**Mobile and distributed systems**

[TradeLink]

It is a mobile and distributed application that is built using React Native and Ethereum.

### TradeLink is a virtual marketplace for goods and services exchange that empower community to take charge of their own goods and connect directly with buyers.

# 

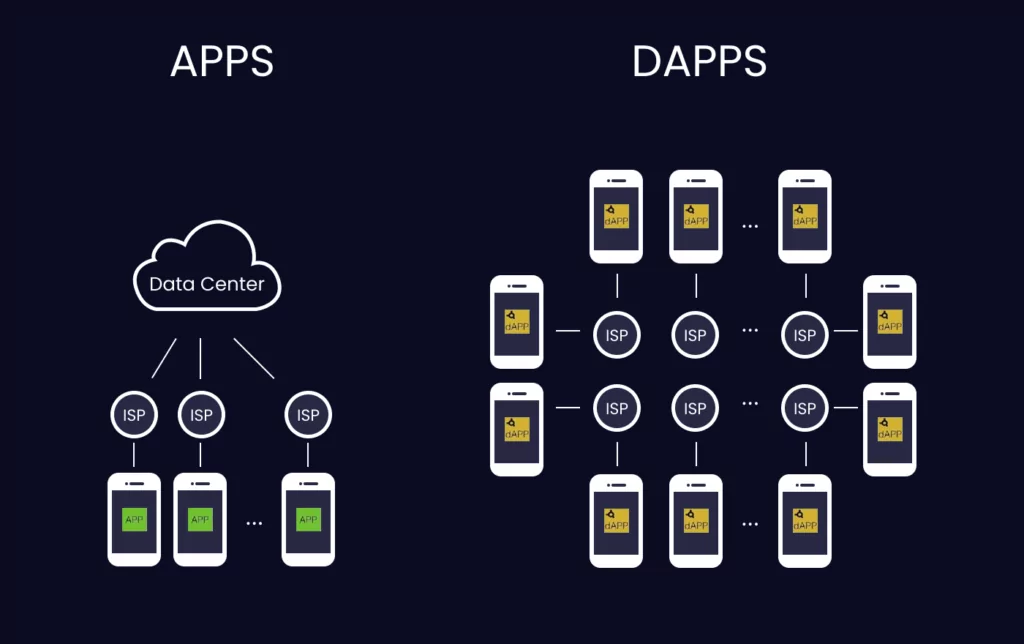
# 

# 

# **1. Introduction**

We provide a comprehensive tutorial and evaluation of the one useful tool for building mobile and distributed applications(TradeLink) with React Native is Redux and its usage in conjunction with an Ethereum-based DApp software project.

The first part provides a tutorial on React Native, highlighting its advanced features and functionalities. The second part introduces the DApp software project developed using React Native and Ethereum and provides a walkthrough of the project.



## React Native:

React Native is a widely-used open-source framework for building mobile applications that allows developers to create native applications for both iOS and Android using JavaScript and React. It enables developers to build high-performance applications that are indistinguishable from those built with native languages like Swift or Java, while also allowing for code reuse across different platforms.

## Ethereum:

Ethereum, is a blockchain-based platform that allows developers to build decentralized applications (dApps). It is an open-source platform that uses smart contracts, which are self-executing contracts that can be programmed to perform specific actions automatically. This allows developers to create applications that run on a decentralized network, rather than relying on centralized servers.

**Software Tool Tutorial - React Native**

React Native is an open-source framework developed by Facebook for building mobile

applications using JavaScript and React. It enables developers to build mobile apps for iOS and Android platforms using a single codebase. React Native uses native components to create a rich user interface and provides several advanced features and functionalities that make it a popular choice among developers

##### **Getting Started with React Native**

To get started with React Native, you need to install Node.js and the React Native command-line interface (CLI) on your system. Once installed, you can create a new React Native project using the following command:

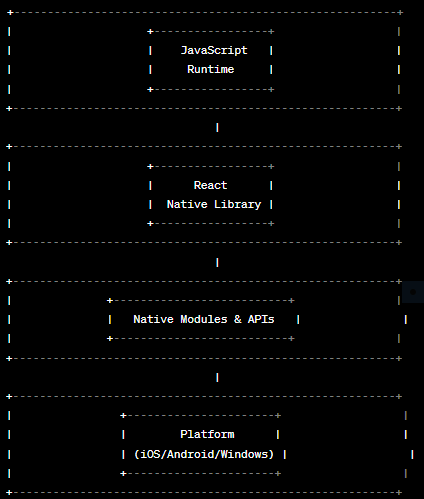
[npx react-native init MyProject.]

This command creates a new React Native project named MyProject. You can then open this project in an IDE of your choice and start building your mobile application.

