DA51 Lab Session 8: Dapp Crowdfunding

Description:

This lab session focuses on developing a decentralized crowdfunding application (DApp) on Ethereum using Solidity and Truffle Suite. The DApp will allow users to create campaigns, contribute Ether, manage withdrawals, and implement a refund mechanism for unsuccessful campaigns. Deliverables include the Solidity contract source code, test cases, frontend code, and a brief report.

Question 2:

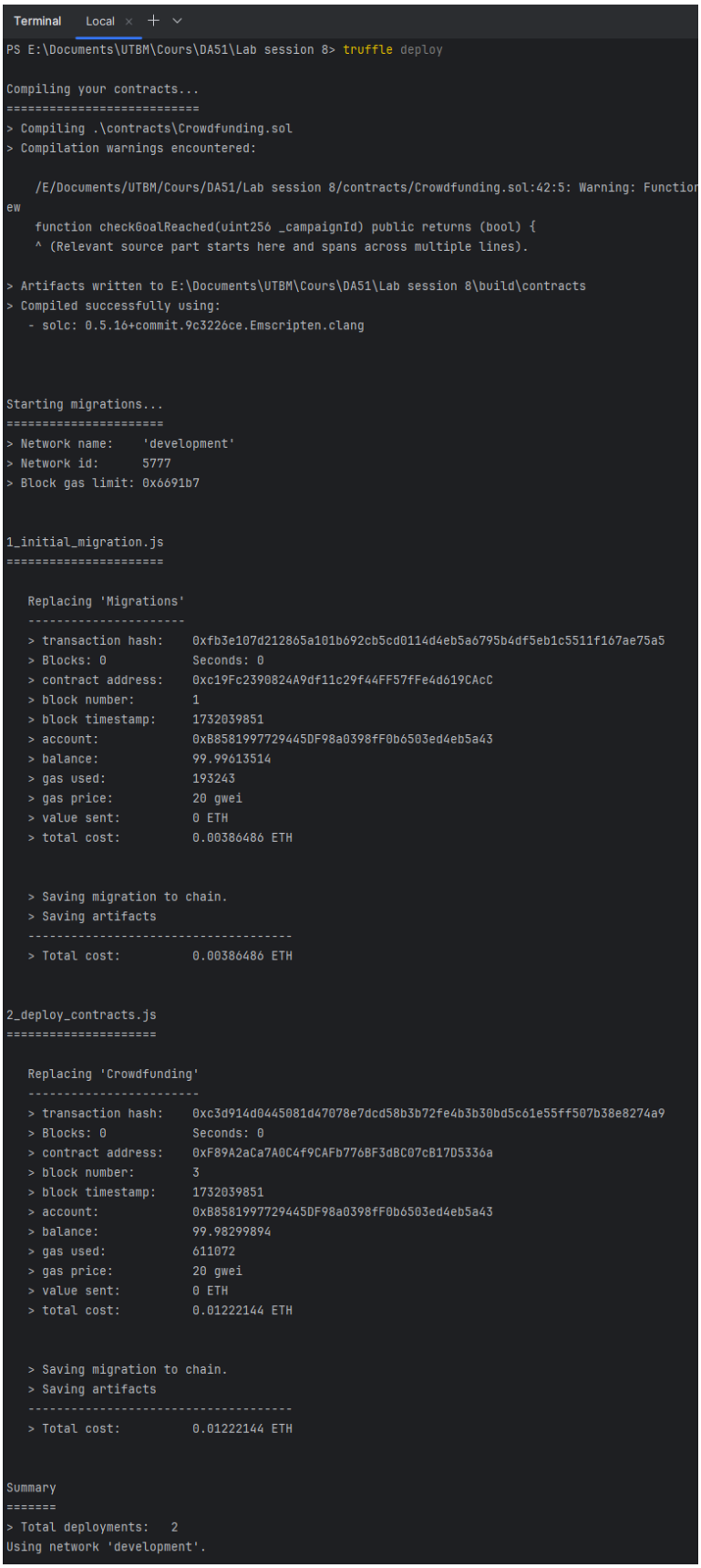
Smart Contract:

