんどん追加することが可能で、コード量などの制約をそこまで意識することなく容易に機能の拡張が可能。



Foundry

Foundryを用いてビルド・デプロイ。

(参考)
通常の開発フロー

自分でシーケンス図のツールを使ってフローを考えて作る。
フロー構築のスキルに加えシーケンス図作成ツールの使用スキルもが必要。

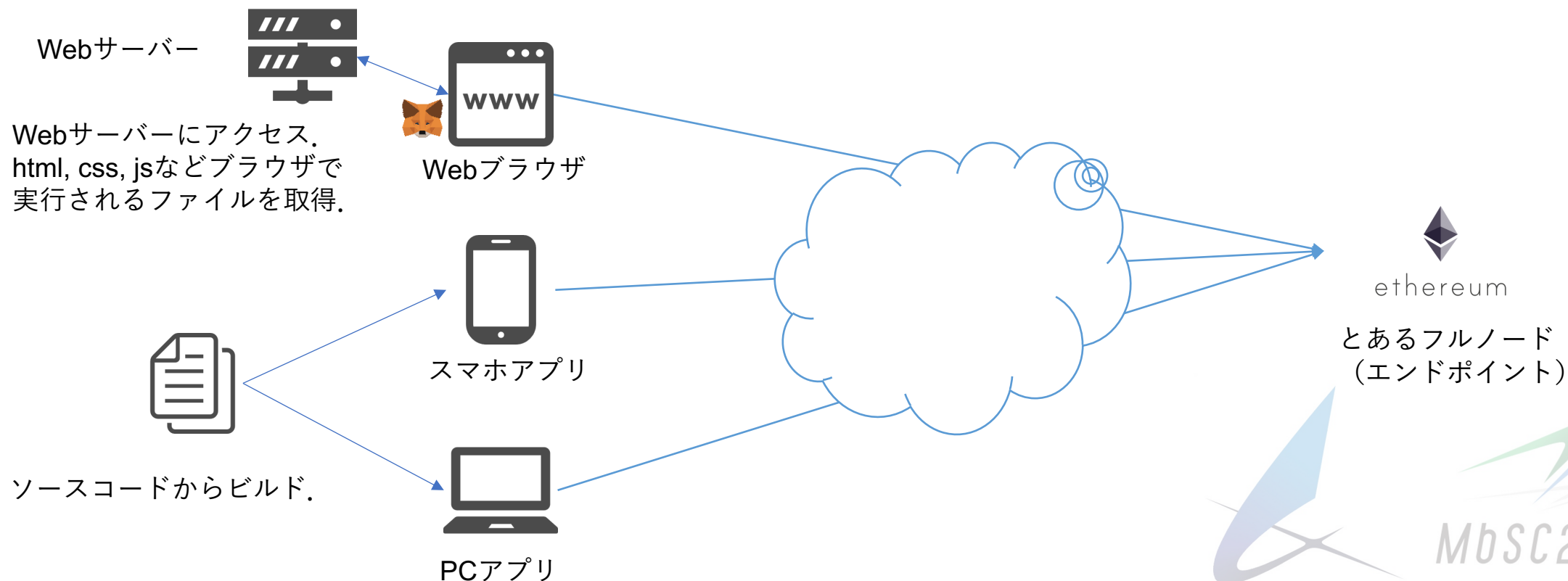
シーケンス図をもとに、contractの設計や各種ロジックを記述する。
Solidityやスマートコントラクトの制約を意識したプログラミングスキルが必要。

Web3アプリのシステムデザイン

- Web3アプリといいつつ、利用者が直接ブロックチェーンにアクセスして操作を行うことは難しく、ユーザーインターフェースとなるシステムが必要になります。
- Web3のシステムを設計する上で、他のシステムと最も違うところは、検証可能性をできる限り担保すべき、という点です。
- ブラックボックスが一部でも介在すると、そこがトラストポイントになってしまい、トラストレスなシステムではなくなります。
 - 逆に言うと、トラストポイントを設計に取り入れることでうまくシステムを構築するのも一案です。
- トラストレスなシステムを実現する方法の一つとして、Javascriptなどのスクリプト言語そのままクライアント側で実行できて、そこから直接ブロックチェーンにアクセスする方法が挙げられます。
 - もしくは、ソースコードが公開されているアプリを自身でビルドし実行するのも同じく検証可能である、と言えるでしょう。
- しかしながら、その設計ではUI/UX的に優れたアプリケーションが作りにくく、中央集権型のシステムと同様、サーバーをシステム内にうまく取り入れて設計されるケースが多いです。

