

This is your **last** free member-only story this month. [Sign up for Medium and get an extra one](#)

Learn how to create your own Dapp with Angular 9 — Part IV Truffle.



Eli Elad Elrom

Follow

Jan 5, 2020 · 7 min read ★

In this six-part article, we will cover how to create a Dapp with Angular. In Part I, which served as an introduction we covered general information regarding developing the dapp benefits and classification. Using Angular, Angular architecture, benefits. You should [start there](#).

[In the previous articles](#), we started developing our dapp. Specifically, we learned about dapp classifications and projects and that you can break your own dapp project into five steps.



We then looked at why to use Angular and its benefits. Next, we created an Angular project, first ensuring the prerequisites were installed and then installing the Angular CLI.

We looked at the pieces that make up Angular such as components, modules, and directives. We also learned how to style a dapp by understanding Angular-style architecture and working with Angular Material.

We started building our own custom components and creating content; we split the app into a footer, header, and body and created a custom transfer component that you will be using in this article.

In this article, I will be creating the dapp's smart contract with Truffle. In part V, VI we will cover the following;

- **Part V; Integrating a smart contract in your dapp's Angular project**

- **Part IV**; Linking and connecting your dapp to the Ethereum network

You will be utilizing the following tools the Angular CLI, Truffle, ganache-cli, and MetaMask.

We will create a smart contract that you will use for your dapp with Truffle, and then you'll use the web3 library to connect to the Ethereum local network and call the smart contract's functions and events. MetaMask will be used to manage and connect to your account.

Transfer a Smart Contract

You already have the front-end logic to transfer tokens in your app from the previous articles; however, you don't have a smart contract to interact with the blockchain. Smart contracts can be created before the front-end portion, after, or in parallel (if you work with a team of developers).

To get started, you will create a new folder in your ethdapp project to hold the Truffle project.

In real-life projects with multiple developers, the smart contract could be a separate project. For simplicity, you will be including it in your project so you can utilize the WebStorm Terminal window's bottom tab to run commands.

Truffle Suite

You will be using Truffle as it's one of the most popular tools and has integrated libraries that help expedite the development cycle. Truffle Suite includes Truffle, Ganache, and Drizzle;

"Truffle is a development environment, testing framework and asset pipeline for Ethereum, aiming to make life as an Ethereum developer easier." — <https://github.com/trufflesuite/truffle>

To install Truffle globally on your machine (at the time of writing, the current Truffle version is 5.1.6) by will use npm.

Start by creating a folder called truffle inside your project and

initialize Truffle to create the project. You can see the expected output in Figure 10–1.

```
cd ~/Desktop/ethdapp
mkdir truffle
cd truffle
npm install -g truffle
truffle init
```

