ConsenSys Academy's Developer Program Final Project

Congratulations on reaching this point in the program! In this document you will find all of the information related to your final project. Please follow this document for all project related instructions and requirements.

This project is to be done individually.

Project Submission

Once you've completed your project and are ready to submit it for grading, please complete the Project Submission Form. In this form, you will be asked to include:

- Email Address
- First Name
- Last Name
- Link to project

Here is an example of the Project Evaluation form that will be used to evaluate your project, so you know what we are looking for.

Course Grade Distribution

Quizzes - 30%
Project (Code & Functionality) - 70%

A grade of at least 80% for the course is required to receive a certificate of proficiency. Else you will be issued a certificate of completion.

Date of submission

To be eligible for a certificate, you need to submit the project within four weeks of receiving the instructions.

Final Project Specifications

This section explains the specifications and requirements for the Developer Program Final Project.

You have the option to choose from several final project ideas and will be expected to implement the following features.

You will submit a final project containing the following items:

- A project README.md that explains your project

    - What does your project do?
    - How to set it up?
        - Run a local development server

- Your project should be a truffle project
    - All of your contracts should be in a contracts directory
        - `Truffle compile` should successfully compile contracts
    - Migration contract and migration scripts should work
        - `Truffle migrate` should successfully migrate contracts to a locally running ganache-cli test blockchain on port 8545
    - All tests should be in a tests directory
        - Running `truffle test` should migrate contracts and run your tests

- Smart Contract code should be commented according to the specs in the documentation

- Create at least 5 tests for each smart contract
    - Write a sentence or two explaining what the tests are covering, and explain why you wrote those tests

- A development server to serve the front end interface of the application
    - It can be something as simple as the lite-server used in the truffle pet shop tutorial

- A document called design_pattern_desicions.md that explains why you chose to use the design patterns that you did.
- A document called avoiding_common_attacks.md that explains what measures you took to ensure that your contracts are not susceptible to common attacks. (*Module 9 Lesson 3)*

- Implement/ use a library or an EthPM package in your project
    - If your project does not require a library or an EthPM package, demonstrate how you would do that in a contract called LibraryDemo.sol

- Record your screen as you demo the application, showing and explaining how you included the required components.

We ask that you develop your application and run the other projects during evaluation in a VirtualBox VM running Ubuntu 16.04 to reduce the chances of run time environment variables.

Requirements

- User Interface Requirements:
  - Run the app on a dev server locally for testing/grading
  - You should be able to visit a URL and interact with the application
    - App recognizes current account
    - Sign transactions using MetaMask or uPort
    - Contract state is updated
    - Update reflected in UI

- Test Requirements:
  - Write 5 tests for each contract you wrote
    - Solidity or JavaScript
  - Explain why you wrote those tests
  - Tests run with truffle test

- Design Pattern Requirements:
  - Implement a circuit breaker (emergency stop) pattern
  - What other design patterns have you used / not used?
    - Why did you choose the patterns that you did?
    - Why not others?

- Security Tools / Common Attacks:
  - Explain what measures you've taken to ensure that your contracts are not susceptible to common attacks

- Use a library or extend a contract
  - Via EthPM or write your own

- Deploy your application onto one of the test networks. Include a document called deployed_addresses.txt that describes where your contracts live (which testnet and address).
  - You can verify your contract source code using etherscan for the appropriate testnet https://etherscan.io/verifyContract
  - Evaluators can check by getting the provided contract ABI and calling a function on the deployed contract at

https://www.myetherwallet.com/#contracts or checking the verification on etherscan

- Stretch requirements (**for bonus points, not required**):
    - Implement an upgradable design pattern
    - Write a smart contract in LLL or Vyper
    - Integrate with an additional service, maybe even one we did not cover in this class

      For example:
        - IPFS
            - Users can dynamically upload documents to IPFS that are referenced via their smart contract
        - uPort
        - Ethereum Name Service
            - A name registered on the ENS resolves to the contract, verifiable on rinkeby.etherscan.io/*contract_name*
        - Oracle

A note on project difficulty

Depending on your current web development skill level, this project may be more or less difficult.

Many of the project ideas offer a range of complexity in their implementations. Implementing the basic features of a project will demonstrate that you can create and deploy smart contracts and link them to a user interface.

In this course, we emphasize functionality and security over style. This is not a course in interface design, it is a course in Ethereum and decentralized application development using Solidity. We want you to demonstrate what you have learned throughout the course. This project should show us that you understand how to write secure smart contracts that follow best practices.

Building a complex, feature-rich decentralized application is not required to pass the course. If you are interested in working in the blockchain space professionally, we

strongly encourage going above and beyond the minimum requirements for the project as this project will provide great material for discussion in the future.