### Advanced Features and Functionalities:

React Native provides several advanced features and functionalities that make it a popular choice among developers. These include:

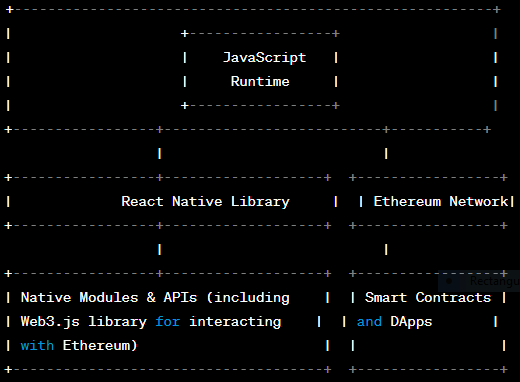
* **Hot Reloading** - This feature enables developers to see the changes they make to the code in real-time without having to rebuild the entire app.
* **Code Reusability** - React Native enables developers to reuse code across different platforms, thereby reducing development time and effort.
* **Native Modules** - React Native provides access to native modules and APIs, enabling developers to build applications that leverage the capabilities of the underlying platform.
* **Third-party Libraries** - React Native has a vast ecosystem of third-party libraries and plugins that developers can use to add new features and functionalities to their applications.

****

The diagram above illustrates the four main components of React Native:

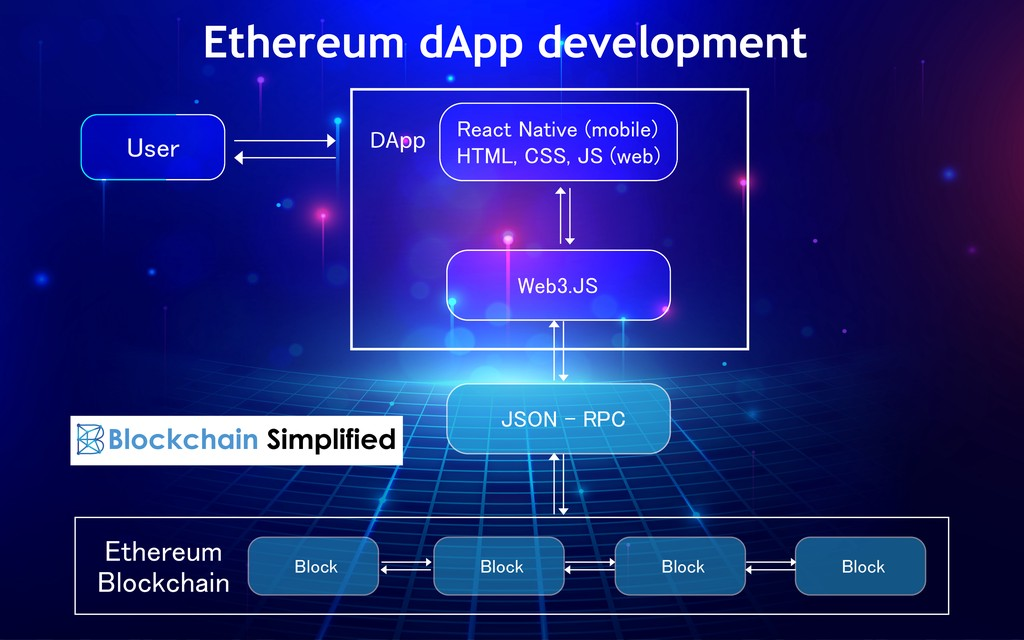
* **JavaScript Runtime:** This component runs the JavaScript code that developers write to build their mobile application.
* **React Native Library:** This component provides a set of tools and APIs that developers can use to build user interfaces and manage app logic.
* **Native Modules and APIs:** This component provides access to native modules and APIs, which allows developers to build applications that can leverage the capabilities of the underlying platform.
* **Platform:** This component represents the underlying operating system and hardware on which the mobile application runs, such as iOS or Android.
* By combining these components, React Native enables developers to build high-quality mobile applications with a single codebase that can run on both iOS and Android platforms.