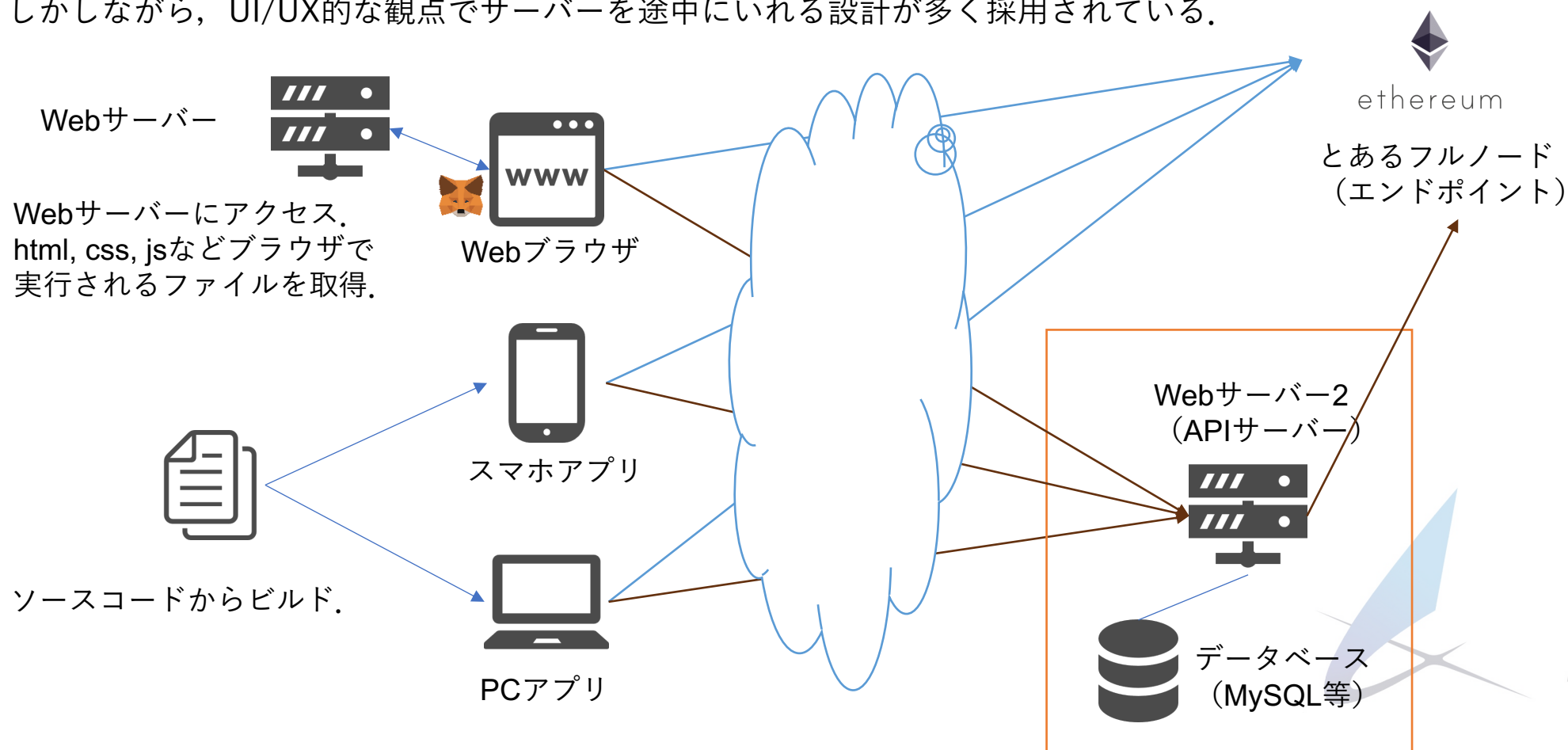
システムデザイン：バックエンドサーバーなし

- 完全トラストレスな構成にする場合.
- クライアントソフトウェアとブロックチェーンの2層構造

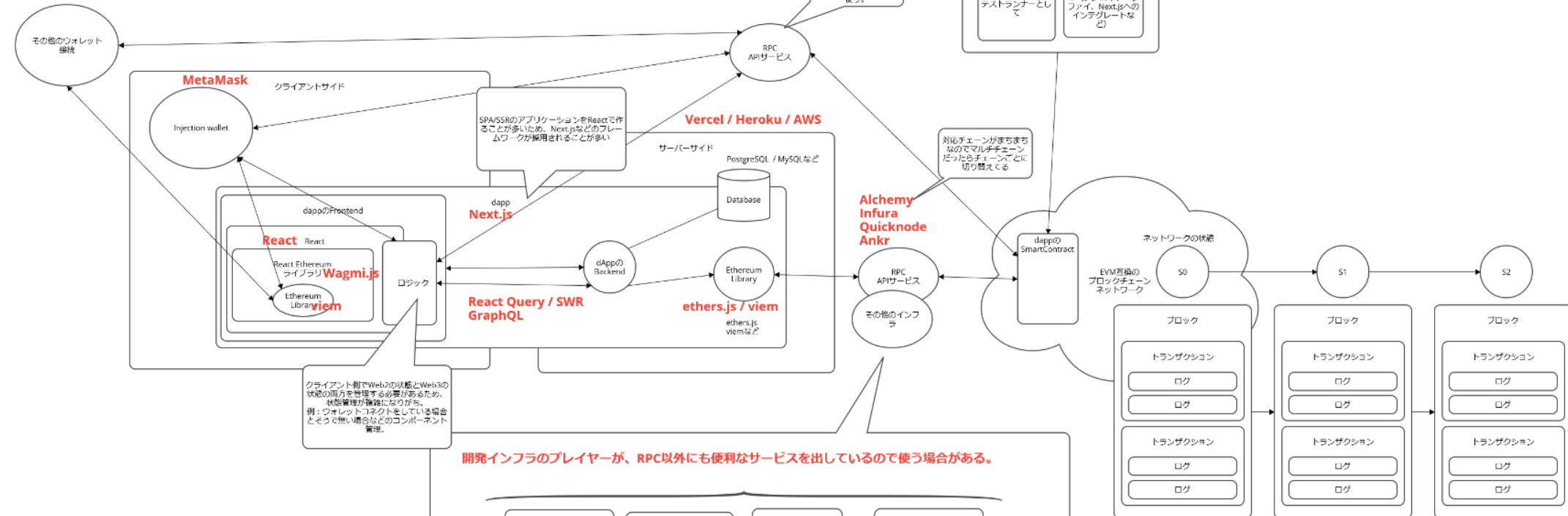


システムデザイン：バックエンドサーバーあり

- ユーザーから見るとサーバーが入る構成にするとブラックボックスになるので、完全トラストレスではなくなってしまう。
- しかしながら、UI/UX的な観点でサーバーを途中にいれる設計が多く採用されている。

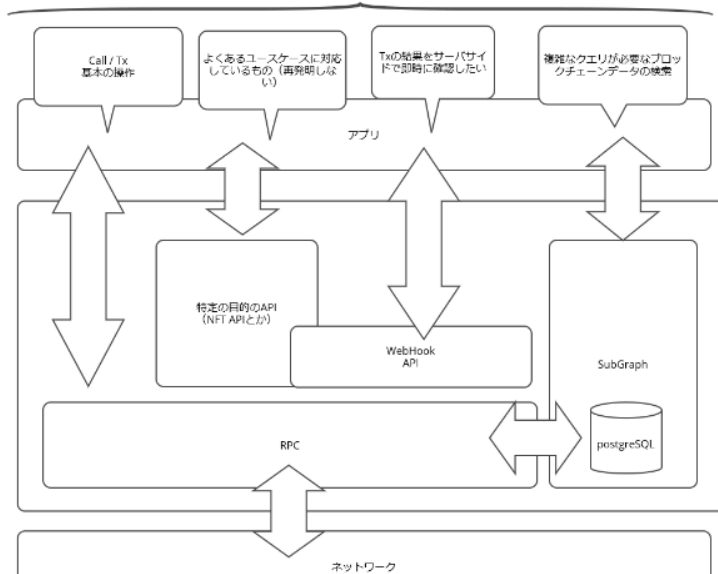


wallet connectとか



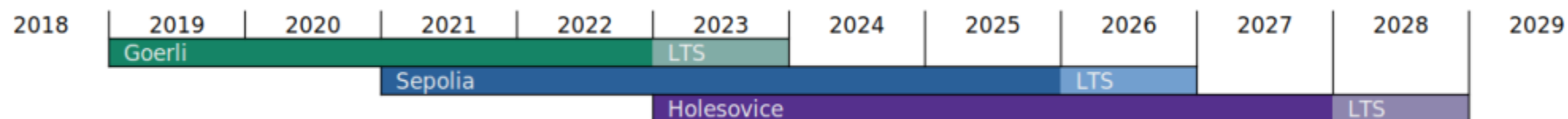
クライアント側でWeb2の状態とWeb3の状態の両方を管理する必要があるため、状態管理が複雑になるが、例：ウォレットコネクトをしている場合とそうでない場合などのコンポーネント管理。

開発インフラのプレイヤーが、RPC以外にも便利なサービスを出しているので使う場合がある。



Ethereumテストネットについて

- 開発を行う上で、実際にブロックチェーンに接続したいケースがあります。
 - コントラクトのロジックのみの開発中はFoundryのみでOK
 - ウォレットの接続などを含めた、最終的な全体テストをする場合などではテストネットを利用することを推奨します。



<https://github.com/eth-clients/holesky>

2024年6月現在では、[Sepolia](#) と [Holesovice](#) の2つのテストネットが使用可能。
用途としては、

- Sepolia : Dappsやコントラクト開発のテスト用
 - Holesovice : PoSやプロトコルのテスト用
- となっています。

テストネットのETH取得の難易度の差はあるにせよ、通常のDapps開発のテストを行う際はSepoliaを使いましょう。
また、他のEVM互換チェーンやL2のテストネットの利用も検討するとよいでしょう。

今日の演習について

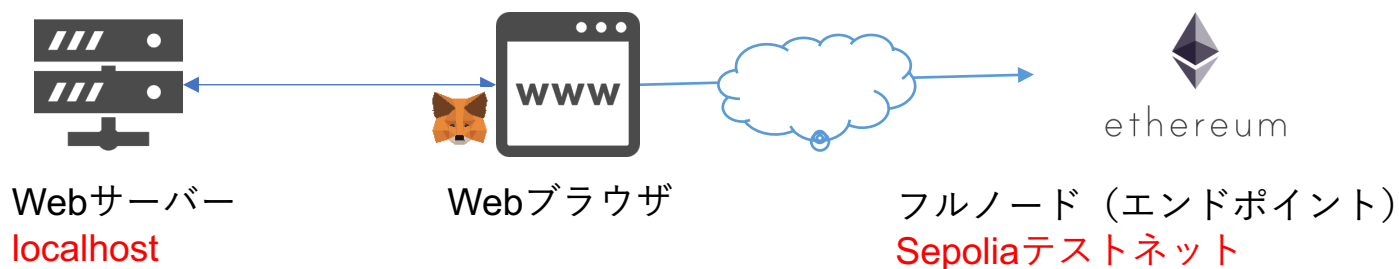
- 簡易版TextDAOを使用したアプリケーションを開発します.
 - 元々のTextDAOから機能を大幅に削除し簡易的に動作するようにしましたものです.
 - <https://github.com/blockchaininnovation/TextDAO>
- コントラクトはすでにSepoliaテストネットにデプロイ済みです.
 - 今日の演習ではコントラクトのソースコードは触りません.
 - <https://sepolia.etherscan.io/address/0x479E7BcBdBF126D8d5B9db44c38f02BA01cA6BC5>
- このスマートコントラクトアプリに対して各種処理を呼び出すためのフロントエンドアプリケーションの開発をサンプルプログラムを通して行います.
 - サンプルプログラムはSparkleAIさん, D-chainさん, Ecdysisさんの協力で完成できました. 誠にありがとうございました.