Crowdfunding.sol

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.4.22 <0.9.0;

contract Crowdfunding {
    struct Campaign {
        address payable creator;
        uint256 goal;
        uint256 deadline;
        uint256 balance;
        bool closed;
    }

    mapping(uint256 => Campaign) public campaigns;
    uint256 public campaignId;

    function startCampaign(uint256 _goal, uint256 _deadline) public returns (uint256) {
        require(_goal > 0, "Goal must be greater than 0");
        require(_deadline > block.timestamp, "Deadline must be in the future");

        campaigns[campaignId] = Campaign( creator: {
            creator: msg.sender,
            goal: _goal,
            deadline: _deadline,
            balance: 0,
            closed: false
        });

        campaignId++;
        return campaignId - 1;
    }

    function contribute(uint256 _campaignId) public payable returns (uint256) {
        Campaign storage campaign = campaigns[_campaignId];
        require(msg.value > 0, "Contribution must be greater than 0");
        require(!campaign.closed, "Campaign is closed");
        require(block.timestamp < campaign.deadline, "Deadline has passed");

        campaign.balance += msg.value;
        return campaign.balance;
    }

    function checkGoalReached(uint256 _campaignId) public returns (bool) {
        Campaign storage campaign = campaigns[_campaignId];
        require(!campaign.closed, "Campaign is closed");
        //require(block.timestamp >= campaign.deadline, "Deadline has not passed");

        if (campaign.balance >= campaign.goal) {
            return true;
        } else {
            return false;
        }
    }

    function closeCampaign(uint256 _campaignId) public {
        Campaign storage campaign = campaigns[_campaignId];
        require(!campaign.closed, "Campaign is already closed");

        campaign.closed = true;
    }

    function withdraw(uint256 _campaignId) public {
        Campaign storage campaign = campaigns[_campaignId];
        require(campaign.closed, "Campaign is not closed");
        require(msg.sender == campaign.creator, "Only the creator can withdraw funds");

        campaign.creator.transfer( value: campaign.balance);
    }
}
```

Question 3:

Compile and migration:



```
Terminal  Local

PS E:\Documents\UTBM\Cours\DA51\Lab session 8> truffle deploy

Compiling your contracts...
===========================
> Compiling .\contracts\Crowdfunding.sol
> Compilation warnings encountered:

    /E/Documents/UTBM/Cours/DA51/Lab session 8/contracts/Crowdfunding.sol:42:5: Warning: Functior
ew
    function checkGoalReached(uint256 _campaignId) public returns (bool) {
    ^ (Relevant source part starts here and spans across multiple lines).

> Artifacts written to E:\Documents\UTBM\Cours\DA51\Lab session 8\build\contracts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang



Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 0x6691b7


1_initial_migration.js
======================

   Replacing 'Migrations'
   ----------------------
   > transaction hash:    0xfb3e107d212865a101b692cb5cd0114d4eb5a6795b4df5eb1c5511f167ae75a5
   > Blocks: 0            Seconds: 0
   > contract address:    0xc19Fc2390824A9df11c29f44FF57fFe4d619CAcC
   > block number:        1
   > block timestamp:     1732039851
   > account:             0xB85819977294455DF98a0398fF0b6503ed4eb5a43
   > balance:             99.99613514
   > gas used:            193243
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00386486 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00386486 ETH


2_deploy_contracts.js
=====================

   Replacing 'Crowdfunding'
   ------------------------
   > transaction hash:    0xc3d914d0445081d47078e7dcd58b3b72fe4b3b30bd5c61e55ff507b38e8274a9
   > Blocks: 0            Seconds: 0
   > contract address:    0xF89A2aCa7A0C4f9CAFb776BF3dBC07cB17D5336a
   > block number:        3
   > block timestamp:     1732039851
   > account:             0xB85819977294455DF98a0398fF0b6503ed4eb5a43
   > balance:             99.98299894
   > gas used:            611072
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.01222144 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.01222144 ETH


