Web Application Development and Blockchain Integration

Questions and Communication

- Use a red question mark emoji in the chat to highlight your questions. This helps the instructor easily spot and address them during the lesson.
- Raising your hand is also effective, but the emoji ensures your question is highly visible.

Transitioning to Web Applications

So far, we've been working with scripts that run locally on our machines. Sharing these applications with others requires sending them the scripts and guiding them through the installation process.

Imagine using a **tokenized ballot** system for your homeowners association. Requiring everyone to install **Node.js** and run scripts from their computers would be impractical.

This week, we'll focus on building **web interfaces** for blockchain interaction:

- Creating web applications $oranyapp$ to easily communicate with the blockchain through a clickable interface.
- Building **back ends** for communication between the front end and the blockchain.
- Using **oracles** to fetch off-chain data and integrate it into the blockchain.

Full-Stack Week Overview

This week will cover applications and data, including:

- Building interfaces
- Creating back ends for interface communication
- Direct back end to blockchain communication
- Back end data provision to the blockchain
- Front end data consumption

  Note: This is a full-stack overview, but becoming a full-stack developer in one week is unlikely. If you already know front-end and back-end development, you will learn how to integrate them with the blockchain. If you are new to front-end or back-end development, you should be able to replicate the examples.

Setting Expectations

- Experienced Developers: You will learn how to integrate your existing front-end and back-end skills with blockchain technology, especially if you use JavaScript-based tools.
- Beginners: You should be able to replicate the examples provided, with step-by-step guidance. Using AI tools can also help.

What is a Web Application?

A web application is a website (like Wikipedia) that you can access through an internet browser without needing to install anything. The browser handles the interaction between the web page and your system, managing button clicks, events, and file uploads.

Client-Side Considerations

When you interact with a web application, like deleting an advertisement banner on Wikipedia, the change only affects your view:

- This is because you are modifying the HTML on the client side.
- Your actions do not affect other users or the server's original content.

When integrating blockchain with front ends, remember:

- Everything on the front end is visible and modifiable by the user.
- Critical checks $like balance verification$ must occur on the blockchain itself, not just on the front end.

Frameworks and Technologies

We will use frameworks to avoid coding HTML, CSS, and JavaScript directly. Browsers understand these languages, which are used to create responsive and styled web pages.

Front End Development Frameworks

Front End Development Tasks

- Smart contract developers may need to implement contract features on the front end.
- A user-friendly interface abstracts away technical complexities for end users.

Framework Analogy: Ordering a Cake

- Ordering a cake is analogous to using frameworks: you provide high-level instructions $flavor, size, toppings$ without micromanaging details $sugar, eggs, flour$.
- Similarly, with frameworks, you specify high-level components $top pane, middle pane, bottom pane$ and the framework handles the HTML, CSS, and JavaScript.

Framework Capabilities

- Frameworks handle:
  - HTML building
  - Responsiveness
  - CSS
  - JavaScript
  - Dynamic content population
  - Pagination
  - Routing
  - State management
  - Blockchain connections
  - Authentication

Popular Frameworks

- Top front end development frameworks:
  - React
    - A library, not a framework

      React itself is a Javascript library that requires developers to understand what to import and how to use it, making it less beginner-friendly compared to full frameworks.
  - Angular
  - Vue.js
- Styling Frameworks:
  - Tailwind
  - Bootstrap
  - Material CSS
  - They aim to streamline development by producing usable HTML, JavaScript, and CSS.

Framework Selection: React and Next.js

- React is not beginner-friendly; Next.js is a framework built on React.
- Next.js prioritizes performance, optimization, and security over beginner-friendliness.
- Web3 SDKs and applications often assume the use of React and Next.js.
- The boot camp will focus on popular technologies like Next.js due to its prevalence in Web3 development.
- Angular is great for enterprise according to the professor.## Web Development Frameworks

Popularity and Support

Using a popular framework can be very helpful because:

- There is more support and documentation available.
- You'll find more engagement on forums.
- AI models are trained more on popular frameworks.

Framework Experience

- The professor has experience with web development frameworks outside of JavaScript.
  - Python $Django, Flask$
  - .NET Core
  - PHP $CakePHP, Laravel$

Challenges with Less Popular Frameworks

- Using less popular frameworks for blockchain development can be challenging.
  - Lack of documentation and support
  - Outdated or unmaintained libraries
  - Painful development experience

Recommendation

It's suggested to use the most popular frameworks for blockchain development to make your life easier.

Scaffold-ETH Package

Overview

We're going to implement the app using a package called Scaffold-ETH, which comes preconfigured out of the box.

Key Features

- Next.js: Already configured and installed.
- Wallet Integration: Necessary integrations for connection with the wallet.
- Burn Wallet:

  Helpful for front-end development. Solves problems related to resetting Metamask and clearing the cache.

