---

title: Kickstart your dapp frontend development with create-eth-app description: An overview of how to use create-eth-app and its features author: "Markus Waas" tags: ["create-eth-app", "frontend", "javascript", "ethers.js", "the graph", "defi"] skill: beginner lang: en published: 2020-04-27 source: soliditydeveloper.com sourceUrl: https://soliditydeveloper.com/create-eth-app

---

Last time we looked at the big picture of Solidity and already mentioned the create-eth-app. Now you will find out how to use it, what features are integrated and additional ideas on how to expand on it. Started by Paul Razvan Berg, the founder of Sablier, this app will kickstart your frontend development and comes with several optional integrations to choose from.

Installation {#installation}

The installation requires Yarn 0.25 or higher (`npm install yarn --global`). It is as simple as running:

```
bash yarn create eth-app my-eth-app cd my-eth-app yarn react-app:start
```

It is using create-react-app under the hood. To see your app, open `http://localhost:3000/`. When you're ready to deploy to production, create a minified bundle with yarn build. One easy way to host this would be Netlify. You can create a GitHub repo, add it to Netlify, setup the build command and you are finished! Your app will be hosted and usable for everyone. And all of it free of charge.

Features {#features}

React & create-react-app {#react--create-react-app}

First of all the heart of the app: React and all the additional features coming with *create-react-app*. Only using this is a great option if you do not want to integrate Ethereum. React itself makes building interactive UI's really easy. It may not be as beginner-friendly as Vue, but it is still mostly used, has more features and most importantly thousands of additional libraries to choose from. The *create-react-app* makes it really easy to start with it as well and includes:

- React, JSX, ES6, TypeScript, Flow syntax support.
- Language extras beyond ES6 like the object spread operator.
- Autoprefixed CSS, so you don't need -webkit- or other prefixes.
- A fast interactive unit test runner with built-in support for coverage reporting.
- A live development server that warns about common mistakes.
- A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps.

The *create-eth-app* in particular is making use of the new hooks effects. A method to write powerful, yet very small so-called functional components. See below section about Apollo for how they are used in *create-eth-app*.

Yarn Workspaces {#yarn-workspaces}

Yarn Workspaces allow you to have multiple packages, but being able to manage them all from the root folder and installing dependencies for all at once using `yarn install`. This especially makes sense for smaller additional packages like smart contracts addresses/ABI management (the information about where you deployed which smart contracts and how to communicate with them) or the graph integration, both part of `create-eth-app`.

ethers.js {#ethersjs}

While Web3 is still mostly used, ethers.js has been getting a lot more traction as an alternative in the last year and is the one integrated into *create-eth-app*. You can work with this one, change it to Web3 or consider upgrading to ethers.js v5 which is almost out of beta.

The Graph {#the-graph}

GraphQL is an alternative way for handling data compared to a Restful API. They have several advantages over Restful Apis, especially for decentralized blockchain data. If you are interested in the reasoning behind this, have a look at GraphQL

Will Power the Decentralized Web.

Usually you would fetch data from your smart contract directly. Want to read the time of the latest trade? Just call `MyContract.methods.latestTradeTime().call()` which fetches the data from an Ethereum node into your dapp. But what if you need hundreds of different data points? That would result in hundreds of data fetches to the node, each time requiring an RTT making your dapp slow and inefficient. One workaround might be a fetcher call function inside your contract that returns multiple data at once. This is not always ideal though.

And then you might be interested in historical data as well. You want to know not only the last trade time, but the times for all trades that you ever did yourself. Use the *create-eth-app* subgraph package, read the documentation and adapt it to your own contracts. If you are looking for popular smart contracts, there may even already be a subgraph. Check out the subgraph explorer.

Once you have a subgraph, it allows you to write one simple query in your dapp that retrieves all the important blockchain data including historical ones that you need, only one fetch required.

Apollo {#apollo}

Thanks to the Apollo Boost integration you can easily integrate the graph in your React dapp. Especially when using React hooks and Apollo, fetching data is as simple as writing a single GraphQl query in your component:

```js const { loading, error, data } = useQuery(myGraphQlQuery)

React.useEffect(() => { if (!loading && !error && data) { console.log({ data }) } }, [loading, error, data]) ```

## Templates {#templates}

On top you can choose from several different templates. So far you can use an Aave, Compound, UniSwap or sablier integration. They all add important service smart contract addresses along with pre-made subgraph integrations. Just add the template to the creation command like `yarn create eth-app my-eth-app --with-template aave`.

### Aave {#aave}

Aave is a decentralized money lending market. Depositors provide liquidity to the market to earn a passive income, while borrowers are able to borrow using collaterals. One unique feature of Aave are those flash loans which allow you to borrow money without any collateral, as long as you return the loan within one transaction. This can be useful for example for giving you extra cash on arbitrage trading.

Traded tokens that earn you interests are called *aTokens*.

When you choose to integrate Aave with *create-eth-app*, you will get a subgraph integration. Aave uses The Graph and already provides you with several ready-to-use subgraphs on Ropsten and Mainnet in raw or formatted form.

### Compound {#compound}

Compound is similar to Aave. The integration already includes the new Compound v2 Subgraph. The interests earning tokens here are surprisingly called *cTokens*.

### Uniswap {#uniswap}

Uniswap is a decentralized exchange (DEX). Liquidity providers can earn fees by providing the required tokens or ether for both sides of a trade. It is widely used and therefore has one of the highest liquidities for a very wide range of tokens. You can easily integrate it in your dapp to, for example, allow users to swap their ETH for DAI.

Unfortunately, at the time of this writing the integration is only for Uniswap v1 and not the just released v2.
```

Sablier {#sablier}

Sablier allows users streaming money payments. Instead of a single payday, you actually get your money constantly without further administration after the initial setup. The integration includes its own subgraph.

What's next? {#whats-next}

If you have questions about *create-eth-app*, go to the Sablier community server, where you can get in touch with the authors of *create-eth-app*. As some first next steps you might want to integrate a UI framework like Material UI, write GraphQL queries for the data that you actually need and setup the deployment.