Summary
=======
> Total deployments:   2
Using network 'development'.
```

Question 4:

Testing:

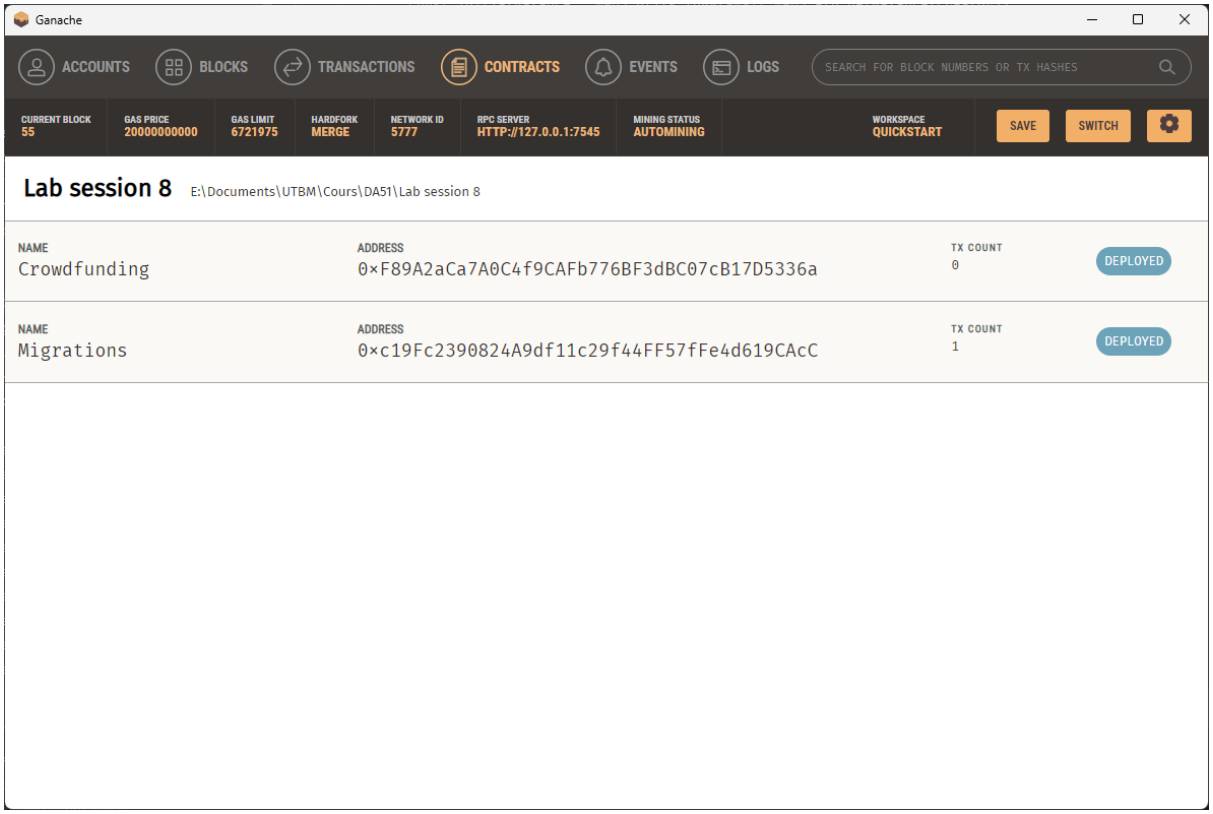
TestCrowdfunding.js

```
const Crowdfunding = artifacts.require("Crowdfunding");

contract("Crowdfunding", (accounts) :void => {
    let crowdfundingInstance;

    const [creator, contributor] = accounts;

    beforeEach(async () :Promise<void> => {
        crowdfundingInstance = await Crowdfunding.new();
    });

    it("should start a campaign", async () :Promise<void> => {
        const goal = web3.utils.toWei("10", "ether");
        const deadline :number = Math.floor( x: Date.now() / 1000) + 3600; // 1 hour from now

        await crowdfundingInstance.startCampaign(goal, deadline, { from: creator });

        const campaign = await crowdfundingInstance.campaigns(0);
        assert.equal(campaign.creator, creator, "Campaign creator is incorrect");
        assert.equal(campaign.goal.toString(), goal, "Campaign goal is incorrect");
        assert.equal(campaign.deadline.toNumber(), deadline, "Campaign deadline is incorrect");
        assert.equal(campaign.balance.toString(), "0", "Campaign balance should start at 0");
        assert.equal(campaign.closed, false, "Campaign should not be closed initially");
    });

    it("should allow contributions", async () :Promise<void> => {
        const goal = web3.utils.toWei("10", "ether");
        const deadline :number = Math.floor( x: Date.now() / 1000) + 3600;