サンプルプログラムの構成

- <https://github.com/blockchaininnovation/frontpractice>
- バックエンドサーバーなしの構成です。
 - ※ 厳密にはNext.jsによるレンダリングなどの一部の処理はサーバー側で実行されています

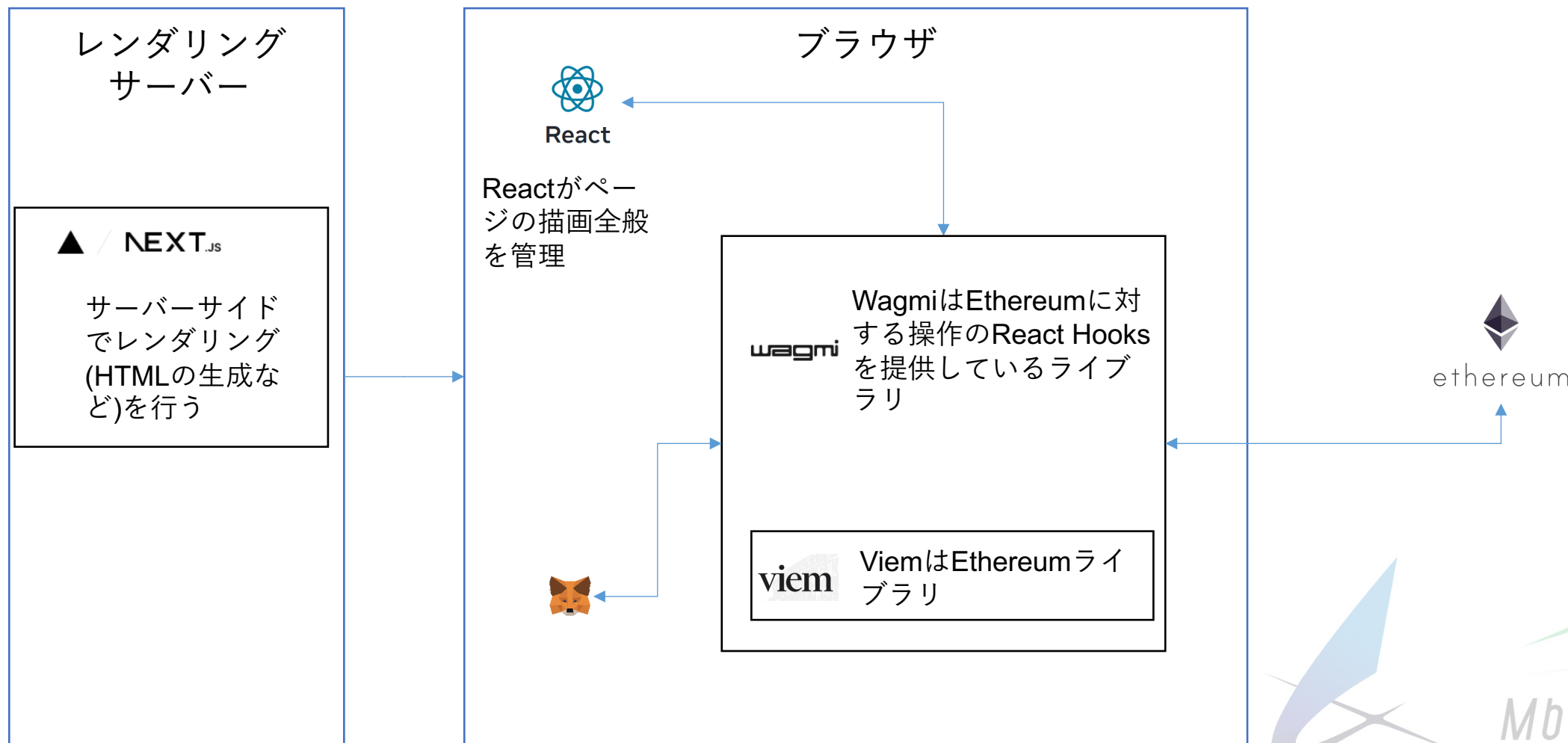
- React
- Next.js
→ フロントエンド開発フレームワーク
- Wagmi.js
- Viem
→ Ethereum接続用ライブラリ



ブラウザからブロックチェーンにアクセス。
ウォレット機能はMetamaskで。



サンプルプログラムの構成



サンプルプログラムを動かしてみよう：ログの重要性

- ログを見るようにしましょう。
 - コンソールでコマンドを実行したり，プログラムを書いてビルドに失敗することは日常茶飯事です。
 - 英語のログは初めは戸惑うかもしれませんが，重要な情報が詰まっています。
 - ログメッセージそのものをネットで検索すると，似たようなエラーに出くわしている人の解決策に出会えるかもしれませんし，生成系AIに解決方法を聞いてもよい答えが得られるかもしれません。
- ログを出力するようにしましょう。
 - コーディングをするときには，適切な箇所で適切なログを出力するようにしましょう。
 - 思ったような動作をしないときは，各種変数やどこまで処理が進んでいるのかなど，ログで確認しつつ作業を進めるようにしましょう。

サンプルプログラムを動かしてみよう

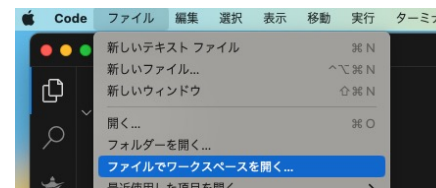
- Gitからコードを取得
 - ※ コマンドの実行はターミナルを開いて行ってください。
 - Macの人はターミナル, Windows (WSL/Ubuntu) の人は「Ubuntu」のターミナルです。
 - ホームディレクトリにgitディレクトリを作成して, その中にコードを取得しています。

```
cd ~  
mkdir git  
cd git  
git clone https://github.com/blockchaininnovation/frontpractice.git
```

- VSCodeで取得したコードを開きます。

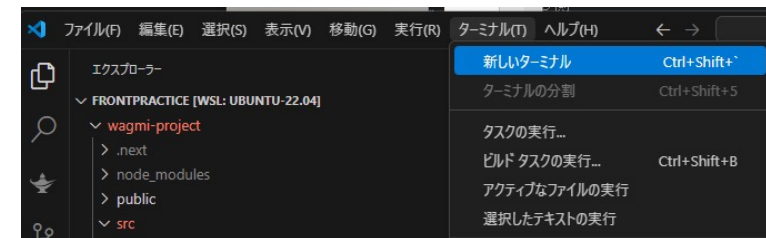
```
code frontpractice
```

※ codeコマンドが効かない場合はVSCodeを開いて, フォルダを開くから上記で取得したfrontpraticeディレクトリを開いてください。



ビルド・実行 (npm install)

- VSCode内のターミナルを開きます。
- この後はVSCode内のターミナルで作業を行います。
 - これはターミナル (Macのデフォルトの, もしくはUbuntuの) で作業をしていることと同じです。
 - ターミナルとソースコードエディタのウィンドウの行き来をすると効率が悪いのでVSCode内にはターミナルを開く機能が搭載されています。
- アプリケーションの開発ではいくつものソースコードを編集するため, ファイル間を何度も行き来したり, ターミナルで作業をしたりします. そのために, VSCodeに代表される開発に便利なエディタが使われます。



Nodejsの必要なライブラリのインストール (環境によって5分ほど時間がかかります)

```
shibano@ryzenjisaku:~/git/frontpractice$ cd wagmi-project
shibano@ryzenjisaku:~/git/frontpractice/wagmi-project$ npm install
```

ビルド・実行 (設定ファイル)

このプログラムには、ブロックチェーンの接続先と、簡易版TextDAOのコントラクトアドレスを指定する必要があり、それらの設定を行う設定ファイルがあります。

src/wagmi.tsファイル

接続先エンドポイントを指定する設定ファイルです。

wagmi.sample.tsファイルをコピーします。

```
shibano@ryzenjisaku:~/git/frontpractice/wagmi-project$ cp src/wagmi.sample.ts src/wagmi.ts
```

wagmi.tsファイルを開き（左側のエクスプローラーから選択すると右側に開きます），createConfig()内を編集しSepoliaエンドポイントを記述します。

```
export const config = createConfig({
  chains: [sepolia],
  connectors: [],
  ssr: true,
  transports: {
    [sepolia.id]: http("https://eth-sepolia.g.alchemy.com/v2/XXXXXXXXXXXXXXXXXXXXXXXXXXXX"),
  },
});
```