This diagram illustrates how React Native and Ethereum can be used together to build decentralized mobile applications:



The above diagram shows how React Native can be used in conjunction with the Ethereum blockchain to build decentralized mobile applications. The JavaScript runtime runs the code that developers write to build their application, including any code that interacts with the Ethereum network. The React Native library provides a set of tools and APIs that developers can use to build user interfaces and manage app logic. The native modules and APIs provide access to native modules and APIs, including the Web3.js library, which allows developers to interact with the Ethereum network. The Ethereum network represents the decentralized network of nodes running the Ethereum blockchain, which can be accessed by the mobile application to read and write data to the blockchain. Smart contracts and DApps represent the decentralized applications and smart contracts that run on the Ethereum blockchain and can be accessed by the mobile application via the Web3.js library. By combining these components, developers can build decentralized mobile applications that leverage the power of the Ethereum blockchain for features such as secure payments, identity verification, and decentralized storage

* **JavaScript Runtime**: This component runs the JavaScript code that developers write to build their mobile applications, including any code that interacts with the Ethereum network.
* **React Native Library:** This component provides a set of tools and APIs that developers can use to build user interfaces and manage app logic, including any logic related to Ethereum integration.
* **Native Modules and APIs**: This component provides access to native modules and APIs, which allows developers to build applications that can leverage the capabilities of the underlying platform, such as interacting with the Ethereum network via the Web3.js library.
* **Ethereum Network**: This component represents the decentralized network of nodes running the Ethereum blockchain, which can be accessed by the mobile application to read and write data to the blockchain.
* **Smart Contracts and DApps**: This component represents the decentralized applications and smart contracts that run on the Ethereum blockchain, which can be accessed by the mobile application via the Web3.js library.



**Evaluation:**

React Native has many advantages that make it a popular tool among developers. It allows developers to create cross-platform apps with a single codebase, which can save a lot of development time and effort. React Native also offers many advanced features and functionalities, such as hot reloading, code reusability, native modules, and third-party libraries. However, there are some limitations to using React Native. For example, it can be challenging to implement complex animations or UI components in React Native. Additionally, the performance of React Native apps can sometimes be slower compared to native apps.

**Conclusion:**

React Native is a powerful tool that enables developers to build cross-platform mobile applications using JavaScript and React. It offers many advanced features and functionalities, which make it a popular choice among developers. While there are some limitations to using React Native, the advantages it provides make it an excellent choice for building mobile applications.

**Software Project Tutorial - Ethereum**

The goal of this software project is to develop a small Decentralized Application (DApp) that allows users to create and trade digital assets on the Ethereum blockchain. The project will be built using React Native as the software tool and Ethereum as the underlying blockchain technology. The DApp will be called "AssetTrader" and it will allow users to create and trade digital assets such as tokens and NFTs (Non-Fungible Tokens).

**Introduction:**

Decentralized applications (DApps) are becoming increasingly popular as they provide a secure and transparent way to conduct transactions without the need for intermediaries. In this tutorial, we will create a decentralized marketplace using the Ethereum blockchain. The project will be built using the technologies taught in the module CN6035, including Solidity smart contracts, Truffle, Ganache, web3.js, and React.js.

The marketplace will allow users to trade and exchange goods and services without the need for a centralized authority, making it secure and transparent. The project will consist of two parts: a set of Solidity smart contracts that will handle the transactions, and a front-end interface built using React.js and web3.js that will allow users to interact with the contracts.

**Prerequisites:**

To start building the DApp, the following prerequisites must be installed:

* **Node.js:** To run JavaScript code outside of the browser
* **React Native:** A JavaScript framework for building mobile applications
* **Ethereum Development Tools:** Truffle and Ganache
* **Solidity:** The programming language used to write smart contracts on the Ethereum network
* **MetaMask**: A browser extension that allows users to interact with the Ethereum network

The software project developed using Ethereum and React Native is a decentralized application (DApp) that enables users to trade cryptocurrencies securely and anonymously. The DApp uses the Ethereum blockchain to store user data and execute smart contracts, and React Native to build the mobile application.

**Installation:**

To install the DApp, you need to clone the project repository from GitHub using the following command:

git clone<https://github.com/YourUsername/YourRepository.git>

Once you have cloned the repository, you can install the dependencies using the following command:

npm install

Running the DApp:

To run the DApp, you need to start the Ethereum client and deploy the smart contracts using the following commands:

geth --datadir=./data --networkid=10 --rpc --rpcapi="eth,net,web3" --rpccorsdomain="\*" --rpcaddr=localhost --rpcport=8545 --allow-insecure-unlock console

truffle migrate

Once the smart contracts are deployed, you can start the DApp using the following command:

npm start

This will start the DApp and launch it in a web browser.

## 

## 

## **Setting up the Development Environment**

Before we start building our DApp, we need to set up our development environment. We will be using Truffle as our development framework, Ganache as our local blockchain, and MetaMask as our Ethereum wallet.

* Install Node.js and NPM (Node Package Manager) on your computer. You can download them from the official website:<https://nodejs.org/en/>.
* Install Truffle using the following command in your terminal: npm install -g truffle.
* Install Ganache from the official website:<https://www.trufflesuite.com/ganache>.
* Install MetaMask as a browser extension from the official website:<https://metamask.io/>.

