CSE426/526 Blockchain Application Development: Lab1 Fall 2019
B. Ramamurthy Part I: Due Date: 9/27/2019 Part II: Due date 10/11/2019

1. **Overview:** In this first lab you will learn to develop a Dapp for common problem of balloting or voting. Balloting is used in widely in many situations from choosing a winner in a democracy to deciding a verdict by the members of a jury in legal system. You are required to design and implement an end-to-end decentralized (Dapp) for digital balloting. Each of you should implement balloting for a unique use case.

2. **Description**: Balloting is an age old method for realizing consensus among participants with different opinions and opposing views. An online digital balloting opens up its application to diverse applications from a local land dispute resolution to planetary level decision on global governance. You are required to design and develop a Dapp that can enable a trustworthy and transparent online voting system with automatic counting of votes. To address these needs, you will design and implement an Ethereum blockchain-based Dapp.

3. **Your use case**: Research about online and digital voting and decide on a unique decentralized problem that you will solve. Lock this unique problem for each of you by discussing it with one of the TAs or the instructor. Think out of the box, think of unusual use cases that may benefit by the use decentralized technologies. **Only for Lab1** you can work in groups of no more than 2 people.

4. **Software installation**: Before you begin development make sure these tools are installed in your laptop: Get help from your TAs during the block hours and office hours.
    1) Truffle: v5.0.27 or latest
    https://www.npmjs.com/package/truffle
    2) Node: v10.15.3 or latest : v10.16.3.
    https://nodejs.org/en/
    3) npm: 6.4.1 or latest
    https://www.npmjs.com/get-npm
    4) ganache v2.1.1
    https://www.trufflesuite.com/ganache

5. **Design and implementation**: The core logic of a blockchain-based application is in a smart contract. That is where all the rules and regulation governing the system are coded. The smart contract (SC) encodes the main logic of the decentralized problem by means of its functions, and data, other elements such as *events* and *modifiers*. But we need to provide users access to the functions encoded in the smart contract. We plan to do that by adding a web user interface (UI) and a glue code connecting the web UI and the SC. Thus we have two components as shown in figure 1. This will be the standard structure for all the Dapps you will develop in this course.

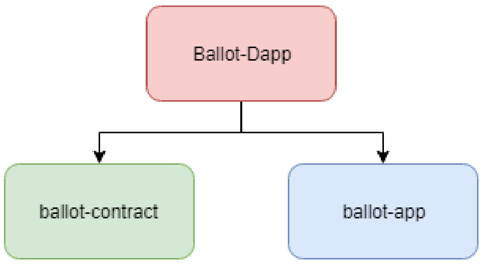

Figure 1 Overall architecture of a Dapp

Where do you begin your design and development? You will begin with the SC, and the <Dapp>-contract based on the problem requirement. Then move on to develop the <Dapp>-app based on the features of the SC that need to be exposed to the user.

6. **Part I: Smart contract design and development**: Now let's focus on the development of the Consider the voting sample code in the in Solidity documentation. Analyze the code and design a *contract diagram*. Execute it and understand the functions. Ballot.sol is already pre-loaded in the Remix environment. Compile, deploy and interact with it.

   Add any other functions you may need (register(), unregister()?? and others you need based on your choice the problem in step 3. Add other rules you may want to enforce for your use case. For example for your particular use case, voter must be 18 years or older. For another use case, voter must be older than 65 (senior citizen) and so on. If this is a rule, it may be necessary to input the age during registration of the voter.