上記<https://eth-sepolia.g.alchemy.com/v2/XXXXXXXXXXXXXXXXXXXXXXXXXXXX>の箇所について、自身のAPIキーに書き換えてください。これはalchemyの例で、Infuraの場合は <https://sepolia.infura.io/v3/XXXXXXXXXXXXXXXXXXXXXXXXXXXX> のような形式です。

ビルド・実行 (設定ファイル)

このプログラムには、ブロックチェーンの接続先と、簡易版TextDAOのコントラクトアドレスを指定する必要があり、それらの設定を行う設定ファイルがあります。

.env.localファイル

コントラクトアドレスを設定するファイルです。
.env.sampleをコピーして使用します。

```
shibano@ryzenjisaku:~/git/frontpractice/wagmi-project$ cp .env.sample .env.local
```

.env.localファイルを開き、コントラクトアドレスを追記します。
今回のコントラクトアドレスは0x479E7BcBdBF126D8d5B9db44c38f02BA01cA6BC5 です。

```
NEXT_TELEMETRY_DISABLED=1  
  
NEXT_PUBLIC_CONTRACT_ADDR=0x479E7BcBdBF126D8d5B9db44c38f02BA01cA6BC5
```

※ Sepoliaのetherscanで実行された関数の履歴など確認できます。

<https://sepolia.etherscan.io/address/0x479E7BcBdBF126D8d5B9db44c38f02BA01cA6BC5>

ビルド・実行（開発モードで起動）

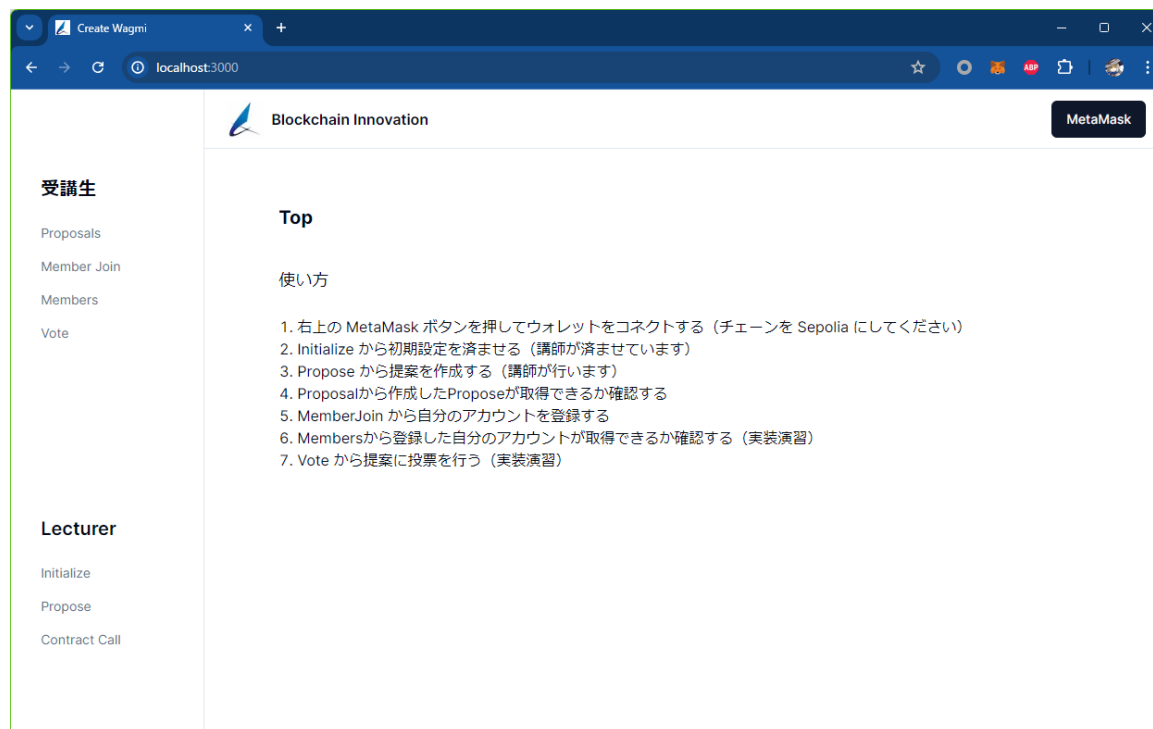
コンソールで以下のコマンドを実行します。

```
shibano@ryzenjisaku:~/git/frontpractice/wagmi-project$ npm run dev
```