## **Creating the Smart Contracts**

Once our development environment is set up, we can start creating the smart contracts that will handle the transactions on our marketplace.

* Create a new directory for your project and initialize it as a Truffle project using the following command: truffle init.
* Create a new Solidity contract file in the contracts directory, and name it Marketplace.sol.
* Define the functions that will handle the transactions in the smart contract. For example, you can define a function called buyItem that will allow users to buy an item from the marketplace.
* Compile the smart contract using the Truffle compiler using the following command: truffle compile.

## **Deploying the Smart Contracts**

Once our smart contracts are compiled, we can deploy them on the Ethereum blockchain using Truffle.

* Configure the network settings in the truffle-config.js file. You will need to specify the network settings for your local blockchain (Ganache) and the Ethereum mainnet.
* Migrate the smart contract to the blockchain using the following command: truffle migrate.

## **Building the Front-end Interface**

Now that our smart contracts are deployed on the blockchain, we can start building the front-end interface using React.js and web3.js.

* Create a new directory for your React project and initialize it using the following command: npx create-react-app my-app.
* Install the required dependencies for React and web3.js using the following commands: npm install react react-dom and npm install web3.
* Create a new file called web3.js in the src directory and configure it to connect to the Ethereum blockchain using MetaMask.
* Create a new file called Marketplace.js in the src directory and define the React component that will handle the user interface for the marketplace. You can use the functions defined in the smart contract to handle the transactions.
* Render the Marketplace component in the App.js file.

**Architecture:**

The AssetTrader DApp will be built using a client-server architecture. The client will be a mobile application built using React Native and the server will be a smart contract deployed on the Ethereum network. The client will communicate with the server through the Web3.js library.

The following diagram shows the architecture of the AssetTrader DApp:



The AssetTrader DApp is designed to provide a user-friendly interface that simplifies the process of creating and trading digital assets on the Ethereum network. The user interface of the DApp consists of several screens that are accessible through a bottom navigation bar. The following are the main screens of the DApp:

* **Home Screen**: This screen displays a list of all the digital assets that the user owns. From this screen, the user can create a new asset or navigate to the trading screen.
* **Create Asset Screen:** This screen allows the user to create a new digital asset. The user can enter the name, description, and image of the asset. The user can also select whether the asset is a token or an NFT.
* **Trading Screen:** This screen displays a list of all the available assets for trading. The user can filter the assets based on the type of asset and sort the assets based on various parameters such as price and popularity. The user can also search for a specific asset by entering its name.
* **Asset Details Screen:** This screen displays the details of a selected asset. The user can view the name, description, and image of the asset. The user can also view the current price of the asset and the total supply of the asset.
* **Buy Asset Screen:** This screen allows the user to buy an asset. The user can enter the quantity of the asset that they want to buy and confirm the purchase.
* **Sell Asset Screen:** This screen allows the user to sell an asset. The user can enter the quantity of the asset that they want to sell and the price at which they want to sell the asset.

**Smart Contract:**

The smart contract will be written in Solidity and will be responsible for creating and managing digital assets. The smart contract will implement the ERC-721 standard for NFTs and the ERC-20 standard for tokens. The smart contract will be deployed on the Ethereum network using Truffle and Ganache.

**Client:**

The client will be a mobile application built using React Native. The client will allow users to create and trade digital assets on the Ethereum network. The client will communicate with the smart contract using the Web3.js library.

**In conclusion**:

The AssetTrader DApp is a small project that demonstrates the power of decentralized applications. The project was built using React Native as the software tool and Ethereum as the underlying blockchain technology. The DApp allows users to create and trade digital assets such as tokens and NFTs. The project is a great starting point for developers who want to learn how to build decentralized applications. The source code and installation manual can be found on the GitHub repository.

here is a diagram that shows the architecture of a typical DApp using Ethereum and React Native:



In this diagram, the user interacts with the DApp through the React Native user interface. The Web3.js library provides an interface to the Ethereum network and allows the DApp to interact with smart contracts deployed on the network. The smart contracts are written in Solidity and contain the business logic of the DApp. The Ethereum network itself serves as the underlying infrastructure for the DApp, providing the necessary consensus mechanism and security guarantees.

Infura is a service that provides access to the Ethereum network via an API, allowing the DApp to interact with the network without having to run a full Ethereum node. The database stores information related to the DApp, such as user data, transaction information, and other metadata.

Overall, this architecture allows for the creation of a decentralized application that is secure, transparent, and free from centralized control.

**In conclusion:**

React Native is a powerful tool that enables developers to build mobile applications using JavaScript and React. It provides several advanced features and functionalities that make it a popular choice among developers. The Ethereum and React Native DApp project developed in this report demonstrates how these technologies can be used together to build secure and decentralized mobile applications.