- Components: Includes components and integrations built with Typescript.
  - Next.js: Framework for building the UI.
  - Rainbow Kit & Wallet Connect: Allow direct connection to the blockchain.
  - Foundry & Hardhat: Backends for building smart contracts.
  - ABIs & Viem: Used for smart contract calls and integrations.

Functionality

- Auto Reload: Contracts auto-reload.
- Front-End Generation: Generates the front-end based on the smart contracts.
- Hooks: Application feels responsive without refreshing the page.
- Components: Includes components for displaying addresses and balances.

Interface

The interface will be built with Next.js and will integrate directly with the smart contract being built.

Technologies and Tools

Overview

We're going to see what Wagmi, Tailwind, Daisy UI, and Rainbow Kit are doing, how to use Scaffold-ETH, and how to write JSX.

Wagmi

- **Role:** Handles the connection with the blockchain.
- **Functionality:**

  > Wagmi is a wrapper that solves problems React created. It uses Viem in the backend to get the balance of an address.

- **Hooks:** Wagmi hooks for React (e.g., `useBalance`) use Viem under the hood.

Question

Why choose this stack?

Scaffold-ETH as a Boilerplate

- **Best Starting Point:** Scaffold-ETH is one of the best starting projects for building a dApp.
- **Completeness:** It is one of the most complete and used stacks.

Styling with Tailwind

- **Integration:** Tailwind is very well-styled.
- **Functionality:**

  > You can add classes to HTML items to style them.

- **Benefits:** Makes it easy to create a presentable UI.

Daisy UI

- **Role:** UI library that complements Tailwind.
- **Functionality:** Provides pre-built components like buttons.
- **Benefits:** Saves time by providing ready-to-use, styled components.

Daisy UI vs. Material UI

- **Daisy UI:** Works with Tailwind.
- **Material UI:** Replacement for Bootstrap material.

Getting Started with Scaffold-ETH

Yarn Package Manager

- **Scaffold-ETH** is organized using **yarn workspaces**, combining frontend and backend within a project.
- **Yarn** is required for running the project.

Installation Steps

1. **Install Yarn** $ifnotalreadyinstalled$**:**

```
corepack enable
```

2. **Clone Scaffold-ETH:**

```
git clone -b yarn4.7 https://github.com/scaffold-eth/scaffold-eth.git
```

3. **Navigate to the Scaffold-ETH directory:**

```
cd dapp
```

4. **Install project dependencies:**

```
yarn install
```

Yarn vs. NPM

Feature	Yarn	NPM
Speed	Faster due to parallel execution and caching.	Slower compared to Yarn.
Global Install	Avoids global installations for recent versions.	Supports global installations.
Compatibility	Can have compatibility issues due to assumptions about npm's `node_modules`.	More universally compatible.
Plug and Play	Offers "Plug and Play" for faster project setup but may cause compatibility issues.	N/A

Project Structure

- Scaffold-ETH uses yarn workspaces to manage multiple projects within a single repository.
- Key directories:
  - `packages`: Contains separate projects, including `hardhat` and a `nextjs` application.

Running the Project

- Use `yarn start` to start the Next.js application in development mode.
  - This command utilizes yarn workspaces to build the application.
  - Compiles high-level code into working HTML, JavaScript, and CSS.
- Use `yarn test` to run tests in the Hardhat workspace.

Security Considerations

Be cautious about installing packages, as they can potentially be infected. Use separate environments for development and sensitive activities like crypto trading. Be wary of unsolicited requests to install software, as they may be scams.

Yarn Workspace Command

- Scaffold-ETH is designed as one project containing multiple sub-projects managed via yarn workspaces.
- Example commands:
  - `yarn start`: Starts the Next.js frontend.
  - `yarn test`: Runs tests using Hardhat.

Compilation Process

Compilation is the process of translating high-level topics in the page for the framework, and the framework translates all of this to actually working HTML, JavaScript and CSS.

Development Environment Setup

Starting the Application

The `yarn start` command utilizes the Next.js framework to launch a local server, making the application accessible through a web browser at `localhost:3000`. This is specifically for development, allowing interaction with the application on your local machine.

Addressing Initial Errors

Upon accessing the application, you might encounter an error message indicating "No local chain running yet" or "HTTP request failed." This occurs because the application attempts to connect to a local Hardhat network by default.

The configuration for the target networks is located in the `packages/nextjs/scaffold.config.ts` file, specifically around line 12.

```
// Example from packages/nextjs/scaffold.config.ts
// targetNetworks: [hardhat, mainnet]
```

When developing a decentralized application $dApp$ for production, it should connect to a public testnet or mainnet. However, for development and learning, a local chain like Hardhat is used.