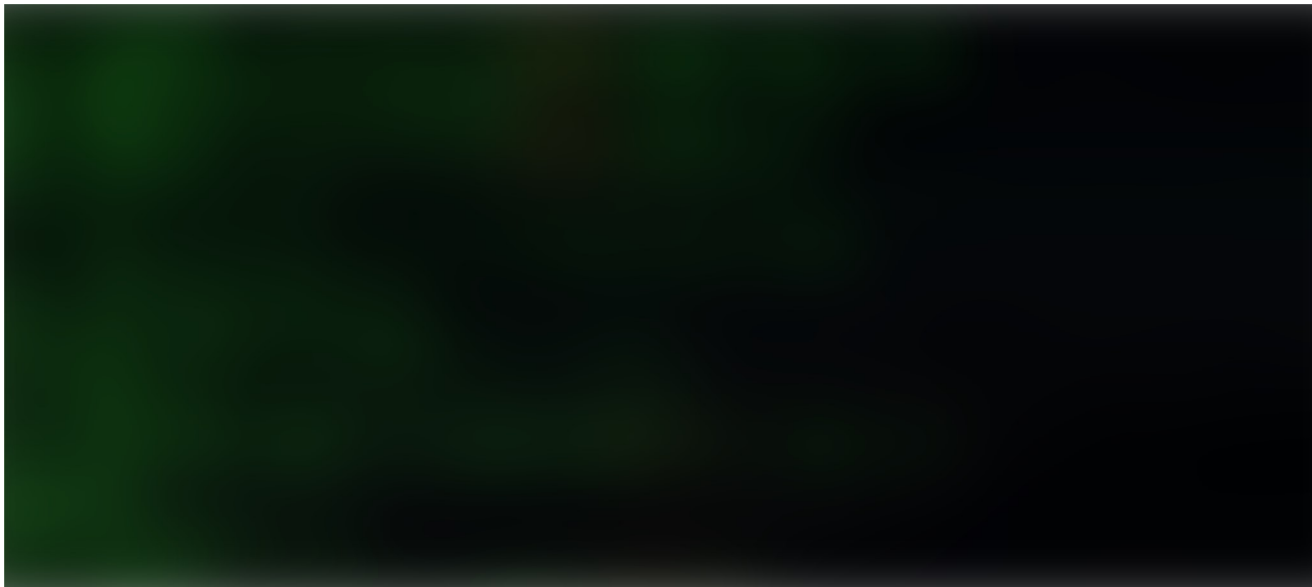


Figure 1. Output of creating a Truffle project

Tip If you have truffle installed already and you get errors such as “Error: Truffle Box,” uninstall Truffle, and then re-install it and try again. To re-install truffle in case of error messages, remove truffle globally and install it again and follow the process below;

```
npm uninstall -g truffle
```

After re-installing or performing a fresh install, run the truffle init command again and make sure you run the test in a new Terminal window to ensure the changes were applied.

```
truffle compile
truffle migrate
```

```
truffle test
```

Create a Smart Contract

We will create a smart contract and call it Transfer.sol; put it here:

```
truffle/contracts/Transfer.sol
```

The contract will allow you to transfer funds from one account to another.

To do so, first navigate to the location of the contracts in Truffle and use an editor to create a new file.

```
cd ethapp/truffle/contracts  
vim Transfer.sol
```

The complete Transfer.sol code is listed here:

```
pragma solidity ^0.5.0;  
contract Transfer {  
    address payable from;  
    address payable to;  
    constructor() public {  
        from = msg.sender;  
    }  
    event Pay(address _to, address _from, uint amt);  
    function pay( address payable _to ) public payable returns  
    (bool) {  
        to = _to;  
        to.transfer(msg.value);  
        emit Pay(to, from, msg.value);  
        return true;  
    }  
}
```

Let's walk through the code. First, you need to define the solidity version you will be using and the contract name.

```
pragma solidity ^0.5.0;
contract Transfer {
    address payable from;
    address payable to;
    constructor() public {
        from = msg.sender;
    }
}
```

You will be using a Pay event that will be dispatched once the pay function is used.

```
event Pay(address _to, address _from, uint amt);
```

The pay function uses the Pay event to interact with the network and do the actual transfer.

```
function pay( address payable _to ) public payable returns
(bool) {
    to = _to;
    to.transfer(msg.value);
    emit Pay(to, from, msg.value);
    return true;
}
```

That's it. You kept it basic and simple with only one event and one function.

Create the Truffle Development Network

The next step is to replace the

```
truffle/truffle-config.js
```

file with the following configuration:

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 8545,
      network_id: "*",
      gas: 5000000,
      gasPrice: 1000000000000
    }
  }
};
```

Notice that you point to port 8545, which will help you when you run MetaMask later.

Deploy the Smart Contract

The other configuration file you need is the deploy contract file. Create a deployment file and call it

```
truffle/migrations/2_deploy_contracts.js
```

In this config file, all you do is to point to the Transfer smart contract SOL code you created.

```
var Transfer = artifacts.require("./Transfer.sol");
module.exports = function(deployer) {
  deployer.deploy(Transfer);
};
```

Install Ganache

To get started, you can install Ganache globally with npm and confirm it's working correctly by calling the help command.

```
[sudo] npm install -g ganache-cli  
ganache-cli help
```

If you have installation issues or want to get more information regarding the tool, visit the Ganache GitHub page: <https://github.com/trufflesuite/ganache-cli>.

You can also check the version of CLI by running this command:

```
ganache-cli -v
```

At the time of writing + ganache-cli@6.7.0.

Now you are ready to create your network on port 8545 with Ganache, so navigate to the Truffle project, and run this command:

```
cd ethdapp/truffle  
ganache-cli -p 8545
```

Next, in a new Terminal window, let's compile and deploy your contract while ganache is still running.