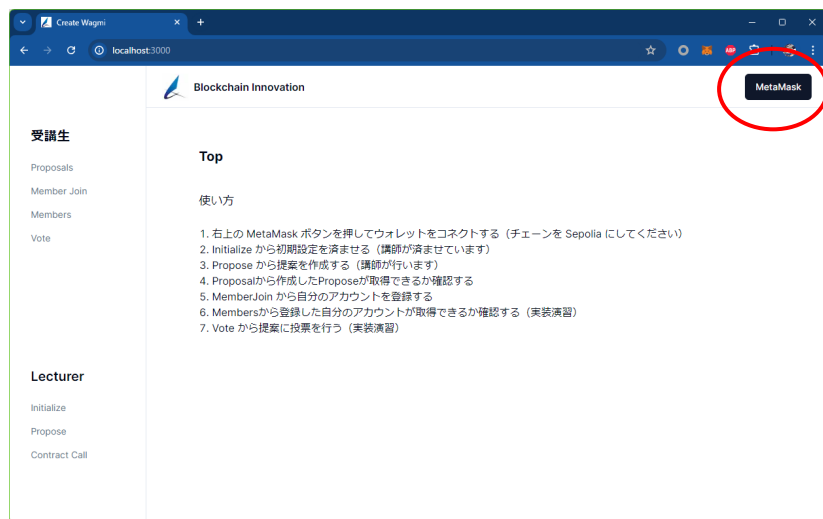
なにか処理が始まり、プロンプトが返らず実行中の状態になります。

この状態でブラウザから <http://localhost:3000> にアクセスしてください。

以下のページが表示されれば成功です。これで、frontpracticeアプリそのものは無事に起動されていることがわかります。

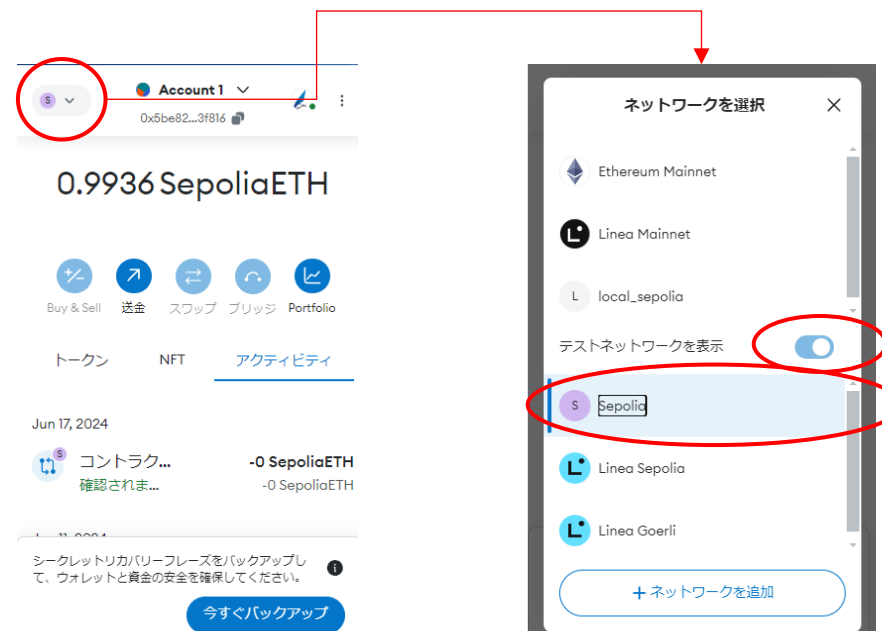


ウォレットとの接続



右上の「MetaMask」ボタンを押してください。
すると、MetaMaskの画面が開きます。

ネットワークをSepoliaにし、今回使いたいアカウントを選択してください。

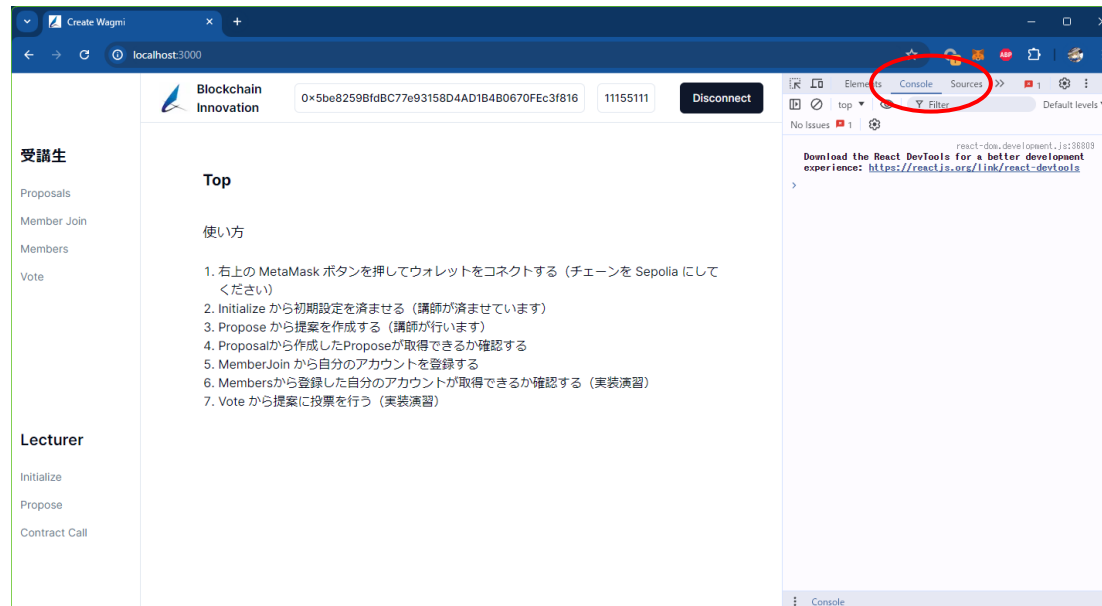


ログを出力してみる

- 開発時にはソースコードを改変していく中で、変数の値などを確認しつつ動作が思った通りになっているかをチェックしていくことが効率的です。
- ログを出力したい箇所に`console.log()`を入れることでログの出力ができます。以下、例です。

```
console.log("aaa: " + val);
```

- ChromeでF12を押下しConsoleを開くと、そこにログが出力されます。

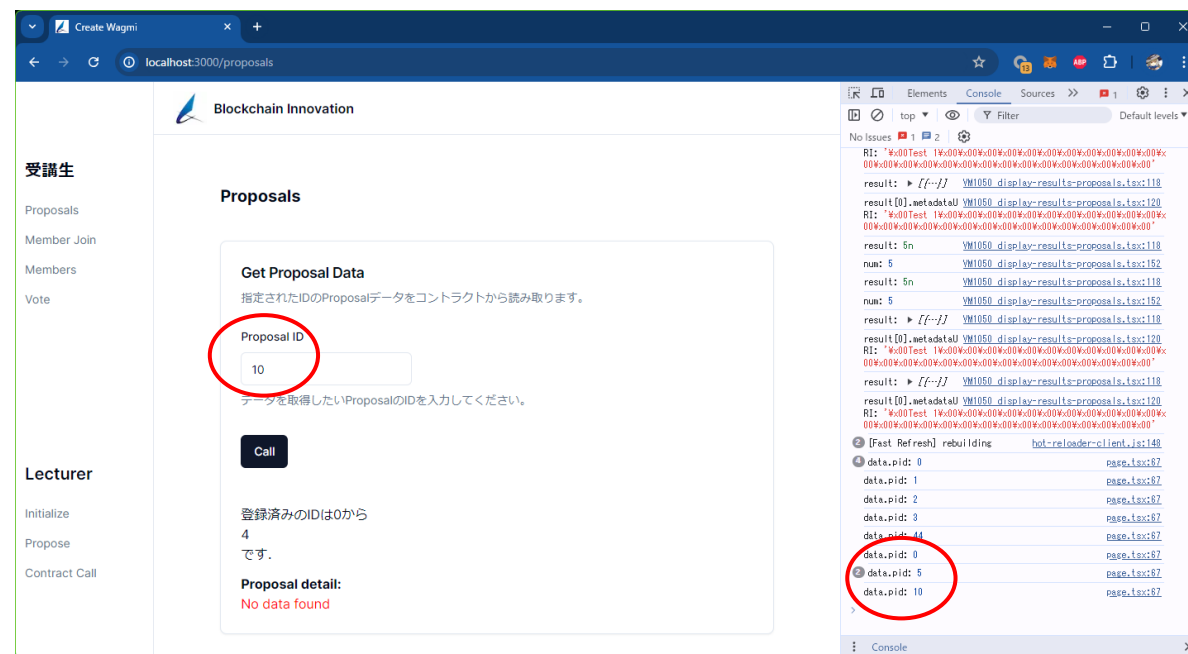


ログを出力してみる

- ボタンを押したときにテキストボックスに入力されているデータをログに出力してみます。
- `src/app/proposals/page.tsx`
- を開き, 67行目に`console.log()`でログを出力してみましょう。
- 以下のコードはProposal IDのテキストボックスに入力された値 (`data.pid`) を出力しています。

```
function getProposalData(data: contractCallPidSchemaType) {  
  console.log("data.pid: %o", data.pid);  
  setPid(data.pid);  
  proposalRefetch();  
}
```

ブラウザで「Proposals」を開き, 「Call」ボタンを押すとテキストボックスに入力された内容がログに出力されます。





スマートコントラクトの構造を理解しよう

- このプログラムはスマートコントラクトを動かすためのアプリであり、まずはスマートコントラクトアプリの仕組みを理解する必要があります。
- 簡易版TextDAOは次のページの流れで動作するように作られています。
 - オリジナルのTextDAOより大幅に機能を削減し、演習として適切なものになっています。