Final Project Ideas
This section outlines the final project ideas and options for the 2018 Developer Program.

1. Implement your own idea
Description: You have the option to create your own dApp.

If you have an idea for a dApp, this is a great reason to build (BUIDL) it! You don't need to build out your entire idea, just make sure that you build enough of it to fulfill the requirements for the project. Remember, evaluators are looking for functionality and demonstration of knowledge over style.

Be sure to clearly explain what your dApp is, why you are building it and what the evaluator needs to do to in order to successfully evaluate it in the project README.md.

Considerations:
Keep in mind that your project submission needs to fulfill the project specifications outlined in the rubric and the project specifications document.

User Stories:
Creating several user stories can help the evaluator understand what your dApp is and how potential users are supposed to interact with it.

User stories outline how users will interact with the application. They should be descriptions of end goals of the application rather than descriptions of features.

A Google search will show you a bunch of examples.

Here are some suggestions for additional components that your project could include:

- Deploy your dApp to a testnet
    - Include the deployed contract address so people can interact with it
    - Serve the UI from IPFS or a traditional web server

2. Online Marketplace

Description: Create an online marketplace that operates on the blockchain.

There are a list of stores on a central marketplace where shoppers can purchase goods posted by the store owners.

The central marketplace is managed by a group of administrators. Admins allow store owners to add stores to the marketplace. Store owners can manage their store's inventory and funds. Shoppers can visit stores and purchase goods that are in stock using cryptocurrency.

User Stories:
An administrator opens the web app. The web app reads the address and identifies that the user is an admin, showing them admin only functions, such as managing store owners. An admin adds an address to the list of approved store owners, so if the owner of that address logs into the app, they have access to the store owner functions.

An approved store owner logs into the app. The web app recognizes their address and identifies them as a store owner. They are shown the store owner functions. They can create a new storefront that will be displayed on the marketplace. They can also see the storefronts that they have already created. They can click on a storefront to manage it. They can add/remove products to the storefront or change any of the products' prices. They can also withdraw any funds that the store has collected from sales.

A shopper logs into the app. The web app does not recognize their address so they are shown the generic shopper application. From the main page they can browse all of the storefronts that have been created in the marketplace. Clicking on a storefront will take them to a product page. They can see a list of products offered by the store, including their price and quantity. Shoppers can purchase a product, which will debit their account and send it to the store. The quantity of the item in the store's inventory will be reduced by the appropriate amount.

Here are some suggestions for additional components that your project could include:

- Add functionality that allows store owners to create an auction for an individual item in their store
- Give store owners the option to accept any ERC-20 token
- Deploy your dApp to a testnet
    - Include the deployed contract address so people can interact with it
    - Serve the UI from IPFS or a traditional web server

3. Proof of Existence dApp

Description: This application allows users to prove existence of some information by showing a time stamped picture/video.

Data could be stored in a database, but to make it truly decentralized consider storing pictures using something like IPFS. The hash of the data and any additional information is stored in a smart contract that can be referenced at a later date to verify the authenticity.

User Stories:
A user logs into the web app. The user can upload some data (pictures/video) to the app, as well as add a list of tags indicating the contents of the data.

The app reads the user's address and shows all of the data that they have previously uploaded.

Users can retrieve necessary reference data about their uploaded items to allow other people to verify the data authenticity.

Here are some suggestions for additional components that your project could include:

- Make your app mobile friendly, so that people can interact with it using a web3 enabled mobile browser such as Toshi or Cipher.
    - Allow people to take photos with their mobile device and upload them from there
- Deploy your dApp to a testnet
    - Include the deployed contract address so people can interact with it
    - Serve the UI from IPFS or a traditional web server

4. Bounty dApp

Description: Create a bounty dApp where people can post or submit work.

Considerations:
Keep in mind that your project submission needs to fulfill the project specifications outlined in the rubric and the project specifications document.

User Stories:
As a job poster, I can create a new bounty. I will set a bounty description and include the amount to be paid for a successful submission. I am able to view a list of bounties that I have already posted. By clicking on a bounty, I can review submissions that have

been proposed. I can accept or reject the submitted work. Accepting proposed work will pay the submitter the deposited amount.

As a bounty hunter, I can submit work to a bounty for review.

Here are some suggestions for additional components that your project could include:

- Include an arbitration process that will settle any disputes between posters and submitters
- Allow job posters to pay out bounties in ERC-20 tokens
- Deploy your dApp to a testnet
    - Include the deployed contract address so people can interact with it
    - Serve the UI from IPFS or a traditional web server