```
truffle compile
```

The compiled output should provide success, creating your contract in the Contract folder.

```
Compiling your contracts...  
=====
```



```
✓ Fetching solc version list from solc-bin. Attempt #1
✓ Downloading compiler. Attempt #1.
> Compiling ./contracts/Migrations.sol
> Compiling ./contracts/Transfer.sol
> Artifacts written to ~Desktop/ethdapp/truffle/build/contracts
> Compiled successfully using:
- solc: 0.5.12+commit.7709ece9.Emscripten.clang
```

The file that was created is `Transfer.json`, which you will be using in your dapp to interact with the network. Next, you will deploy your contract with the `migrate` command.

```
> truffle migrate --network development
```

The output should confirm the contract was migrated to the network, as shown in Figure 2.



Figure 2. Truffle migrate project

The output summary should also show that the deployment went well and a charge.

```
Summary
```

```
=====
```

```
> Total deployments:    2
> Final cost:           0.049916 ETH
```

Truffle Console

Now that you have the contract compiled and deployed, to interact with the network, start a console, as shown here:

```
truffle console --network development
```

A good resource for the commands you can run against the Truffle CLI is at the Ethereum JavaScript API wiki page here: <https://github.com/ethereum/wiki/wiki/JavaScript-API>.

Accounts

If you run `getAccounts`, you'll get a list of accounts associated with your wallet.

```
truffle(development)> web3.eth.getAccounts()

[ '0x1eFf25A40C82EA65BC88E45d02368897EC922FEf',
  '0xC135058b33d5df78636Cf14b74F281f95c4a407c',
  '0xe682300Ef633F7d4f0d8Cb07c1bAD5d9B4eaE974'
...]
```

You can then define `address1` and `address2` as the first and second accounts.

```
truffle(development)> web3.eth.getAccounts().then( function(a)
{address1=a[0]})

undefined
```

```
truffle(development) > web3.eth.getAccounts().then(function(a)
{address2=a[1]})

undefined
```

Now that they are defined, you can call them and get the first and second accounts in the output.

```
truffle(development) > address1

'0x1eFf25A40C82EA65BC88E45d02368897EC922FEf'

truffle(development) > address2

'0xC135058b33d5df78636Cf14b74F281f95c4a407c'
```

You can also use `getBalance` to get the balance you have in these addresses.

```
truffle(development) > web3.eth.getBalance(address1)
'9994213440000000000000'

truffle(development) > web3.eth.getBalance(address2)
'100000000000000000000000'
```

Test the Transfer of a Smart Contract

Now that you have defined two addresses and you know the balance in these accounts, you can define your contract and pass some funds between the accounts. To do so, first define the contract and call it `transferSmartContract`.

```
truffle(development) > Transfer.deployed().
then(function(instance){transferSmartContract = instance;})

undefined
```

Next, run the `transferSmartContract` variable you defined to ensure it worked and show the object value.

```
truffle(development)> transferSmartContract
```

Now you can transfer funds with your smart contract between the two accounts. Account 2 holds a nice round number, so you will transfer 5 eth.

```
truffle(development)> transferSmartContract.pay(address2, {from:
address1, value: 5});
```


Ethereum Truffle Ganache Dapps Blockchain

The command output shows information about the transaction and mining. Now you are able to see the balance updated.

```
web3.eth.getBalance(address1);
'9994213439999999995'
```

```
web3.eth.getBalance(address2);
'10000000000000000005'
```



the , and you were able to transfer tokens between two addresses.

Let's recap!

In this article, we created a transfer smart contract and Truffle development project as well as connected to the Ganache local development network. We learned how to work with the Ethereum network via Truffle and how to test your smart contract. We tested the transfer of funds using a smart contract via the command line.

Where to go from here

Continue following our articles as in the next article we will be covering; [Integrating the smart contract with the dapp](#)

To learn more about what's possible with Blockchain as well as develop your own project check out [The Blockchain Developer](#).