The Charitable Blockchain

Crowdfunding Crypto-Philanthropy

Student:

Andrew Starling | 100191710

Instructors:

Dave McKay

Dhruvin Parikh

Course:

The Charitable Blockchain: Crowdfunding Crypto-Philanthropy BCDV1014

Problem | 01

In the charitable donation ecosystem, it is often the case that a small fraction of the funds donated, actually end up reaching the intended beneficiaries. Inefficient use of funds via excessive project management, strategy, fundraising, advertising and campaigning costs, and the risk of fraudulent activity, are significant problems within many charitable enterprises. For example:

> When a devastating earthquake leveled Haiti in 2010, millions of people donated to the American Red Cross. The charity raised almost half a billion dollars. … The Red Cross says it has provided homes to more than 130,000 people, but the number of permanent homes the charity has built is six. (Sullivan, 2015)

It would be bad enough if misuse of charitable donations was isolated, but it is common. And like the Red Cross example illustrates, the donation amounts unaccounted for are significant:

> Every year, Kids Wish Network raises millions of dollars in donations in the name of dying children and their families. Every year, it spends less than 3 cents on the dollar helping kids. Most of the rest gets diverted to enrich the charity's operators and the for-profit companies Kids Wish hires to drum up donations. (Taggart, 2013)

In a recent study of 2,100 adults released by the Better Business Bureau in the USA, it was found that:

> Only 19% of individuals say they highly trust charities … but 70% rate trust in a charity as essential before giving. … This suggests that improving trust could increase charitable giving." (BBB Give.org, 2019)

The blockchain has been referred to as a 'trust machine.' (The Economist, 2015) The implementation of an autonomous and distributed blockchain solution would effectively provide the trust and transparency needed, to ensure that charitable donations are used more efficiently. A blockchain solution would help restore faith in philanthropy, and trust in the systems delivering it.

Solution | 02

A solution avoiding some of the pitfalls of previous methods, like lack of transparency and unnecessary administration fees depleting the funds going to intended recipients, could utilize the Ethereum blockchain. Smart contracts, autonomously executing on the blockchain, would ensure the correct completion of agreements, rather than assuming that intermediate organizations would provide that functionality. Smart contracts would also reduce much of the resources currently expended on administration and accounting.

Using a computer-based decentralized application, a donor would select a charitable organization they want to support, and donate funds in Ether (future upgrade would allow for fiat currencies and other cryptocurrencies). The funds are pooled in a digital escrow account and held until the funds necessary to meet the selected goal are available, and the charitable project is verified as complete.