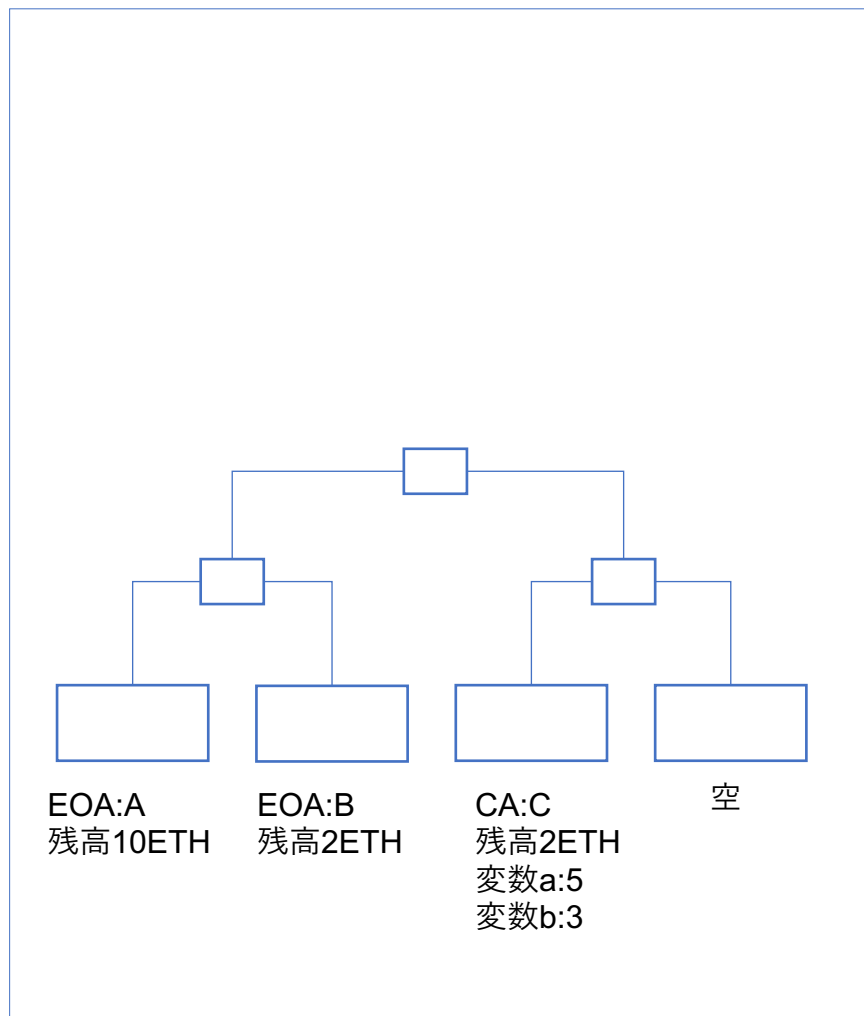


演習の流れ

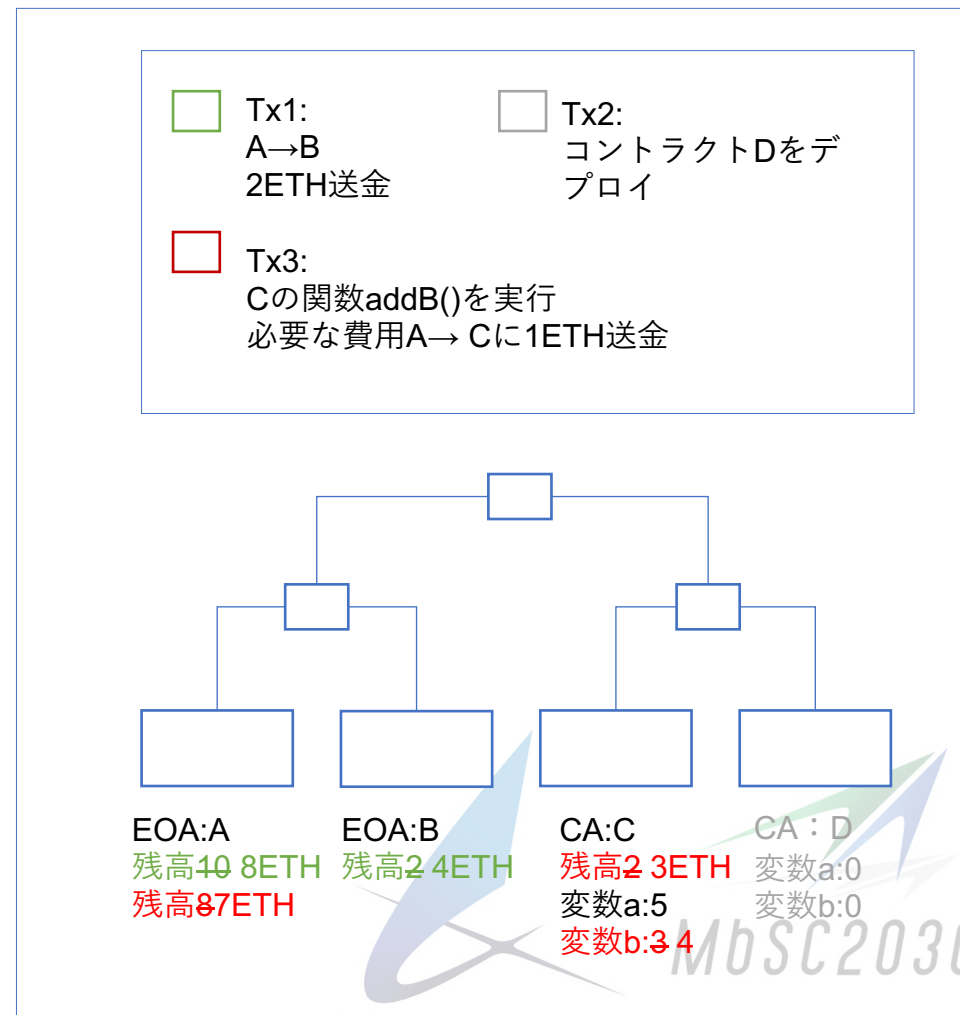
- 管理者が提案を起票
 - `propose()`
- 起票された提案の提案名, 投票数を確認
 - `Propose`には固有のIDが付与されており, IDを指定して提案内容を閲覧する.
 - `getProposalHeaders()`
 - `getNextProposalId()`
- メンバー登録
 - このDAOのメンバーのみ投票できます.
 - まずは自分のアカウントをメンバーとして登録する必要があります.
 - `memberJoin()`
- メンバー情報確認
 - `getMember()`
- メンバーが投票
 - `voteHeaders()`



スマートコントラクト復習：コントラクトとTxとステート



ブロック内のTxを実行することでステートが変化



スマートコントラクト復習：ガスが必要なケース，不要なケース

- ・ コントラクトの関数によって，使用する際にガスがかかるケースとかからないケースがあります。

ステートの更新が必要なケース

データの記録，更新をする場合はこちら。

EOAによるトランザクションの生成が必要で，ブロックチェーンに取り込まれて初めて有効になります。
記録するデータ容量， その際に実行する計算量に応じたガスが必要になります。

ステートの更新が不要なケース

データの記録・更新が行われない場合はこちら。

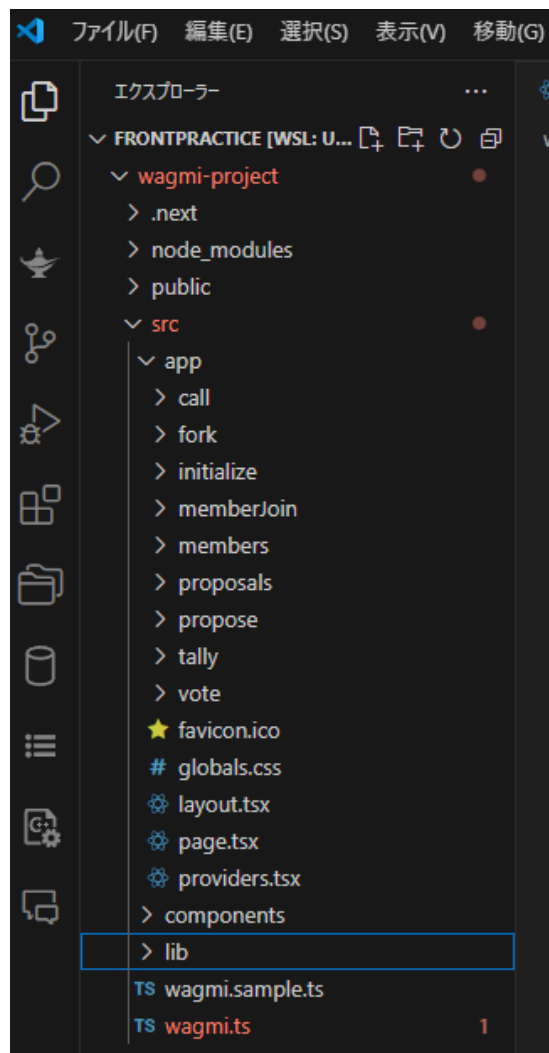
ブロックチェーンの現在のステートの問い合わせ，もしくはそこから計算によって得られる値を取得する場合は該当します。

この場合はTxの生成は不要で，ガス代もかかりません。

特定のフルノードに問い合わせをすることで，そのフルノードから値が返ってきます。



プロジェクトの説明：ディレクトリ構造（各ページ）



一つ一つのページについて：

`src/app/`

以下にディレクトリがあり，その中に`page.tsx`が入っています。
このディレクトリ名が一つのページの名前と一致しています。

例

<http://localhost:3000/proposals>

というページは

`src/app/proposals/page.tsx`

が該当のソース。

合わせて `src/lib/paths.ts` にもそのページの情報に記載されている。

これはサイドバー `src/components/sidebar.tsx` のリンク先を抽象化するために使用。

コントラクトから状態を取得しよう

- wagmi.jsのuseReadContracts()を使ってコントラクトの読み出し専用の関数を実行し、データを取得します。
- Proposalsページの例を見てみましょう：