        await crowdfundingInstance.startCampaign(goal, deadline, { from: creator });

        const contributionAmount = web3.utils.toWei("1", "ether");
        await crowdfundingInstance.contribute(0, { from: contributor, value: contributionAmount });

        const campaign = await crowdfundingInstance.campaigns(0);
        assert.equal(campaign.balance.toString(), contributionAmount, "Campaign balance should reflect the contribution");
    });

    it("should close the campaign and allow withdrawal when the goal is reached", async () :Promise<void> => {
        const goal = web3.utils.toWei("2", "ether");
        const deadline :number = Math.floor( x: Date.now() / 1000) + 3600;

        await crowdfundingInstance.startCampaign(goal, deadline, { from: creator });

        await crowdfundingInstance.contribute(0, { from: contributor, value: web3.utils.toWei("2", "ether") });

        await crowdfundingInstance.checkGoalReached(0, { from: creator });

        const campaign = await crowdfundingInstance.campaigns(0);
        assert.equal(campaign.closed, true, "Campaign should be closed after goal is reached");

        const initialBalance = web3.utils.toBN(await web3.eth.getBalance(creator));
        await crowdfundingInstance.withdraw(0, { from: creator });

        const finalBalance = web3.utils.toBN(await web3.eth.getBalance(creator));
        assert(finalBalance.gt(initialBalance), "Creator should have withdrawn funds");
    });

    it("should fail to contribute after the campaign deadline", async () :Promise<void> => {
        const goal = web3.utils.toWei("10", "ether");
        const deadline :number = Math.floor( x: Date.now() / 1000) + 1; // 1 second from now

        await crowdfundingInstance.startCampaign(goal, deadline, { from: creator });

        await new Promise( executor: (resolve) :number => setTimeout(resolve, timeout: 2000));

        try {
            await crowdfundingInstance.contribute(0, { from: contributor, value: web3.utils.toWei("1", "ether") });
            assert.fail("Contribution should fail after the deadline");
        } catch (error) {
            assert(error.message.includes("Deadline has passed"), "Expected 'Deadline has passed' error");
        }
    });
});
```

Question 5:

```
PS E:\Documents\UTBM\Cours\DA51\Lab session 8> truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.



  Contract: Crowdfunding
    √ should start a campaign
    √ should allow contributions
    √ should close the campaign and allow withdrawal when the goal is reached (59ms)
    √ should fail to contribute after the campaign deadline (2046ms)


  4 passing (2s)
```

Question 6-7:



Question 8-9-10:

Index.html:

index.html

```html
<!DOCTYPE html>
<html lang="en">
  <head...>
  <body>
  <h1>Crowdfunding DApp</h1>

  <!-- Start Campaign Section -->
  <h2>Start a Campaign</h2>
  <label for="goal">Goal (in Wei):</label>
  <input type="number" id="goal" placeholder="Enter goal in wei">
  <label for="deadline">Deadline (UNIX Timestamp):</label>
  <input type="number" id="deadline" placeholder="Enter deadline timestamp">
  <button class="btn btn-primary btn-block btn-start-campaign">Start Campaign</button>
  <p id="startCampaignResult"></p>

  <!-- Contribute Section -->
  <h2>Contribute to a Campaign</h2>
  <label for="campaignId">Campaign ID:</label>
  <input type="number" id="campaignId" placeholder="Enter campaign ID">
  <label for="amount">Amount (in Wei):</label>
  <input type="number" id="amount" placeholder="Enter amount to contribute">
  <button class="btn btn-primary btn-block btn-contribute">Contribute</button>
  <p id="contributeResult"></p>

  <!-- Check Goal Reached Section -->
  <h2>Check if Campaign Goal Reached</h2>
  <label for="campaignIdCheck">Campaign ID:</label>
  <input type="number" id="campaignIdCheck" placeholder="Enter campaign ID">
  <button class="btn btn-primary btn-block btn-check-goal">Check Goal</button>
  <p id="checkGoalResult"></p>

  <!-- Close the campaign -->
  <h2>Close a Campaign</h2>
  <label for="campaignIdClose">Campaign ID:</label>
  <input type="number" id="campaignIdClose" placeholder="Enter campaign ID">
  <button class="btn btn-primary btn-block btn-close">Close</button>
  <p id="closeResult"></p>