Running a Local Blockchain

To resolve the connection error and proceed with local development, run the Hardhat chain continuously in a separate terminal window. This hosts a local blockchain on your computer.

Unlike running tests or scripts, where the blockchain is started and killed after execution, here, it's kept running.

To start the Hardhat chain, use the command in a separate terminal:

```
yarn chain
```

This command builds a lightweight node of a working Hardhat blockchain, closely resembling the Ethereum blockchain, providing accounts prefunded with 10,000 ETH. Note that these aren't real ETH.

Once the Hardhat chain is running, refresh the web page at `localhost:3000` to establish the connection with the local network.

After connecting to the local network, you can connect with a wallet, such as Metamask or a burn wallet, to interact with the application.

Alternative Approach $Not Recommended$

An alternative, but less efficient, approach would be to modify the `packages/nextjs/scaffold.config.ts` file to target a public testnet like Sepolia. However, this would significantly slow down the development process due to the need for testnet tokens and dealing with blockchain block times.

Troubleshooting Common Issues

- Hydration Failed Error: If you encounter a "console error hydration failed" message, try a hard refresh of the page by pressing Ctrl+Shift+R or Ctrl+F5 to clear the cache.
- Yarn Start Issues: If the `yarn start` command fails, verify that you're in the correct directory and that the `yarn.lock` file exists.
- Metamask Connection Problems: If you can't reach `localhost:3000` after connecting Metamask, ensure the front end and blockchain are running concurrently in separate terminals. If issues persist, restart both.

Private Keys

Private keys for the prefunded accounts are available when running the local Hardhat blockchain. However, it is not recommended to use these private keys on public testnets.

Troubleshooting Wallet Connection Issues

- If you're having trouble connecting your wallet, ensure you're using the correct network settings.
- If the page seems stuck, it's normal for the compilation process to take a minute or so.
- Yarn start should run continuously; avoid interacting with that terminal tab directly. Open a new tab for other commands.
- You should manually open `localhost:3000` in your browser to access the application.

Interacting with the Blockchain

- When you connect your wallet, you're interacting with a blockchain.
- Initially, you won't have any funds.
- Use the faucet to grab funds for testing. This will send transactions to your burner wallet's address.
- You can monitor these transactions in the terminal running `yarn chain`.
- The block explorer should display these transactions, showing the money being "dripped" into your account.

Questions and Answers

Metamask and Yarn

- Question: Do you need to run `yarn chain` to connect to Metamask?
- Answer: Yes, `yarn chain` is essential to connect to the local network with Metamask or Burner Wallet.

Transactions

- Question: How are transactions being done?
- Answer: By clicking the button to get money from the faucet. The inner workings will be explained later.

Yarn vs. NPM/NPX

- Question: Is Yarn necessary, or can NPM/NPX be used with Scaffold-ETH?
- Answer: You need Yarn. The steps involve enabling Yarn, cloning, installing, and starting.

Common Yarn Issues

- Node version issues may arise with the first command (`yarn create @scaffold-eth/nextjs`).
- Using Node version 18 or above should resolve this.
- Do not install `corack` or `npm` globally, as this can cause version conflicts.
- If facing issues with `corack` directly, check your Node installation.

Permission Denied Errors

- Permission issues, especially on macOS, can be tricky.
- Using Node Version Manager NVM can help manage permissions correctly.

Scaffold-ETH as a Tool

- Scaffold-ETH is a valuable tool for managing local connections, similar to Ganache from Truffle.
- It provides a starting point for local blockchain development.
- Even before diving into front-end development, it helps in interacting with your local blockchain.

Burner Wallet Issues

- If you don't see the Burner Wallet option, ensure you're connected to the local testnet.
- Changing networks might cause the Burner Wallet to disappear.

Emphasis on Local Development

- Focus on local development for now.
- Run `yarn chain` and use the local chain for all commits.
- Avoid public testnets for the time being.

Local Blockchain Development

- We're using a **local blockchain** for development to avoid the slowness of public networks.
- To run the local chain, use the command `yarn deploy`.
    - This picks a sample smart contract and runs a sample deployment script.
    - It uses a **pre-funded account** to deploy the contract (`yourContract.sol`).
- The deployment script is located in the `deploy` folder and uses deployments from **Hardhat**.
- The script deploys the smart contract to the local network that you have started. This network is initially empty.

Deploying Smart Contracts

- Whenever you build a new smart contract in `scaffold-eth`, you need to call your **deployment script**.
- The deployment script uses a library $like ethers$ to abstract the connection to the blockchain.
- It uses **Hardhat** and passes a **deployer** as an argument for the contract deployment.
- After running the `yarn deploy` command, you'll see logs on your chain indicating the contract deployment.
- The script may also include a call to the smart contract for initial greetings.

