ConsenSys Academy's 2018 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.

Rationale
Peer evaluation of projects offers an excellent opportunity to learn from your peers. Evaluating the work of your peers is an excellent strategy to reinforce your understanding. It can help identify gaps in your knowledge and reinforce concepts that you already understand. With clearly defined project specifications and requirements, the peer evaluation process can enhance your learning experience while also providing you with better feedback. It is also an extremely useful method that is often used in the industry.

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
    - We highly recommend either uploading your code to GitHub and sharing that link, or uploading a zip folder to a Google Drive and sharing that link

Project Submission Form

We will then assign you to a group and give your group a group number. You will be expected to review the projects that the peers in your group have submitted. This is a requirement to successfully pass the course.

The final project grading rubric should help you assign a total point value for the project. You are required to submit a completed rubric for each project that you evaluate.

These evaluations count toward your final grade for the course. If it is determined that you do not evaluate your peers fairly or honestly, you will not receive credit for this part of the course. In addition, your project grade will not be released until you've completed the peer grading portion.

Each group will be given **two weeks** to grade their projects. Once you've finished grading all of the projects in your group, you will be asked to fill out the Rubric Submission Form which will include the following:

- Email Address
- First Name
- Last Name
- Group Number
- Link to Grading Rubric 1
- Link to Grading Rubric 2
- …
- Link to Grading Rubric #

Rubric Submission Form

The last date for project submissions will be on August 27th, which is the last date of the course.

Review

Each student's set of evaluations is reviewed by a mentor for clarity and consistency. Students that do not receive a fair and honest evaluation from their peers will be given the chance to have a mentor review their project. A mentor evaluation is final and will not be reconsidered.

Remember that you are not evaluating your peers project's on how complex, intricate, clever or beautiful they are. You are evaluating based on the rubric. Some students may be taking this course with extensive experience in web development and can rapidly develop a professional looking user interface, others may have simple interfaces with basic text and buttons to demonstrate basic functionality.

Appeals

If you feel that you have not received a fair grade based on the project specifications and the grading rubric, you can appeal your grade. If you choose to appeal, a mentor will review the peer evaluations and personally grade your project *if deemed necessary*. The mentor will follow the project specs and rubric and you may receive a lower grade

than you had before your appeal. Even if this is the case, **your final grade will be the grade given by the mentor.**

Grade Distribution
Project (Code & Functionality) - 80%
Submitted Peer Grading Reports - 20%

Final Project Specifications
This section explains the specifications and requirements for the 2018 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 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

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 / 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 emergency stop
    - 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
    - Via EthPM or write your own

- Stretch requirements (**for bonus points, not required**):
    - Deploy your application onto the Rinkeby test network. Include a document called deployed_addresses.txt that describes where your contracts live on the test net.
    - Integrate with an additional service, maybe even one we did not cover in this class

      For example:
        - IPFS
        - uPort
        - Ethereum Name Service
        - 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, but ConsenSys will screening projects as part of the recruitment process. If you are interested in working at ConsenSys, we strongly encourage going above and beyond the minimum requirements for the project.

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

1. 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

2. Battleship

Description: Battleship is a two player guessing game where each player attempts to sink the other's battleships by correctly guessing the location of their battleships. You can read about the details of the game here.

Considerations:
All data posted on the blockchain is public. You may want to consider using a commit reveal design pattern that does not reveal a player's submitted ship placement, but verifies their ship placement after the game is over. This will prevent players from being able to read the blockchain to discover an opponent's ship placements.

User Stories:
A player opens the web app. The contract recognizes their address and shows any battleship games that the address has participated in (ongoing and completed). They can click on a game to see the game details.

- The player can see which squares they have guessed for each specific game as well as the outcomes (hit or miss).
- The player can make a guess if it is their turn.
- The player can create a new game, either specifying an opponent address or leaving it open to play against the first one to claim it.

When the player sinks the final ship, the players are prompted to reveal their boards. The players have a set amount of time to reveal their boards, or they lose.

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

- Allow players to bet on the outcome. The contract will store the bets that each player has made and automatically pay the winner
    - Allow bets to be placed in 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

5. 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