Get Proposal Data
指定されたIDのProposalデータをコントラクトから読み取ります。

Proposal ID

データを取得したいProposalのIDを入力してください。

Call

登録済みのIDは0から4です。

Proposal detail:
ID: 0
title: Test 1
score: 8

<http://localhost:3000/proposals>

```
const {
  refetch: proposalRefetch,
  data: proposalData,
  isPending: isProposalPending,
} = useReadContracts({
  contracts: [
    {
      ...TextDAOFacade,
      functionName: "getProposalHeaders",
      args: [BigInt(pid)],
    },
    {
      ...TextDAOFacade,
      functionName: "getNextProposalId",
    },
  ],
});
```

src/app/proposals/page.tsx

これはReactのフックであり、コンポーネントの再レンダリング時に呼び出されます。

pidが状態として管理されており、更新されるとコンポーネントが再レンダリングされ、すなわちuseReadContracts()も呼び出されます。

functionNameのところに呼び出すコントラクトの関数名を、argsにはその引数を入れています。

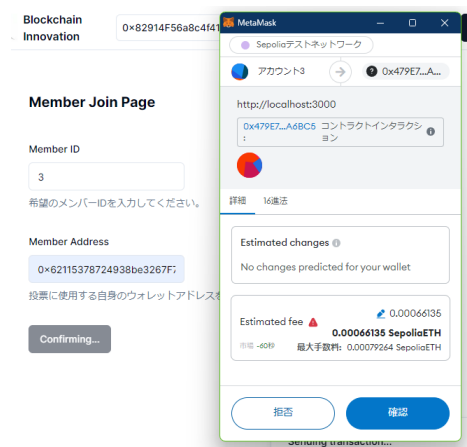
ここでは2つの関数を連続して呼び出しています。getProposalHeaders()とgetNextProposalId()の2つ。前者には引数をpidをBigInt型に変換して入れており、後者の関数には引数がない。もともとのコントラクトでの関数定義は以下：

```
function getProposalHeaders(uint id) external view returns
(Schema.Header[] memory) {}
```

```
function getNextProposalId() external view returns (uint) {}
```


コントラクトで情報を書き込んでみよう

- 今度はwagmi.jsのwriteContract()を使用して、書き込み処理を実行します。
 - この場合は、Txをブロックチェーンに投げる必要があるのでMetaMaskによる署名・Tx作成が必要になります。
- Member Joinのページを見てみましょう：



<http://localhost:3000/memberJoin>

Submit押下時にMetaMaskが起動します。

```
function handleSubmit(data: memberJoinSchemaType) {
  const args = {
    _candidates: {
      id: BigInt(data.candidates.id),
      addr: getAddress(data.candidates.addr),
      metadataURI: toHex("0", { size: 32 }),
    },
  };

  writeContract(
    {
      ...TextDAOFacade,
      functionName: "memberJoin",
      args: [[args._candidates]],
    },
    {
      // 途中省略
    }
  );
}
```

src/app/memberJoin/page.tsx

「Submit」ボタンを押すとこのhandleSubmit()関数が呼ばれます。

その中で、writeContract()関数が実行されています。

ここでは、コントラクトに以下で定義されているmemberJoin()関数を呼び出しています。

引数にはargs._candidatesを入れています。

```
function memberJoin(Schema.Member[] calldata _candidates)
public {}
```

このページで、自分のアカウントを登録してください。

Member IDは任意の数ですが、他の人と重なると登録できません。適当に重ならないような値を入力してください。もし重なった場合はエラーが出るので、その場合は別の値を入れてやり直してください。

コントラクトから状態を取得しよう（演習）

- MembersページはuseReadContracts()の中身が欠けていて実行できません。
- P.35を参考にMember IDをもとにMemberの情報を取得するように修正してください。
- 修正するコードはsrc/app/members/page.tsxの「TODO」とコメントが書いてある箇所です。

正しく表示される例：

Contract Call Page

Get Member Data
指定されたIDのMemberデータをコントラクトから読み取ります。

Member ID

データを取得したいMemberのIDを入力してください。

Call

Member Information
ID: 0
address: 0x5be8259BfdBC77e93158D4AD1B4B0670FEc3f816

<http://localhost:3000/members>

Memberデータの取得は、コントラクトに定義されている以下のgetMember()を使います。

引数には変数「memberID」をわたします。
BigInt型への変換を忘れずに。

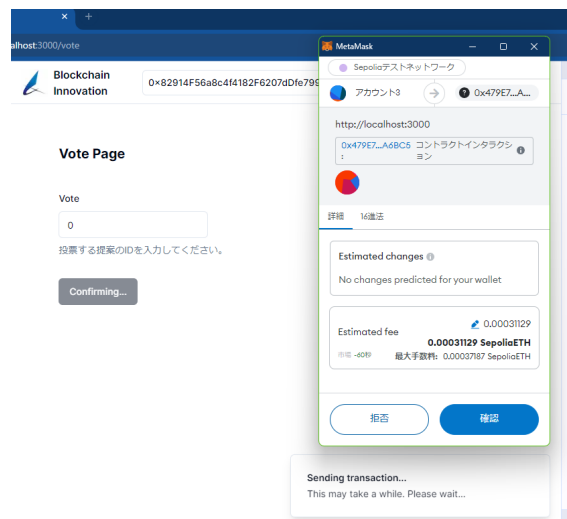
```
function getMember(uint id) external view returns (Schema.Member memory) {}
```

コントラクトで情報を書き込んでみよう(演習)

- voteページはuseReadContracts()の中身が欠けていて実行できません。
- P.36を参考に入力されているPropose IDをもとに投票するように修正してください。
- 修正するコードはsrc/app/vote/page.tsxの「TODO」とコメントが書いてある箇所です。

正しく表示される例：

「Submit」押下後，MetaMaskが起動しTx作成の承認処理へ進みます。



<http://localhost:3000/vote>

投票は，コントラクトに定義されている以下のvoteHeaders()を使います。

引数には変数「`args.proposalId`」と「`args.headerIds`」の2つをわたします。
これは型宣言時にBigintになっているので変換は不要です。

```
function voteHeaders(uint _proposalId, uint[3] calldata _headerIds) public {}
```

一つのアカウトで何回も投票できてしまいます。
すみませんが，1人1票の投票をお願いします。（まちがって2回以上投票してしまっても害はありません）



投票結果の閲覧

- Proposalsページでscoreが増えていくの確認できます。

Proposals

Get Proposal Data

指定されたIDのProposalデータをコントラクトから読み取ります。

Proposal ID

データを取得したいProposalのIDを入力してください。

Call

登録済みのIDは0から
5
です。

Proposal detail:

title: Test 1

score: 10

<http://localhost:3000/proposals>



予備課題

- 自分自身のパソコンのfoundry上ブロックチェーンにTextDAOを実行し，frontpracticeからそこに接続して起動せよ.
 - 1) TextDAOのソースコードをGitHubから取得
 - ヒント) git clone コマンドを使用します.
 - リポジトリ：<https://github.com/blockchaininnovation/TextDAO>
 - 2) foundryでビルド，ローカルの開発用テストネットにデプロイする.
 - ヒント) <https://github.com/blockchaininnovation/TextDAO> や <https://github.com/blockchaininnovation/frontpractice> のREADMEに実行手順が書かれています.
 - ヒント) 「foundry local デプロイ」などで検索すると参考になるサイトが出てきます.
 - 3) frontpracticeのブロックチェーン接続先，コントラクトアドレスの設定を2)のローカルのものに変更する.
 - ヒント) 本資料P.18, 19
 - ヒント) <https://github.com/blockchaininnovation/frontpractice> のREADME



まとめ

- Web3のシステムのデザインについて学んだ.
- フロントエンドのサンプルアプリケーションを用いてコントラクトの処理を呼び出すことを学んだ.
 - データの読み込み処理
 - データの書き込み処理
 - ウォレットでの署名処理 (Tx作成) が必要でガスコストがかかる.
- wagmi.jsを用いてのフロントエンドアプリケーションを動かし, スマートコントラクトを実行させるアプリケーションの外観を知ることができた.