  <!-- Withdraw Section -->
  <h2>Withdraw Funds</h2>
  <label for="campaignIdWithdraw">Campaign ID:</label>
  <input type="number" id="campaignIdWithdraw" placeholder="Enter campaign ID">
  <button class="btn btn-primary btn-block btn-withdraw">Withdraw</button>
  <p id="withdrawResult"></p>

    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="js/bootstrap.min.js"></script>
    <script src="js/web3.min.js"></script>
    <script src="js/truffle-contract.js"></script>
    <script src="js/app.js"></script>
  </body>
</html>
```

App.js:

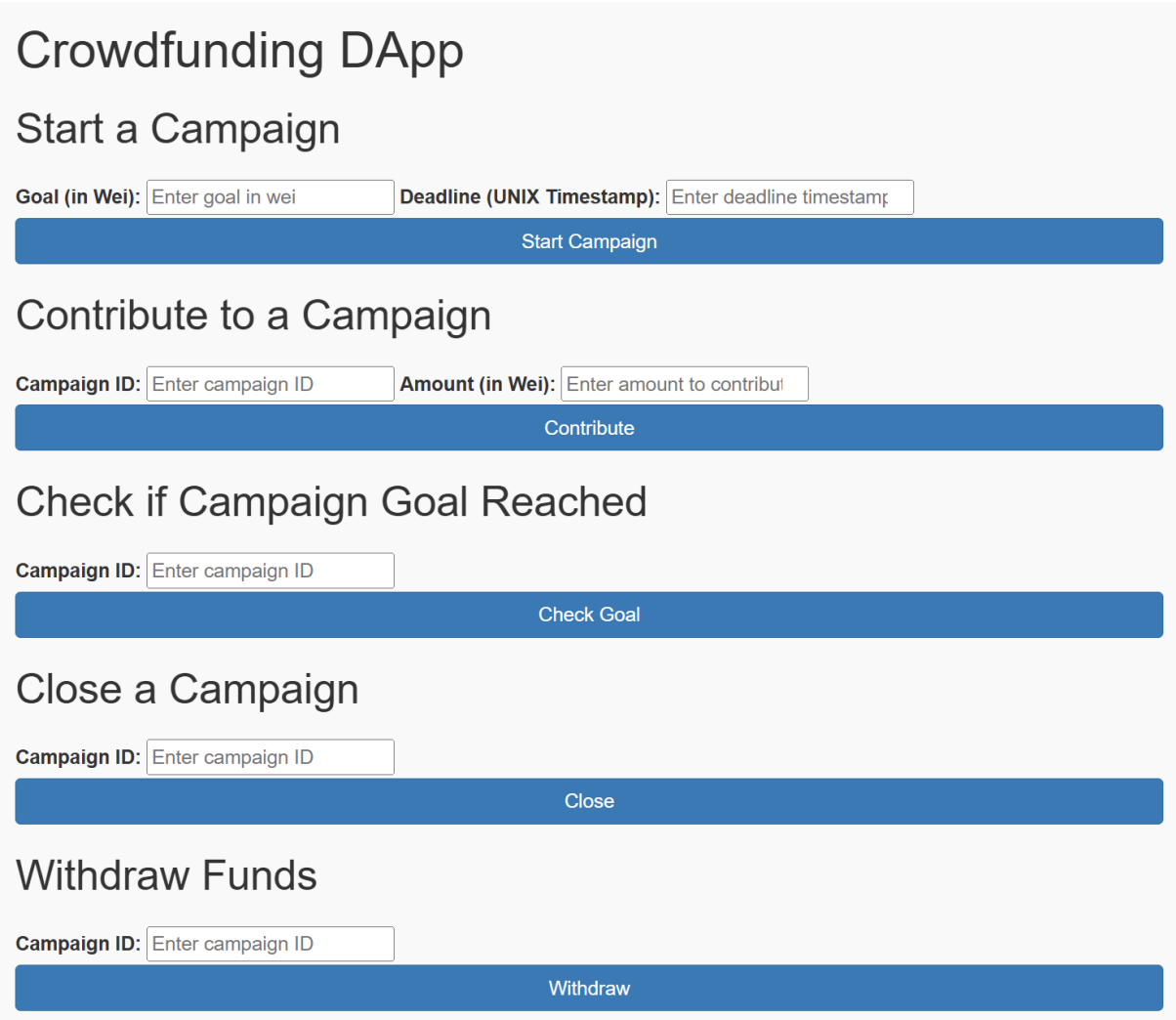
- Function to dialog with smart contract
    - The first screen is about starting a campaign, contributing to and check if the crowdfunding reaches the goal

