**Powerpoint Presentation - Slides**

**My First Ethereum Smart-Contract**

**A brief overview of the topics we will cover in this workshop:**

* The basics: Blockchain, smart-contracts, Ethereum accounts, Ethereum networks, Solidity…
* Our first smart-contract on Remix
* Setting up the development environment
* Setup and configuration of our first smart-contract project
* The basics of Solidity
* A more advanced smart-contract - lottery on the blockchain
* Smart-contract compilation
* Smart-contract deployment using Infura
* Testing our smart-contract with Mocha
* A very simple DAPP - interacting with our smart-contract from a web fronted using web3
* Useful references

**Let's get started with the basics…**

* The Ethereum network
* Interacting with various Ethereum Networks
* Different types of ethereum accounts
* What's a transaction
* Wei and Ether
* Smart-contracts
* The Solidity programming language

**Hello World - our first smart-contract**

* Writing, compiling, deploying, testing, debugging… our smart-contract on Remix in 10 minutes: <http://remix.ethereum.org>
* Selecting the environment: In-memory javascript VM or injected Web3 (from MetaMask)
* Selecting a test account and providing other required parameters
* Deploying our contract or loading an already deployed contract from a specific address
* Testing and playing around with our smart-contract
* Debugging on Remix

**Setting up our environment for local development**

* Remix does everything - why bother with a local environment?
* Installing Node.js: <https://nodejs.org/en/download>
* Installing an editor - Atom: <https://atom.io> + language-ethereum
* Installing and configuring the MetaMask Chrome/Firefox plugin  
  + Selecting a public or private network
  + Creating/Importing an account
  + Account info on Etherscan
  + Getting free test Ether on Rinkeby Faucet

**Creating our first Node.js smart-contract**

* Setting up our project: npm init - creates a package.json file
* Project structure: ethereum/contracts folder, test folder
* Installing a few basic npm modules:

npm install --save solc mocha ganache-cli truffle-hdwallet-provider@0.0.3 web3

* Or, download the project from github and execute **npm install** in the project directory

**More on Solidity**

* Docs: <https://solidity.readthedocs.io/en/v0.4.24>
* Value types: bool, int, uint8, int256, fixed…
* Address: holds 20 byte value - members: balance, transfer
* Reference types: Arrays, struct, mapping  
  uint[] memory a = new uint[](5);  
  mapping(address => uint) public balances;
* Modifiers - check a specific condition prior to executing a function
* The msg object: sender, value, gas…
* Transferring ether: address.transfer(amount)
* Function keywords: public, payable, view, returns
* State changes and gas cost

**A more advanced smart-contract - lottery on the blockchain**

* Contract manager and players
* Entering the lottery with a specific amount
* Everyone can consult the current number of players
* Picking a winner - can only be executed by the contract manager

**Compiling and deploying a smart-contract**

* A simple compilation script: node compile.js
* Bytecode and ABI (Application Binary Interface)
* A simple deployment script: node deploy.js
* Connecting to your wallet (MetaMask, Ganache…) using web3 and the truffle-hdwallet-provider

**Testing your smart-contract with Mocha - https://mochajs.org/**

* Deploying our smart-contract locally with ganache-cli
* Executing a code block before each test: beforeEach()
* Defining a group of test cases: describe()
* Defining a single test case: it()
* The assert library: assert.equal(1, players.length); …

**DAPP's - Interacting with a smart-contract from a web frontend**

***DAPP's 1: Creating the project***

* Creating boiler-plate React apps: npm install -g create-react-app
* Creating a new project: create-react-app lottery-web
* Installing some packages: npm install --save web3
* Or: Download the app from github, open a cmd window, navigate to the project directory and execute: npm install (this installs all required npm modules)
* Running the app: npm run start

***DAPP's 2: Accessing smart-contract methods from JavaScript***

* Connecting the web3 provider that was injected by MetaMask with our own web3 component: const web3 = new Web3(window.web3.currentProvider);
* Copy/paste contract address and ABI after smart-contract compilation
* Calling a smart-contract method:   
  const manager = await lottery.methods.manager().call();
* Getting the contract address: const contrAddr = await lottery.options.address;
* Getting the list of accounts: const accounts = await web3.eth.getAccounts();
* Getting the contract balance:   
  const balance = await web3.eth.getBalance(lottery.options.address);
* Sending ether in a transaction:  
  await lottery.methods.enter().send({from: accounts[0],   
  value: web3.utils.toWei(this.state.value, 'ether') })
* Converting from wei to ether: web3.utils.fromWei(this.state.balance, 'ether')

**Playing around with your smart-contract**

* Make some modifications to your smart-contract code
* Re-compile, deploy, run and test your modified contract
* Add a new function to your smart-contract
* Debug your smart contract in Remix

**Step-by-step docs and References**

* Step-By-Step Environment Setup.docx
* Step-By-Step Project Setup.docx
* Step-By-Step Compiling Deploying Running a Project.docx
* Useful Links.docx