演習の解答

- 今回の演習の解答は同じリポジトリのanswerブランチで確認できます.
- <https://github.com/blockchaininnovation/frontpractice/tree/answer>

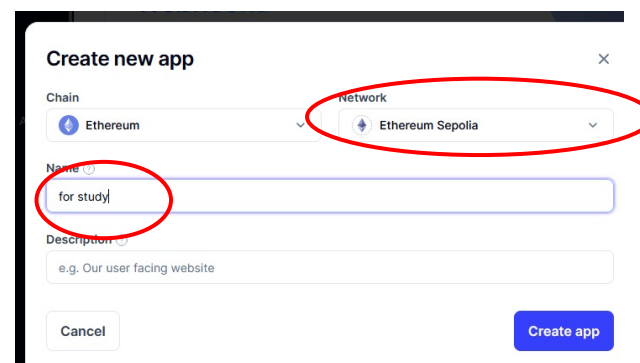
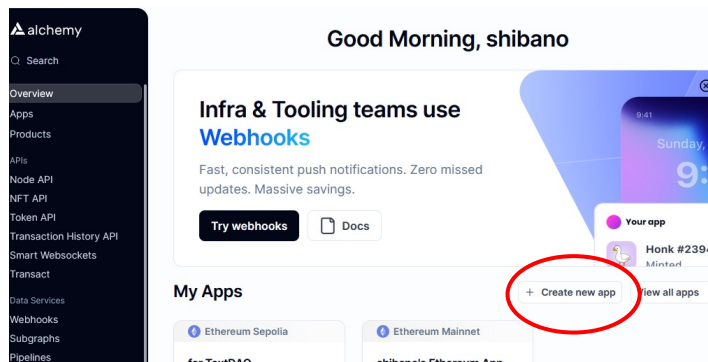


お知らせ・もっと詳しく学習したい方へ

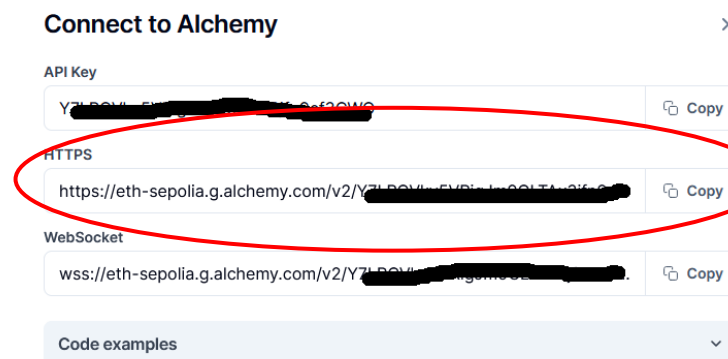
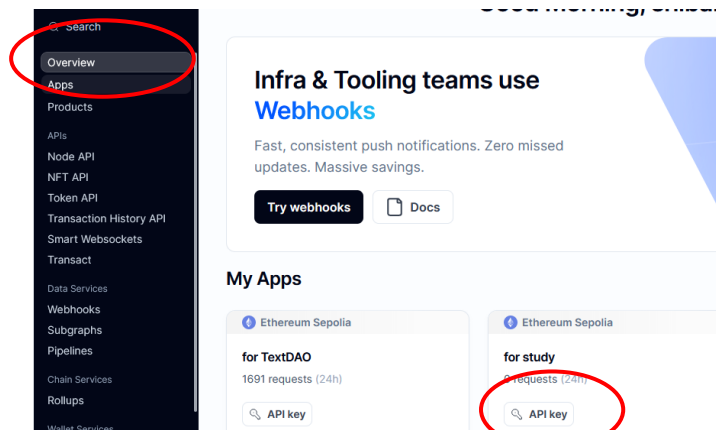
- SparkleAIさんが、今回の講義に関連した内容を詳細に解説してくれるサイドセミナーを開催します。
- 興味のある方は、応募ください。
 - <https://forms.gle/vp8KpHg45sxXoKiK6>
- 対象者
 - ウェブサイト（特にウェブアプリ）の開発経験（商用/ホビー問わない）、あるいは専門学習の経験があり、基本的な開発を理解している方に向けて。
- 日時・募集人数
 - 7月にオンライン＆オフライン同時で1度開催。日取りは現在未定。
 - 人数：20名程度（オンライン10名、オフライン10名）
 - 応募多数の場合は抽選
- コンテンツ
 - 実際のWeb3プロダクトを題材にして、DApps開発の勘所（アーキテクチャー、具体的な技術スタック、TIPS）を紹介。
 - Dapp全体像やアーキテクチャ、React/Next.js/Wagmi.js/viemなど。

API Keyの取得 (Alchemy)

1. アカウント登録をする (<https://www.alchemy.com/> にアクセスし左上の「Sign in」から)
2. Sepolia用Appを作成する (「Create new app」→ Nameを適当に決めて、Networkを「Ethereum Sepolia」に)



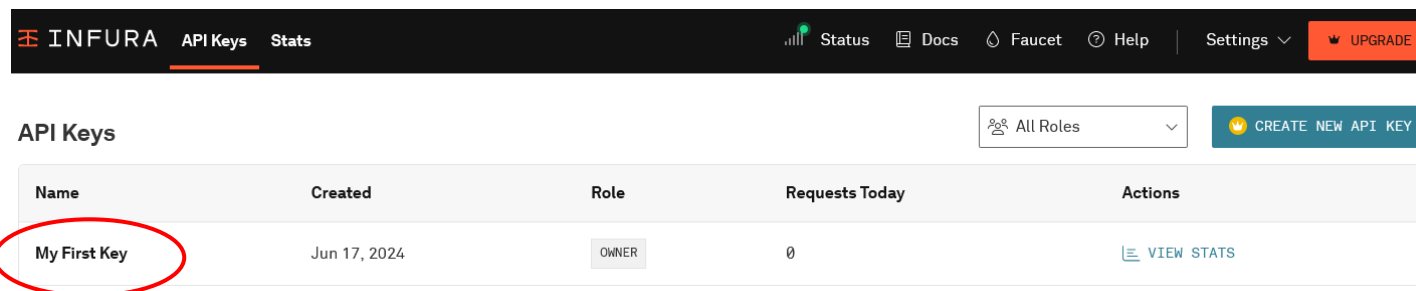
3. 左のメニューからOverviewを開き、今作ったappの「API Key」を押下して表示されるURLを使用します。



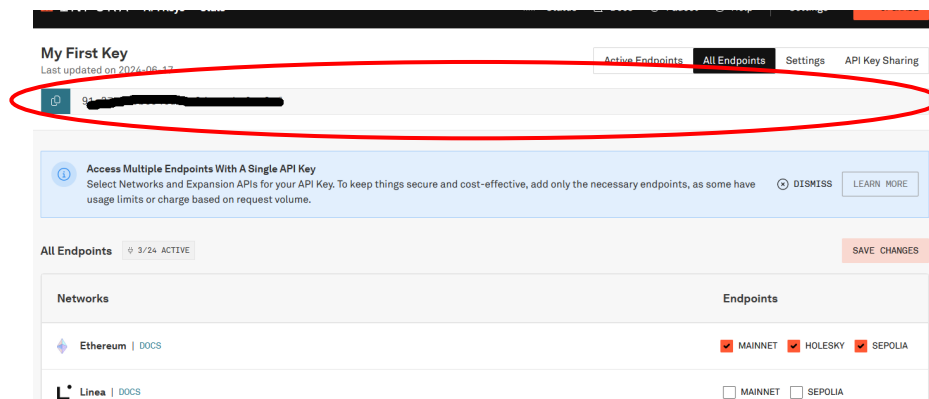
これが表示されていれば
OKです。

API Keyの取得 (infura)

1. アカウント登録をする: <https://www.infura.io/> にアクセスし左上の「Sign in」からアカウントの作成。
アカウント作成中に聞かれる質問はなんでもOKです。プランは「Free」で十分です。
2. ログインできたら「My First Key」ができています。名前をクリックしてください。

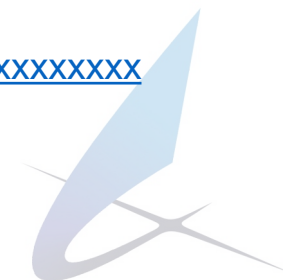


INFURA API Keys Stats				
Status	Docs	Faucet	Help	Settings Settings UPGRADE
API Keys				
All Roles		CREATE NEW API KEY		
Name	Created	Role	Requests Today	Actions
My First Key	Jun 17, 2024	OWNER	0	VIEW STATS



これが表示されていればOKです。当日はこちらのKeyをコピーして以下のURLをエンドポイントとして使用します。

<https://sepolia.infura.io/v3/xxxxxxxxxxx>
(xxxxxxxxxxxがこのキー)



-
- 本スライドの著作権は、東京大学ブロックチェーンイノベーション寄付講座に帰属しています。 自己の学習用途以外の使用、無断転載・改変等は禁止します。