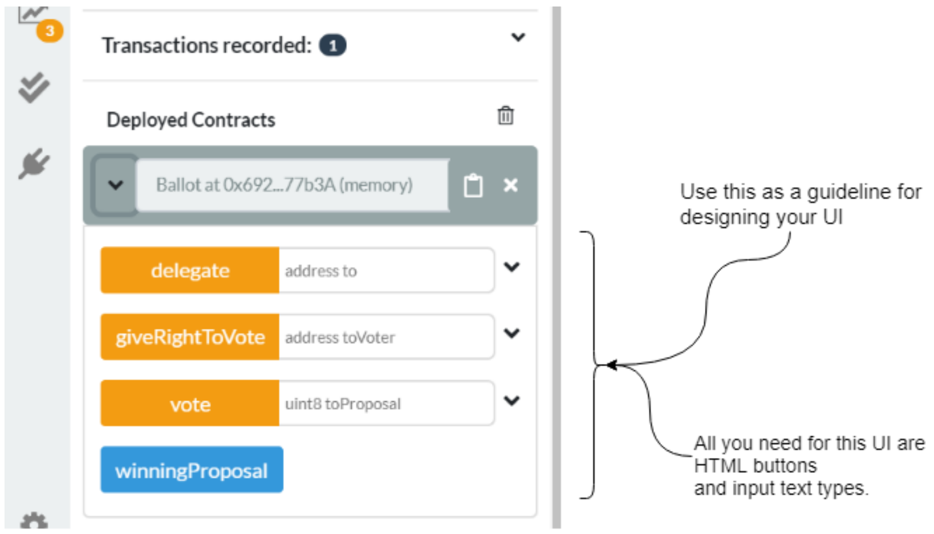
Transactions recorded: 1
Deployed Contracts
Ballot at 0x692...77b3A (memory)
delegate
address to
giveRightToVote
address toVoter
vote
uint8 toProposal
winningProposal
Use this as a guideline for designing your UI
All you need for this UI are HTML buttons and input text types.

Figure 2 Ballot user interface on Remix IDE

   Using Ballot.sol as the base code, and the contract diagram as the design, develop the smart contract: let it be named XYZBallot.sol where XYZ stands for your use case. It could be an abbreviation of your application. Submit the XYZBallot.sol and an image similar to figure 2 showing the interface of your application on Remix. Demo it to your TA to get full grade. **Due date: 9/27/2019 6.00pm.**

7. **Requirements: You cannot simple submit the Ballot.sol.** There has to be a significant addition:

At least four more significant functions, and at least two more modifiers. The use case you chose should be unique for this class and must be a practical one. Also start a private github repo for the code you create for the labs in this course.

8. **Part II: Develop the web UI** or the <Dapp>-app part for the smart contract developed in step 6. You will integrate the two components using Truffle suite, and use Ganache test chain for deployment. Partial directory structure is shown in figure 3. More details about the development of the web UI and app.js will be provided during the lecture and also during the recitation and block hours. For the web server you will use Node.js server and any suitable web frameworks such as Express, React and Bootstrap. **Due date 10/11/2019 by 6pm.**

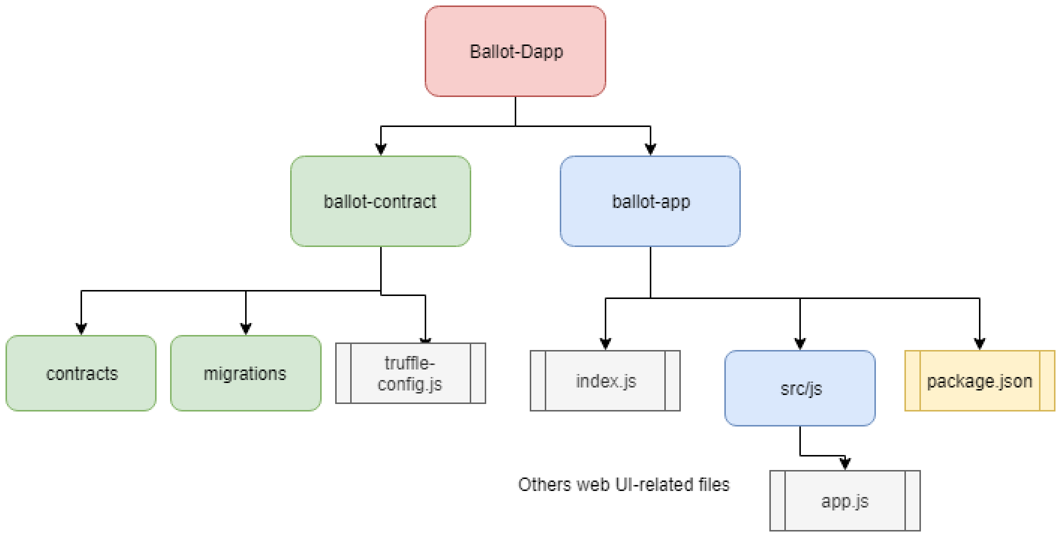

Figure 3 Directory structure for lab1