Debugging Contracts

- The "Debugging Contracts" page automatically tracks all deployments.
- It picks a list of contracts deployed with the current deployment package.
- It shows an interface to debug any smart contract.

Creating a New Smart Contract: Hello World

1. Create a new file called `hello.sol` inside the `packages/hardhat/contracts` folder.

2. Add the Hello World contract code. This can be copied from previous lessons.

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HelloWorld {
    string public greeting;

    constructor() {
        greeting = "Hello, World!";
    }

    function setGreeting(string memory _greeting) public {
        greeting = _greeting;
    }

    function getGreeting() public view returns (string memory) {
        return greeting;
    }
}
```

3. Create a deployment script `deploy-hello.ts` in the `deploy` folder.

4. Copy the structure from `deploy-contracts.ts`.

```
// deploy/deploy-hello.ts
import { HardhatRuntimeEnvironment } from 'hardhat/types';
import { DeployFunction } from 'hardhat-deploy/types';

const deployHello: DeployFunction = async function (hre: HardhatRunti
    const { deployer } = await hre.getNamedAccounts();
    const { deploy } = hre.deployments;

    await deploy('HelloWorld', {
        from: deployer,
        args: [], // constructor arguments
        log: true,
    });
};

export default deployHello;

deployHello.tags = ['HelloWorld'];
```

5. In the deployment script, you'll:

    - Get a deployer.
    - Deploy based on the artifact name.
    - This includes the contract in the blockchain.

6. You can create one deployment script for each smart contract, or one script for all.

Smart Contract Registry

- After running the deployment script, it keeps track of deployed smart contracts.
- It deploys only the new ones and updates the registry.
- Existing contracts are reused, not redeployed.
- The "Debugging Contracts" page automatically updates based on the deployed smart contracts.
- You can list tokenized ballots, ballots, and other contracts in the interface.

Updating Smart Contracts

- To update a smart contract $e.g., addinganewfunction$:
    1. Stop your blockchain.
    2. Start your chain again.
    3. Redeploy your contracts.
- Starting the chain fresh clears all contracts.
- The deploy script will detect that deployments are needed and redeploy the new versions, updating the contract registry.

Viewing Transactions on Chain

- Transactions, including contract deployments and calls, can be seen on the chain.
- By going to the "Debugging Contracts" page, you can see the deployed contracts listed.

Getting Gas Money

If you ever run into an issue where you "can't pay for the transaction," it means you need to grab money from the faucet.

- Click the faucet button in the top right pane.
- Alternatively, use the faucet at the bottom to pick the destination and send directly to that address.

The Cake Metaphor: Next.js and page.TSX

The **page.TSX** file in a Next.js application is where you specify what you want in your "cake." Using the analogy of baking a cake, the page.TSX file specifies the top, middle, and bottom elements of your webpage.

- This file mixes **TypeScript** and **HTML**, which is characteristic of React.
- It includes a **return statement** that returns HTML.

Home Screen Component

When you access the application's homepage, the framework handles all the routes for you.

- The header and footer remain consistent.

- The content within the home component can be edited directly, with changes reflected in real-time.

> Abstract away the complexities of React and focus on how modifications to the code $e.g., addingaparagraphorbutton$ directly translate to the rendered output on the page.

Scaffold-ETH Components

Initially, the lecture uses Scaffold-ETH components to expedite development. Later lessons will cover integrations done directly, without relying on Scaffold-ETH. This will include using the Scaffold-ETH project structure but not directly importing components from it.

- The lecture demonstrates using a Scaffold-ETH read contract hook to perform a smart contract read on the local test network, similar to previous exercises with Hardhat.

- This hook fetches information from the smart contract and displays it on the page.

  `ExampleContractRead` gets the contract by name, the function by name, and displays information based on what it gets.

  For example: changing some data within the Bug contract, and then displaying it on the homepage component. $ `The text from the contract is: This is a test text` $

Wallet Kit Integration

The integration of the wallet kit facilitates communication with the blockchain.

- Wallet Kit: A package used by Scaffold-ETH to handle all communication with the blockchain, including the wallet connection model.
- By forking and cloning Scaffold-ETH, you have access to all integrated components.
- Previously known as Wallet Connect.

Homework Assignment: Wallet Kit Configuration

Configure your wallet kit using your own keys instead of Scaffold-ETH's direct integration.

1. Locate the `scaffold.config.ts` file (or `scaffold.config.js`) in the `next.js` folder.
2. Explore the provided links to understand the wallet kit.
3. Create a real cloud and obtain a project ID to input into your environment.
4. For issues or questions, open a GitHub issue or reach out on Discord.