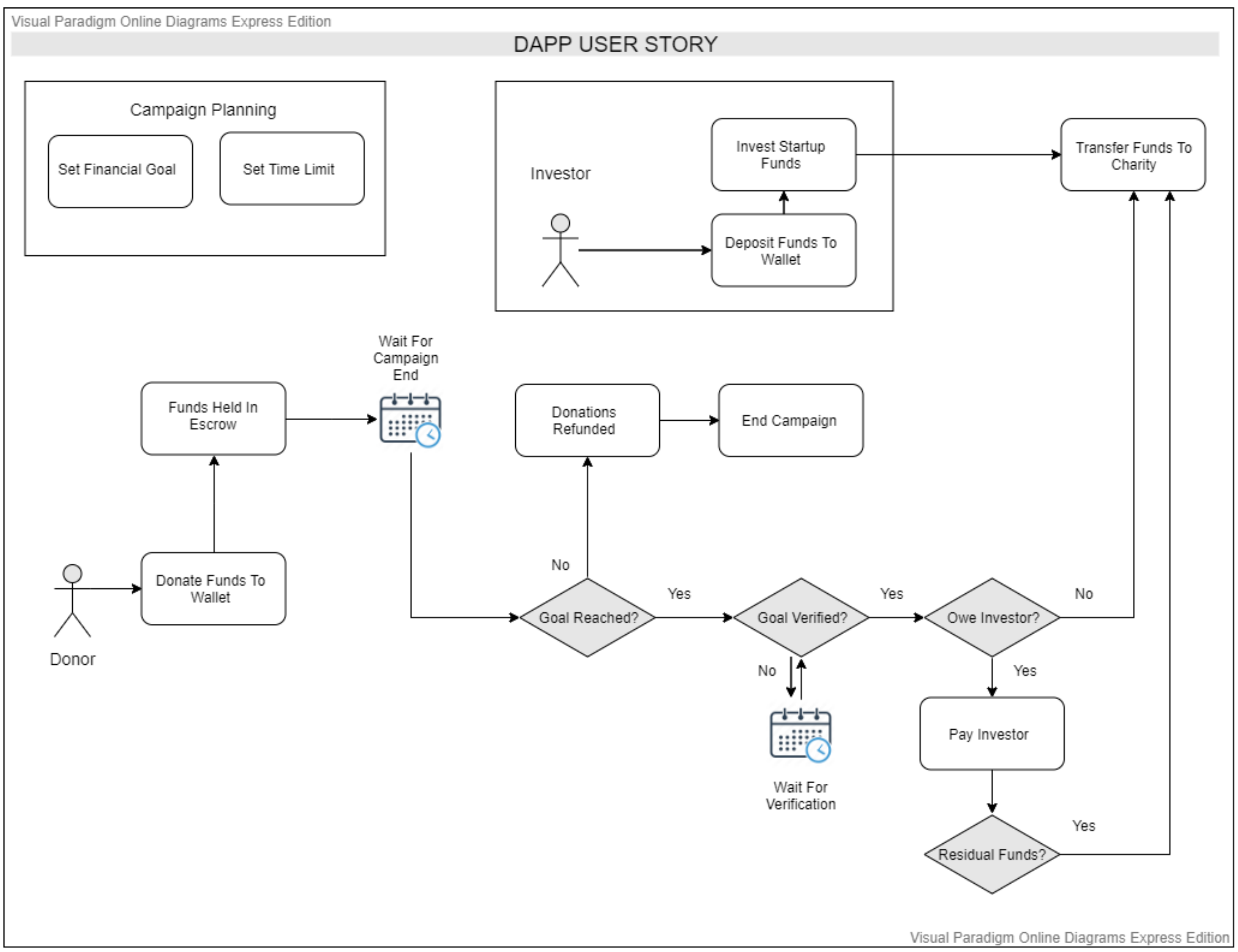
Upon project completion, as determined by independent auditors/oracles and smart contracts, funds would be released from escrow to cover seed capital investment, with excess funds supporting the charity.

For the donor, blockchain technology provides increased transparency and removes the necessity of having to trust intermediaries (trusting that your donation will actually get to the intended beneficiary as efficiently as possible). For a charitable organization, a blockchain mitigates concern about failures and fraudulent activity impacting the organization's public perception and integrity. Rather, trust is placed in 'the code' -- the smart contracts, and the underpinning blockchain technology, removing intermediaries, inefficiency and fraud. A blockchain solution would allow for a donor to transfer currency and trace that transaction, to the items purchased and services rendered for the recipients.

The increase in trust that blockchain technology provides, has the positive effect of a potential increase in donations. This helps direct more assistance to the charitable causes and ultimately the beneficiaries. The efficiency of the system and removal of intermediaries like banks and top-heavy aid organizations, would reduce costs and eliminate some transaction fees. Corruption and financial fraud would be severely curtailed, with smart contracts controlling the release of funds as project goals are met and verified.

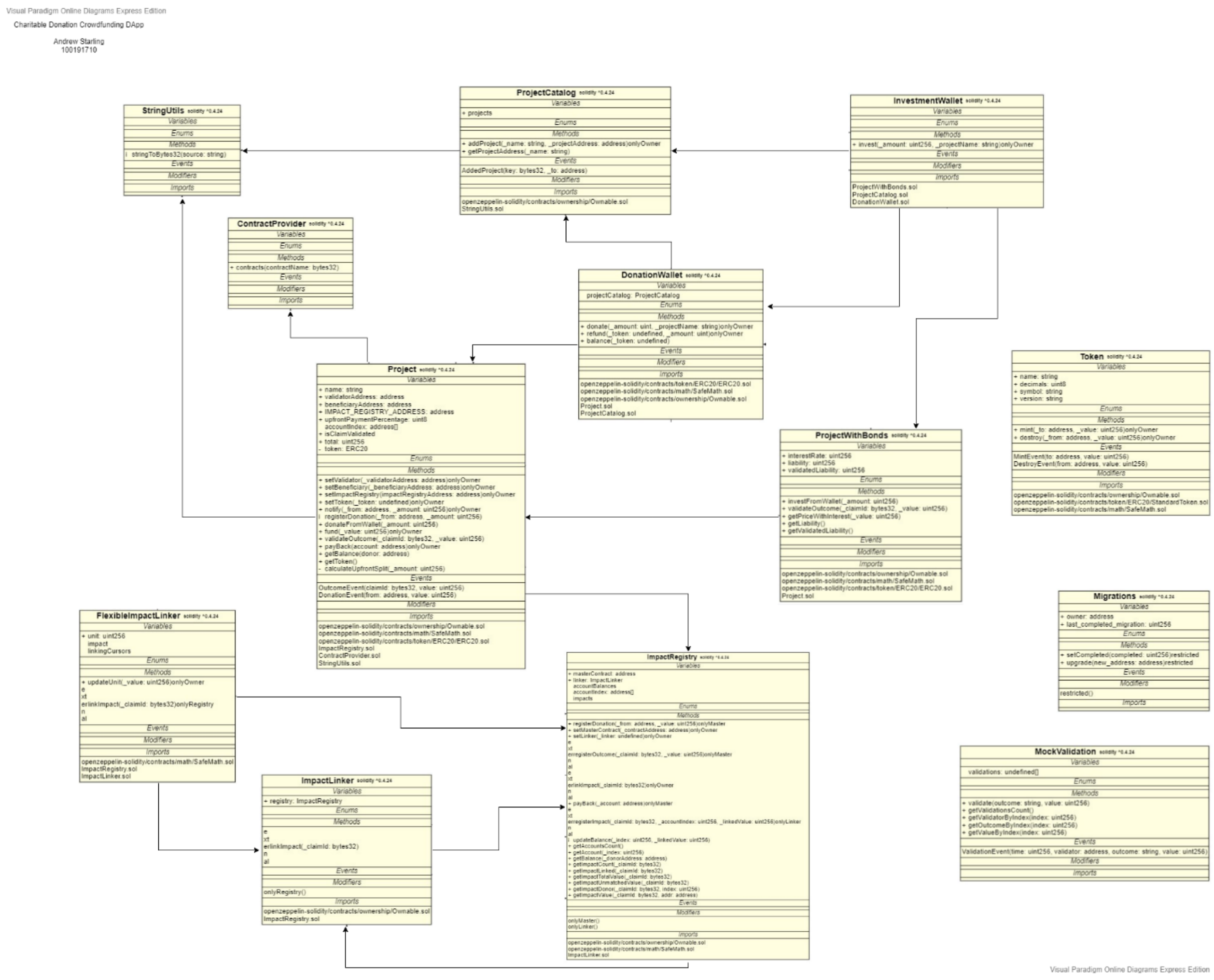
Analysis | 03

User Story



Architecture | 04

Smart Contract UML Diagram



Smart Contract Functions

The following tables provide an overview of the smart contract functions, the function modifiers enforcing the constraints related to its call, parameters, and a description of the function's code.

Token.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
mint	public		onlyOwner	address _to uint256 _value
Action – Notes: Add _value to totalSupply Add _value to balances[_to] Emit MintEvent				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
destroy	public		onlyOwner	address _from uint256 _value
Action – Notes: Subtract _value from totalSupply Subtract _value from balances[_from] Emit DestroyEvent				

ContractProvider.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
contracts	public			bytes32 contractName Returns address addr

DonationWallet.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
donate	public		onlyOwner	uint _amount string _projectName

Action – Notes:
Charitable project address set from a catalog of charitable projects.
address projectAddress = projectCatalog.getProjectAddress(_projectName);
The project cannot be address(0)
require(projectAddress != address(0));
ERC20 token = Project(projectAddress).getToken();
Approve token donation to a project
token.approve(projectAddress, _amount);
Donate all Ether from donor wallet to project
Project(projectAddress).donateFromWallet(_amount);

Function Name	Function Visibility	Function Type	Modifiers	Parameters
refund	public		onlyOwner	ERC20 _token uint _amount

Action – Notes:
Refund tokens
_token.transfer(owner, _amount);

Function Name	Function Visibility	Function Type	Modifiers	Parameters
balance	public	view		Returns uint256

Action – Notes:
Return token balance
return _token.balanceOf(this);

Project.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
setValidator	public		onlyOwner	address _validatorAddress
Action – Notes: validatorAddress = _validatorAddress				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
setBeneficiary	public		onlyOwner	address _beneficiaryAddress
Action – Notes: beneficiaryAddress = _beneficiaryAddress				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
setImpactRegistry	public		onlyOwner	address impactRegistryAddress
Action – Notes: IMPACT_REGISTRY_ADDRESS = impactRegistryAddress				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
setToken	public		onlyOwner	ERC20 _token
Action – Notes: token = _token;				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
notify	public		onlyOwner	address _from uint256 _amount

Action – Notes:
require(_from != 0x0);
require(_amount > 0);
registerDonation(_from, _amount);				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
registerDonation	internal		onlyOwner	address _from uint256 _amount

Action – Notes:
Future implementation where a percentage of donations go immediately to charitable projects, rather than being locked in escrow until validated.
(uint256 upfront, uint256 remainder) = calculateUpfrontSplit(_amount);
if (upfront > 0) {
getToken().transfer(beneficiaryAddress, upfront);
}
total = total.add(remainder);
ImpactRegistry(IMPACT_REGISTRY_ADDRESS).registerDonation(_from,remainder);
emit DonationEvent(_from, _amount);

Function Name	Function Visibility	Function Type	Modifiers	Parameters
donateFromWallet	public		onlyOwner	uint256 _amount

Action – Notes:
getToken().transferFrom(msg.sender, address(this), _amount);
registerDonation(msg.sender, _amount);				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
fund	public		onlyOwner	uint256 _value
Action – Notes: total = total.add(_value);				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
validateOutcome	public			bytes32 _claimId uint256 _value

Action – Notes:
require(msg.sender == validatorAddress);
getToken().transfer(beneficiaryAddress, _value);
total = total.sub(_value);
ImpactRegistry(IMPACT_REGISTRY_ADDRESS).registerOutcome(_claimId,_value);				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
payBack	public		onlyOwner	address account

Action – Notes:
uint256 balance = getBalance(account);
if (balance > 0) {
getToken().transfer(account, balance);
total = total.sub(balance);				

```
        ImpactRegistry(IMPACT_REGISTRY_ADDRESS).payBack(account);
    }
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getBalance	public	view		address donor returns uint256

Action – Notes:

```
return ImpactRegistry(IMPACT_REGISTRY_ADDRESS).getBalance(donor);
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getToken	public	view		returns ERC20

Action – Notes:

```
return token;
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
calculateUpfrontSplit	private	view		returns uint256 upfront, uint256 remainder

Action – Notes:

For future upgrade, where charity receives a percentage of donations upfront, rather than the entire amount being locked in escrow.

```
upfront = _amount.mul(upfrontPaymentPercentage).div(100);
remainder = _amount.sub(upfront);
```

ProjectWithBonds.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
investFromWallet	public			Uint256 _amount

Action – Notes:

```
require(getToken().transferFrom(msg.sender, beneficiaryAddress, _amount));
```

The investor is owed investment + interest for supplying startup capital.

```
liability = liability.add(getPriceWithInterest(_amount));
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
validateOutcome	public			bytes32 _claimId uint256 _value

Action – Notes:

Project validator must be the contract owner
require (msg.sender == validatorAddress);
Make sure validation amount is less than total
require (_value <= total);
Projects are only paid out from the Escrow account after validation.
Subtract the validated amount from the total of investor capital.

```
uint256 unvalidatedLiability = liability.sub(validatedLiability);
if (_value > unvalidatedLiability) {
  uint256 surplus = _value.sub(unvalidatedLiability);
  getToken().transfer(beneficiaryAddress, surplus);
  validatedLiability = validatedLiability.add(unvalidatedLiability);
} else {
  validatedLiability = validatedLiability.add(_value);
}
total = total.sub(_value);
ImpactRegistry(IMPACT_REGISTRY_ADDRESS).registerOutcome(_claimId, _value);
emit OutcomeEvent(_claimId, _value);
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getPriceWithInterest	public	view		uint256 _value Returns uint256

Action – Notes:
Investor receives 1% interest on their investment.

```
return _value.add(_value.mul(interestRate).div(10000));
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getLiability	public	view		Returns liability

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getValidatedLiability	public	view		Returns validatedLiability

ProjectCatalog.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
addProject	public		onlyOwner	string _name address _projectAddress

Action – Notes:
Add a new charitable project to the catalog of projects.

```
bytes32 nameAsBytes = _name.stringToBytes32();
```

```
require(projects[nameAsBytes] == address(0));
projects[nameAsBytes] = _projectAddress;
emit AddedProject(nameAsBytes, _projectAddress);
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getProjectAddress	public	view		string _name return projects[nameAsBytes]
Action – Notes: bytes32 nameAsBytes = _name.stringToBytes32();				

ImpactRegistry.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
registerDonation	public		onlyMaster	address _from uint _value

Action – Notes:
Register the donation from a new donor.
if (accountBalances[_from] == 0) {
accountIndex.push(_from);
}
accountBalances[_from] = accountBalances[_from].add(_value);
}				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
setMasterContract	public		onlyOwner	address _contractAddress
Action – Notes: masterContract = _contractAddress;				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
setLinker	public		onlyOwner	ImpactLinker _linker
Action – Notes: linker = _linker;				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
registerOutcome	external		onlyMaster	bytes32 _claimId uint256 _value
Action – Notes: impacts[_claimId] = Impact(_value, 0, 0);				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
linkImpact	external		onlyOwner	bytes32 _claimId

Action – Notes:

```
  linker.linkImpact(_claimId);
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
payBack	public		onlyMaster	address _account

Action – Notes:

```
accountBalances[_account] = 0;
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
registerImpact	external		onlyLinker	bytes32 _claimId uint _accountIndex uint _linkedValue

Action – Notes:

```
Impact storage impact = impacts[_claimId];
  address account = this.getAccount(_accountIndex);
  if (impact.values[account] == 0) {
    impact.addresses[impact.count++] = account;
  }
  require(impact.value.sub(impact.linked) >= _linkedValue);
  updateBalance(_accountIndex, _linkedValue);
  impact.values[account] = impact.values[account].add(_linkedValue);
  impact.linked = impact.linked.add(_linkedValue);
 }
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
updateBalance	internal			uint _index uint _linkedValue

Action – Notes:

```
  uint oldBalance = accountBalances[accountIndex[_index]];
  uint newBalance = oldBalance.sub(_linkedValue);
  accountBalances[accountIndex[_index]] = newBalance;
  if (newBalance == 0) {
    accountIndex[_index] = accountIndex[accountIndex.length-1];
    accountIndex.length = accountIndex.length - 1;
  }
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getAccountsCount	public	view		return accountIndex.length

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getAccount	public	view		uint _index return accountIndex[_index]

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getBalance	public	view		address _donorAddress return accountBalances[_donorAddress]

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getImpactCount	public	view		bytes32 _claimId return impacts[_claimId].count

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getImpactLinked	public	view		bytes32 _claimId return impacts[_claimId].linked

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getImpactTotalValue	public	view		bytes32 _claimId return impacts[_claimId].value

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getImpactUnmatchedValue	public	view		bytes32 _claimId
Action – Notes: return impacts[_claimId].value.sub(impacts[_claimId].linked);				

Function Name	Function Visibility	Function Type	Modifiers	Parameters

getImpactDonor	public	view		bytes32 _claimId uint index
Action – Notes: return impacts[_claimId].addresses[index]				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getImpactValue	Public	view		bytes32 _claimId address addr
Action – Notes: return impacts[_claimId].values[addr];				

ImpactLinker.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
linkImpact	external		onlyOwner	bytes32 _claimId

FlexibleImpactLinker.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
updateUnit	public		onlyOwner	uint _value
Action – Notes: unit = _value;				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
linkImpact	external		onlyRegistry	bytes32 _claimId

Action – Notes:

```
uint value = registry.getImpactTotalValue(_claimId);
uint linked = registry.getImpactLinked(_claimId);
uint left = value.sub(linked);
if (left > 0) {
  uint i = linkingCursors[_claimId];
  address account = registry.getAccount(i);
  uint balance = registry.getBalance(account);
  if (balance >= 0) {
    // Calculate impact
    uint impactVal = balance;
    if (impactVal > left) {
        impactVal = left;
```

```
        }
        if (impactVal > unit) {
           impactVal = unit;
        }
        registry.registerImpact(_claimId, i, impactVal);
        // Update index
        if (balance == impactVal) {
          i--;
        }
        uint accountsCount = registry.getAccountsCount();
        if (accountsCount > 0) {
          linkingCursors[_claimId] = (i + 1) % accountsCount;
        } else {
          linkingCursors[_claimId] = 0;
        }
```

InvestmentWallet.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
invest	public		onlyOwner	uint _amount string _projectName

Action – Notes:
Invest funds in a charitable project in order to provide seed capital.

```
address projectAddress = projectCatalog.getProjectAddress(_projectName);
require(projectAddress != address(0));
ERC20 token = ProjectWithBonds(projectAddress).getToken();
token.approve(projectAddress, _amount);
ProjectWithBonds(projectAddress).investFromWallet(_amount);
```

MockValidation.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
validate	public			string outcome uint256 value

Action – Notes:

```
Validation memory validation = Validation(now, msg.sender, outcome, value);
    validations.push(validation);
    emit ValidationEvent(
       validation.time,
       validation.validator,
       validation.outcome,
       validation.value
    );
  }
```

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getValidationsCount	public	constant		returns uint256 count
Action – Notes: return validations.length;				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getValidatorByIndex	public	constant		uint256 index returns address validator
Action – Notes: return validations[index].validator;				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getOutcomeByIndex	public	constant		uint256 index returns string outcome
Action – Notes: return validations[index].outcome;				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
getValueByIndex	public	constant		uint256 index returns uint256 value
Action – Notes: return validations[index].value;				

StringUtils.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
stringToBytes32	internal	pure	onlyOwner	string memory source returns bytes32 result

Action – Notes:
assembly {
result := mload(add(source, 32))
}				

Migrations.sol

Function Name	Function Visibility	Function Type	Modifiers	Parameters
setCompleted	public	restricted	onlyOwner	uint256 completed
Action – Notes: last_completed_migration = completed;				

Function Name	Function Visibility	Function Type	Modifiers	Parameters
upgrade	public	restricted	onlyOwner	Address new_address
Action – Notes: Migrations upgraded = Migrations(new_address); upgraded.setCompleted(last_completed_migration);				

Project Plan | 05

Environment Setup

This project requires node-js runtime.

A test network is required: Ganache simulates the Ethereum blockchain, which can then be used to develop a blockchain-based application locally. This local test network will process transactions instantly, and Ether can be distributed as needed.

In order to interact with smart contracts, they must be deployed on the network, which is configured in truffle.js.

To run the smart contracts on the network, the contracts must be compiled first. The Solidity files (.sol) are compiled into .json artifacts. These artifacts will be created in build/contracts/*.json. Contracts are referenced in the application with an import or require statement.

Launch the Ganache personal Ethereum blockchain with a workspace and accounts with sufficient Ether, and run truffle migrate. This deploys the contracts onto the default network running at 127.0.0.1:7545.

Webpack build: use webpack to compile the application and place it in the build/ folder. On the command line issue npm run dev to build the application and serve it on the localhost (http://127.0.0.1:8080).

Create Smart Contracts

Code will be built and tested using VS Code and Remix. Refer to the Architecture section for smart contract details and relationships.

Create Tests

Truffle uses the Mocha testing framework and Chai for assertions in order to create JavaScript tests of the Solidity smart contracts.

Build Front End Interface

The front end is constructed using HTML, CSS, JavaScript, and various image files.

The main UX files are: index.html, app.css and app.js.

Various packs are also used:

Bootstrap 4.4.1
Ionicons 5.0.0
Google Material Icons
Font Awesome
AdminLTE
jQuery
Popper.js

Source Code | 06

Files are available at the following GitHub repository:

https://github.com/tobogganhill/charity

Conclusion | 07

The intersecting of blockchain technology and charitable fundraising can positively transform philanthropy. Sending charitable donations via the blockchain would disrupt the current state of fundraising, and provide accountability and transparency for all stakeholders.

Blockchain technology is in its early days, similar to the state of the internet in the mid-1990s. Understanding and adoption by the general public are in their infancy. As 'blockchain' is more accepted in society, people will have more faith in its underlying value. Long-term, it will most likely take years before charitable fundraising is implemented to any great extent on the blockchain. Currently, very few transactions in the public sphere are done using cryptocurrency. Although people's adoption of token-based solutions would certainly increase when it is realized that by removing the layers of intermediaries in the current system, accountability would vastly improve. Future upgrades to this application would make use of a stablecoin, which would help address unpredictability when dealing with more volatile cryptocurrency. In the future with more use cases, people will realize that they can put their trust in blockchain technology, and there will be less reluctance to embrace it. When this state is realized, a blockchain solution will help charitable fundraising positively transform the altruistic goals of philanthropy.

References | 08

BBB Give.org. (2019, November 14). *Fewer Than One in Four Donors "Highly Trust" Charities Says New Study by BBB's Give.org*. Retrieved from BBB Give.org: http://www.give.org/news-updates/2019/11/14/fewer-than-one-in-four-donors-highly-trust-charities-says-new-study-by-bbb-s-give.org

Sullivan, L. (2015, June 3). *In Search Of The Red Cross' $500 Million In Haiti Relief*. Retrieved from npr : https://www.npr.org/2015/06/03/411524156/in-search-of-the-red-cross-500-million-in-haiti-relief

Taggart, K. H. (2013, June 13). *Above the law: America's worst charities*. Retrieved from CNN: https://www.cnn.com/2013/06/13/us/worst-charities/index.html

The Economist. (2015, October 31). *The trust machine*. Retrieved from The Economist: https://www.economist.com/leaders/2015/10/31/the-trust-machine