```
startCampaignJs: function (event) :void {
    event.preventDefault();
    var doc;
    const goal = $('@ #goal').val();
    if (!goal) {
        alert('Please enter a goal');
        return;
    }
    const deadline = $('@ #deadline').val();
    if (!deadline) {
        alert('Please enter a deadline');
        return;
    }
    web3.eth.getAccounts(function (error, accounts) :void {
        if (error) {
            console.log(error);
        }
        var account = accounts[0];
        App.contracts.Crowdfunding.deployed().then(function (instance) {
            doc = instance;
            return doc.startCampaign(goal, deadline, {from: account});
        }).then(function (result) :void {
            $('@ #startCampaignResult').text( value: 'Campaign started successfully' + result);
        }).catch(function (err) :void {
            console.log(err.message);
        });
    });
},

contributeJs: function (event) :void {
    event.preventDefault();
    var doc;
    const campaignId = $('@ #campaignId').val();
    const amount = $('@ #amount').val();
    if (!amount) {
        alert('Please enter an amount');
        return;
    }
    web3.eth.getAccounts(function (error, accounts) :void {
        if (error) {
            console.log(error);
        }
        var account = accounts[0];
        App.contracts.Crowdfunding.deployed().then(function (instance) {
            doc = instance;
            return doc.contribute(campaignId, {from: account, value: amount});
        }).then(function (result) :void {
            $('@ #contributeResult').text( value: 'Contribution successful, balance:' + result);
        }).catch(function (err) :void {
            console.log(err.message);
        });
    });
},

checkGoalJs: function (event) :void {
    event.preventDefault();
    const campaignId = $('@ #campaignIdCheck').val();
    web3.eth.getAccounts(function (error, accounts) :void {
        if (error) {
            console.log(error);
        }
        var account = accounts[0];
        App.contracts.Crowdfunding.deployed().then(function (instance) {
            return instance.checkGoalReached(campaignId, {from: account});
        }).then(function (result) :void {
            if (result) {
                $('@ #checkGoalResult').text( value: 'Goal reached');
            } else {
                $('@ #checkGoalResult').text( value: 'Goal not reached');
            }
        }).catch(function (err) :void {
            console.log(err.message);
        });
    });
},
```

- The second screen is about closing and withdrawing a campaign

```
closeCampaignJs: function (event) :void {
    event.preventDefault();
    const campaignId = $('#campaignIdClose').val();
    web3.eth.getAccounts(function (error, accounts) :void {
        if (error) {
            console.log(error);
        }
        var account = accounts[0];
        App.contracts.Crowdfunding.deployed().then(function (instance) {
            return instance.close(campaignId, {from: account});
        }).then(function (result) :void {
            $('#closeResult').text( value: 'Campaign closed');
        }).catch(function (err) :void {
            console.log(err.message);
        });
    });
},

withdrawJs: function (event) :void {
    event.preventDefault();
    const campaignId = $('#campaignIdClose').val();
    web3.eth.getAccounts(function (error, accounts) :void {
        if (error) {
            console.log(error);
        }
        var account = accounts[0];
        App.contracts.Crowdfunding.deployed().then(function (instance) {
            return instance.withdraw(campaignId, {from: account});
        }).then(function (result) :void {
            $('#withdrawResult').text( value: 'Withdrawal successful');
        }).catch(function (err) :void {
            console.log(err.message);
        });
    });
}
```

Result:


Crowdfunding DApp
Start a Campaign
Goal (in Wei):
Enter goal in wei
Deadline (UNIX Timestamp):
Enter deadline timestamp
Start Campaign
Contribute to a Campaign
Campaign ID:
Enter campaign ID
Amount (in Wei):
Enter amount to contribut
Contribute
Check if Campaign Goal Reached
Campaign ID:
Enter campaign ID
Check Goal
Close a Campaign
Campaign ID:
Enter campaign ID
Close
Withdraw Funds
Campaign ID:
Enter campaign ID
Withdraw

Function tests: