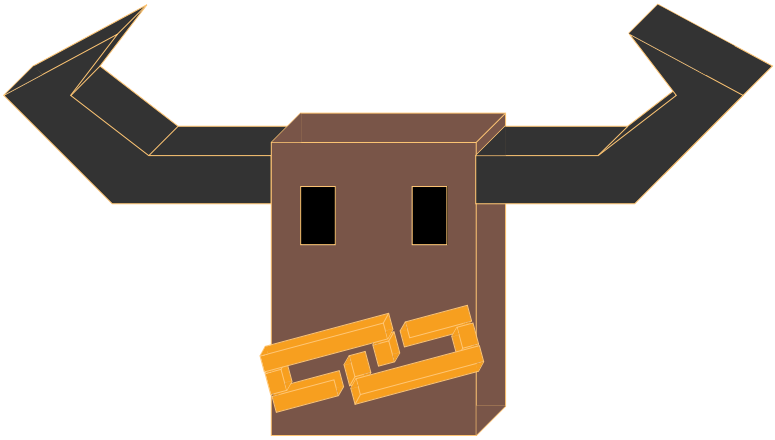
Chainge

An Incentive System to Save the Planet

Patrick Züst, David Schmid, Nico Hauser, David Rode, Ben Spöttling

Ethereum-powered Đapp that allows for impactful donations with a money back guarantee, decentralized storage of sensor data and active citizen participation.



Patrick Züst: zuestp@student.ethz.ch
David Schmid: schmidav@student.ethz.ch
Nico Hauser: hausern@student.ethz.ch
David Rode: rodeda@student.ethz.ch
Ben Spöttling: bensp@student.ethz.ch

All members contributed equally to this report

Contents

1 Introduction . . . 2
1.1 Challenge Context . . . 2
1.2 Platform Nodes . . . 2
1.2.1 Donor/Investor Node . . . 2
1.2.2 Ecosystem Node . . . 3
1.2.3 Hunter/Gatherer Node . . . 3
1.2.4 Community Node . . . 3
2 Challenge Solution . . . 4
2.1 Overview . . . 4
2.2 Donors . . . 4
2.3 Gatherer . . . 5
2.4 Community . . . 5
2.5 Ecosystem . . . 6
2.6 Concepts for action proofing . . . 6
2.6.1 Naive poofing . . . 6
2.6.2 Authority proofing . . . 6
2.6.3 Crowd proofing . . . 6
2.6.4 Sensor maintenance proofing . . . 6
2.6.5 Sensor based evidence . . . 7
2.6.6 AI assisted proofing . . . 7
2.6.7 Reputation assisted proofing . . . 7
2.6.8 Combinations . . . 7
3 Solution Design (technology) . . . 7
3.1 Hardware . . . 7
3.2 Software . . . 8
3.2.1 Smart Contract(s) . . . 8
3.2.2 Sensor data / Campaign goals . . . 12
3.2.3 Frontend . . . 12
4 Evaluation . . . 13
4.1 Blockchain platform choice . . . 13
4.2 Campaign administration . . . 14
4.3 Smart contract implementation . . . 14
5 Conclusion . . . 14

1 Introduction

Preserving nature and limiting human impact on the environment are some of the most important global challenges today. However, the planet can not be saved in one day alone. Environmental campaigns oftentimes fail not due to limited funds, but because the local population only receives limited short-term rewards for acting ecologically. Additionally the measurement of the impact of such campaigns is extremely complicated and untransparent. Using blockchain technology, we propose a system that offers incentives to citizens for actively participating in the preservation of the environment they live in. Donations are processed in a decentralized way on the blockchain and are therefore completely transparent. Furthermore, the donations are reimbursed in case they do not have the desired impact. In this report we mainly focus on the example of a campaign proposed by the World Wildlife Fund (WWF) as a challenge for the hackathon. The goal of our blockchain solution is to receive higher donations by allowing for a transparent processing of the payments, and provide incentives to change the behaviour and attitude of the local population in order to create a sustainable and stable ecosystem.

1.1 Challenge Context

The "Wild AI" challenge is built around a nature and wildlife preservation project by WWF in the Carpathian mountains in Romania. The Carpathian mountain range is one of the largest forests in Europe and home to many species including the European bison. The Carpathians are one of the few ecosystems in Europe which have a sizable bison population. Although this species has recovered from near-extinction, it is still classified as an endangered species. Therefore the WWF has declared the goal to create a better environment for the European Bison, so that its population continues to grow. Instead of just supporting bison rewilding sites or financing reforestation efforts, the WWF envisions to include the local community in their efforts and make them more conscious towards the environment. This project intends to realize these ambitious goals by employing modern blockchain technology using smart contracts to handle the donations, the project goals, to monitor the progress and to automate the distribution of rewards and compensation to participating locals.

1.2 Platform Nodes

The WWF splits the Project into four distinct nodes which should be covered in an ideal solution. These nodes are explained in the following:

1.2.1 Donor/Investor Node

Contrary to existing nature preservation projects, this project also heavily focuseson the local community. In a day and age of instant information transfer, charities are still lacking behind. Therefore we plan on providing donors with accuratereal-time data to show them the effects of their donations to encourage further involvement and donations. We want to go even one step further: In case an impact goal is not fulfilled in time, refunds will be granted. Furthermore, an interactive website will allow donors to witness the progress of the program from the comfort of their home. Maps will show improvements to the environment, cameras will show free roaming bison herds and sensor data will document progress over time as validation.

Fig. 1: Plattform nodes as defined by WWF

1.2.2 Ecosystem Node

By using IoT technology and by engaging local citizens, the WWF intends to improve the health of the ecosystem in the Carpathians. Individuals contribute to conservation efforts and receive appropriate compensations in form of cash and voting tokens. Sensors and locals collect data, providing valuable insight into the progress of conservation efforts. This data is intended as a proof for donors.

1.2.3 Hunter/Gatherer Node

The WWF wants to encourage a more ecological lifestyle within the population in the Carpathians. An important step in achieving this goal is providing locals with worthwhile alternatives to their current interactions with the environment. Instead of poaching, cutting down trees and burning firewood, locals should be compensated for environment-friendlier actions. This could include involving locals in conservation projects by offering paid jobs, such as wildlife photography, reforestation efforts, etc. Over time a healthy ecosystem will be linked to personal gains and community benefits, encouraging a lasting change of perception.

1.2.4 Community Node

By supporting the efforts of the WWF, individuals can earn community tokens. These tasks could include anything from conservation work to data collection. After providing proof of their work, they are immediately compensated monetarily for their work via the blockchain system. Additionally they receive community tokens, which represent voting rights. Locals can vote on several different community investments using their tokens. At the end of the voting period, the funds are allocated to the projects with the most votes. By implementing this system, locals are incentivized to perform more task. It also ensures that funds towards the community are utilized in the most efficient way, as locals themselves decide which of their needs are more urgent.

2 Challenge Solution

2.1 Overview

We developed the *chainge* Platform, which covers the whole project cycle of environmental campaigns like the one proposed by WWF. Such campaigns are more successful, if they are supported by local citizens. *chainge* campaigns thus allocate only a certain part of the campaign's total budget to environmental preservation measures. Remaining funds go towards rewarding citizens for their participation and supporting local communal initiatives, if the previously defined impact goals are met. This is secured by blockchain technology and checked by sensors that constantly monitor different environmental factors. Each campaign consists of three stages:

Immediately after the smart contract is submitted to the blockchain, the donation period starts. Donors can read information about the campaign online and donate Ether to support the cause. After a certain period of time, the campaign is initiated. Citizens now have the opportunity to complete actions, which were defined by the campaign initiator. They then get rewarded with Ether and special voting tokens. Sensors are monitoring changes of environmental factors and store those values on the blockchain. At the end of a campaign, it is automatically checked, whether the impact goals were met. If that is not the case, the remaining funds are returned to the donors proportionally to their initial investment. However, if the community was able to achieve the goals of the campaign, the voting phase starts: Active gatherers can allocate their voting tokens towards different community initiatives and the remaining funds are split among those projects according to the token distribution.

The features of *chainge* are presented in regards to the nodes of the WWF diagram detailed in . As this project is only a prototype of a much bigger vision, some ideas for further improvements are listed, too.

2.2 Donors

- Information about a campaign and specific impact goals are displayed on the website. Those details are stored in a blockchain and it is hence impossible to change them retrospectively.
- Donors can make an Ether donation to the campaign, if they are convinced of the specific parameters and impact goals. The funds are then stored in a decentralized smart contract and can therefore only be used for their intended purpose. This can easily be verified by donors as well as regulators - there is no need to trust a middleman.
- A previously defined split of the donations is automatically sent to the initiator of the campaign. It is however not possible to claim any other donations.
- In case the campaign goals are not met, the remaining funds are reimbursed automatically to the donors. This is a reassurance for them but also an incentive for the citizens and the project initiator to meet said goals.

Further development

- Donors could be allowed to explicitly state how much of their money should go towards the environmental campaign and how much towards the community project. This is currently implemented as a campaign property, which can be set by the campaign initiator once. As a flexible split may lead to other issues, such as most donors only donating money to

the environmental campaign, this feature would require further discussions and probably a study on the donating behaviour.
- With the data gathered during a campaign, it would be possible to automatically generate regular updates for donors. As all values are stored on the blockchain anyway, they could be loaded by the frontend (website) and displayed there. Showing live-values of all quantifiable impact goals would give the donors a better insight and make the whole campaign even more transparent.

2.3 Gatherer

- The project initiator is able to define positive actions that the citizens can take to support the campaign. Each action automatically rewards a predefined amount of voting tokens and Ether to the citizen after successful completion.

Further development

- Data submitted by gatherers is currently not processed, as we are only working with test data. As this data can be quite large, it may be infeasible to store everything on the blockchain directly. One solution would be to use an existing decentralized storage solution such as the IPFS to achieve better performance.
- There is currently no verification system in place to check the submitted actions. This check could be performed either by the campaign initiator or (preferably) by other community members. This would demand a reputation score system: The trustworthiness of citizens decreases, if they consciously make false claims or if they verify claims that turn out to be false.
- Certain actions can be verified automatically by using technologies like image recognition and statistical analysis.

2.4 Community

- If the impact goals are met, active community members can participate in a vote about the allocation of the remaining funds. They can distribute their tokens to different projects according to their preferences. Tokens can not be traded to prevent fraudulent use.
- The smart contract automatically transfers the funds to the communityprojects proportionally to the allocated voting tokens. Neither thecampaign initiator nor the donors can interfere in this process.
- Blockchain technology is keeping track of every action and awarded token. Fraud and hacks are effectively prevented.

Further development

- Community projects can currently only be initiated by the contractowner. In a future version, it might be possible for community membersto propose new projects, while the campaign is running. To preventabuse, it would be necessary to verify these projects - eithercentrally by the campaign initiator or alternatively by the communitymembers and/or the campaign donors.

2.5 Ecosystem

- Sensors are sending data in regular intervals to the blockchain, where they are stored decentralized. The data is immutable and manipulations are therefore not possible.

Further development

- Sensor data is currently stored on chain which leads to high transaction costs. Similarly to the proof data, it could alternatively be stored using existing decentralized storage solutions such as the *IPFS*.

2.6 Concepts for action proofing

A core problem is to make sure, that gatherers can't cheat the system to get rewards without actually fulfilling actions. Therefore, proofing mechanisms have to be implemented, at best directly on the blockchain. Because different proofing methods are required for different actions, the *chainge* platform is built so that arbitrary functions can be added as prove functions. The general workflow will always be the same. After completing the action, the gatherer submits proofs, collected data and comments to the *chainge* platform. This data is then passed to the predefined proofing function for evaluation. The result determines whether tokens are awarded. Different ideas for proving functions are evaluated in the following.

2.6.1 Naive poofing

This method was implemented for the development process to accelerate testing. There is no verification at all and the action is set as valid automatically. It is not recommended to use this in production, but it can be handy for demonstration purposes.

2.6.2 Authority proofing

The campaign initiator or a delegate of his is probably best suited to decide if their task have been properly fulfilled. This is a simple method and useful for small projects, but it defeats the concept of trust-less systems and creates a lot of work for the initiator.

2.6.3 Crowd proofing

Instead of one authoritative entity, a large collective is randomly selected as jury. This jury can either consist of community members or donors. For every action submission, another jury should be selected. It is presented with the task description and the submitted evidence.If a certain percentage of the jury determines the evidence sufficient, the evaluation is successful.

2.6.4 Sensor maintenance proofing

A lot of deployed sensors requires a central management system to monitor their health. If an action to do maintenance work was created based on the health monitoring data, this action would get verified automatically when the health data improves. This method needs no external authority an can be fully automated, but it is only applicable to actions which create directly measurable results.

2.6.5 Sensor based evidence

A sensor equipped with GPS or cameras could prove the presence of a gatherer nearby the sensor location. This most likely doesn't directly prove the action, but it automatically creates supplementary evidence which can be used by any other verification method.

2.6.6 AI assisted proofing

To reduce the amount of human involvement, a machine-learning algorithm could first check the evidence and decide whether it can automatically approve or pass on to a manual verification method. For AI to work properly, standardisation and preprocessing of the input data is highly recommended and a lot of training data is indispensable. The AI would be designed to take the task and provided evidence as input and return a confidence score. During training phase, this score is compared with the outcome of the conventional proofing method to improve the AI system. It is important to note, that successful implementation of AI needs high quality training and close monitoring to prevent the acquisition of biased behaviour from the training data.

2.6.7 Reputation assisted proofing

In addition to other proofing mechanisms, a reputation score can be assigned to every gatherer for successfully verified actions. This will ease the verification process for this user on further actions. On the other hand, bad reputation following a fraud attempt will result in more meticulous checking. A reputation score would also play well with the AI assisted verification.

2.6.8 Combinations

Of course a campaign isn't limited to the use of just one proofing mechanism. It is encouraged to use different proofing mechanism for the different types of actions of a campaign. Depending on the action it may also be beneficial to use a combination of these proofing mechanism ideas.

3 Solution Design (technology)

3.1 Hardware

Sensors play a vital role during a campaign. They independently gather data that allows initiators and donors to track the progress of the campaign. This will ultimately determine whether it was a success and allow for the payout of rewards to the community. The accuracy of sensors and the secure storage of data are therefore crucial to ensure a transparent and fair campaign.

The *chainge* prototype consists of a standard Raspberry Pi processor and currently contains GPS-, temperature- and humidity-sensors for both air and soil.

It can easily be equipped with additional sensors like air quality sensors, cameras, microphones and smoke detectors to provide more precise monitoring of the local biosystem in the future. As the Raspberry Pi is very modular, defect parts can easily be replaced in the field.

Cost is a major factor for public foundations, therefore we tried to make the system as cost-effective as possible. The individual components of the prototype are regular off-the-shelf consumer products. The plastic casing was manufactured by additive manufacturing at a low price. The heart of the prototype is a Raspberry Pi single-board computer; the sensors are connected via cable connections and a breadboard. No soldering is required to build the

system, which allows for easy assembly. Besides a 3D-printer hardly any tools and machinery are needed. It is possible for local communities to build the entire device at a cost of less than 70$ without any lengthy training. If they were manufactured in larger quantities, the price would be considerably lower.

Sensor input is processed by a Python program and then sent to the blockchain using Node.js. The current prototype has to be plugged in to the power system at all times and connects to the internet via WIFI. However, the electricity of future versions could be supplied by batteries. To achieve further self-reliance solar panels could be attached to charge the battery during sunny hours.Data would ideally be transferred over a LoRaWAN network. This would ensure high mobility and self-reliance without compromising connectivity.

The prototype has QR-codes placed at the inside and the outside of its case. When checking sensors for functionality, performing service or replacing batteries, local workers would scan these codes with their smartphones to update the status of the respective system and as proof of their work to obtain a compensation.

A revolutionary part of this campaign is the possibility for donors to track the progress of campaigns in real time. To get donors even more involved in the campaign and feel personally committed, some systems could be fitted with webcams providing live feed of interesting locations eg. bison habitats.

3.2 Software

3.2.1 Smart Contract(s)

Individual campaigns can differ largely in terms of citizen actions, impact goals and different sensor data. It is very inconvenient to predefine a set of options and lock the whole system by only deploying one single smart contract and operate on it for the rest of eternity. As technology evolves quickly, it is important for a system to be agile and adaptable, but at the same time provide the trust of immutability that blockchains and smart contracts offer. An easy solution to this problem is to not have a single smart contract handling all campaigns, but instead deploy different smart contracts for different campaigns. As campaigns have a predefined runtime, the technology used for a certain smart contract should not get outdated in a way that requires one to update the contract. Having multiple smart contracts with a specified lifetime ensure that the terms do not have to be changed and it limits the impact of possible design flaws. This way, the smart contract cannot be mutated by any party once it is deployed. It justifies the use of the term “contract”, as all users, especially the donors, know exactly, what will happen with their money, if they examine the smart contract.

Because of this design choice, it is possible to write a default `payable` function, which accepts all Ether that is transferred to the contract address. The smart contract also stores information about who donated how much.

Once the donation phase is over, the `startCampaign` function can be called.

```
// default function that processes donations, which are sent to the contract
function() external payable {
    require(donationInProgress, "donation not in progress");
    if(donorsAmount[msg.sender] == 0) {
        donors.push(msg.sender);
    }
    donorsAmount[msg.sender] += msg.value;
}
```

Fig. 2: `payable` function

If the defined `runTimeDonations` is over, a certain percentage of the current balance is transferred to the owner of the contract, which in our case is the WWF. The share that goes to the contract owner is stored in `ratioProject`. It is also calculated, how much every single donor has contributed (percentage-wise) to the whole pot. This will later be used to refund part of the money, in case the campaign goals are not met.

```
// forwards a share of the funds to the campaign initiator and starts the campaign
function startCampaign() external {
    require(donationInProgress, "donation not in progress");
    require(now >= startTimeDonations + runTimeDonations, "too soon, donations time not yet up.");

    _calculateDonorsShare();
    startTimeCampaign = now;
    donationInProgress = false;
    campaignInProgress = true;

    uint forOwner = (address(this).balance / 100) * ratioProject;
    owner.transfer(forOwner);
}
```

Fig. 3: `startCampaign()` function

One of the limitations of the blockchain is that we cannot automatically call `startCampaign`, when the runtime for the donation phase is over. This issue can be mitigated by having a cronjob running on the server, which will automatically call this function on the blockchain, as soon as it is time.

When the actual campaign phase begins, the community can start working on the so called *actions*. Actions can only be added by the contract owner, as in our case we only want the WWF to define options for citizen participation - and the rewards, which are associated with them. This is done with the `createAction` function.

Actions can of course be suggested by the citizens themself, but it should not be possible to exploit the system by adding irrelevant actions.

```
// contract owner can define actions, which award ether / tokens to citizens
function createAction(string memory _actionTitle, string memory _actionDescription, uint _actionReward, uint _actionProovingType) public{
    require(msg.sender == owner, "only campaign owner is allowed to create actions");
    require(address(this).balance > 0, "no balance");
    actions.push(Action(_actionTitle, _actionDescription, _actionReward, false, _actionProovingType, false, "", address(0)));
}
```

Fig. 4: `createAction()` function

As there can be many different types of actions, which possibly need to be proven in different ways, all actions contain a `provingType` property.

Everybody can submit proof for the fulfillment of an action, if the action has not been completed yet. Additional information can be submitted as a string parameter. Depending on the requirements of the campaign, this can also be changed to a byte array, but a string is not at all limited to text. It is possible to encode arbitrary bytes inside a string, for example by using Base 64.

```
// citizens claim that they fulfilled an action and send data to prove it
function submitAction(uint _actionId, string memory _actionSubmissionData) public {
    require(actions[_actionId].done == false, "action is already done by other user");
    actions[_actionId].user = msg.sender;
    actions[_actionId].submissionData = _actionSubmissionData;
    actions[_actionId].done = true;
}
```

Fig. 5: `submitAction()` function

Once a proof is submitted, it needs to be verified. During the limited time at the hackathon, only a mock proving type (denoted with id 0) was actually implemented; it always returns true.

```
// voting tokens and Ether are rewarded for verified actions proportionally to its complexity
function _getReward(uint _actionId) internal {
    Action memory action = actions[_actionId];
    gatherersToken[action.user] += action.reward;
    action.user.transfer(action.reward * paymentBaseUnit);
}
```

Fig. 6: `_getReward()` function

If the submission is verified, the property verified is set to true, a small immediate reward is transferred to the citizen that submitted the proof and a defined amount of `gatherersTokens` are added to the citizen's address. The tokens can be used to vote on the different community project, as soon as the campaign is over. This function has again to be called manually, as the ethereum blockchain itself does not support cronjobs.

As soon as the variable `votingInProgress` is set to true, citizens with `gatherersTokens` can call the `vote` function to decide on the community project(s) to support. If the campaign goals are not met, the remaining pot will be refunded to the donors in respect to their individual contributions.

```
// when campaign stage is over, this function checks the completion of the impact goals
// and proceeds accordingly
function startVoting() external {
    require(campaignInProgress, "campaign not in progress");
    require(now >= startTimeCampaign + runTimeCampaign, "too soon, campaign time not yet up");
    if(_impactGoalsAchieved() == false) {
        _refund();
    } else {
        startTimeVoting = now;
        campaignInProgress = false;
        votingInProgress = true;
    }
}
```

Fig. 7: `startVoting()` function

```
// voting for community projects
function vote(uint _communityProjectId, uint _voteCount) public {
    // check if voting is in progress
    require(votingInProgress, "Voting not in progress");

    // require that they haven sufficient votingTokens
    /* msg.sender: address of the function caller */
    require(gatherersToken[msg.sender] >= _voteCount, "The sender doesn't have enough tokens!");

    // require a valid CommunityProject
    require(_communityProjectId >= 0 && _communityProjectId < communityProjects.length, "The given CommunityProject id is invalid!");

    // reduce votingToken count
    gatherersToken[msg.sender] -= _voteCount;

    // update CommunityProject vote Count
    communityProjects[_communityProjectId].voteCount += _voteCount;

    voteCountTotal += _voteCount;

    //trigger vote event
    emit votedEvent(_communityProjectId);
}
```

Fig. 8: `vote()` function

`endVoting` is the last function that has to be called from outside of the blockchain after the voting period has ended.

```
// terminates the voting phase and triggers the payout to community projects
function endVoting() external {
    require(votingInProgress, "voting not in progress");
    require(now >= startTimeVoting + runTimeVoting, "too soon, voting time not yet up");
    votingInProgress = false;
    //pay out according to tokens invested
    payout();
}
```

Fig. 9: `endVoting()` function

The remaining pot is then distributed to the different community projects, proportionally to the votes they received by the community with the `payout` function.

```
// releases fund to community projects proportionally to the voting tokens received
function payout() private{
    for(uint i = 0; i < communityProjects.length; i++) {
        CommunityProject memory communityProject = communityProjects[i];
        uint votesReceived = communityProject.voteCount;
        uint share = (votesReceived * precision) / voteCountTotal;
        uint payOutAmount = (voteCountTotal / precision) * share;
        communityProject.account.transfer(payOutAmount);
    }
}
```

Fig. 10: `payout()` function

Further Development As the smart contract should mainly be a blueprint for different campaigns, it is definitely required to refactor the code and possibly add even more comments for easier understanding. It would also be nice to have some more examples for possible campaign goal calculations or proof types to illustrate the use cases, which are possible with this code base.

3.2.2 Sensor data / Campaign goals

In order to check, whether the campaign goals were met, we built the sensor described in the section 3.1 Hardware. In order for the sensor to be able to submit data to the blockchain, the respective smart contract first needs to register the address of the sensor. That is because the function `saveData` only accepts data from verified addresses. If there are multiple sensors, this could of course also be a mapping from allowed addresses to a boolean.

Further Development The set of data points is currently defined exactly for the sensor we built. It can of course be adjusted for different campaigns, but it may also be possible to find a data structure for storing all kind of sensor data. As discussed before, other types of decentralized storage systems could be employed here as well.

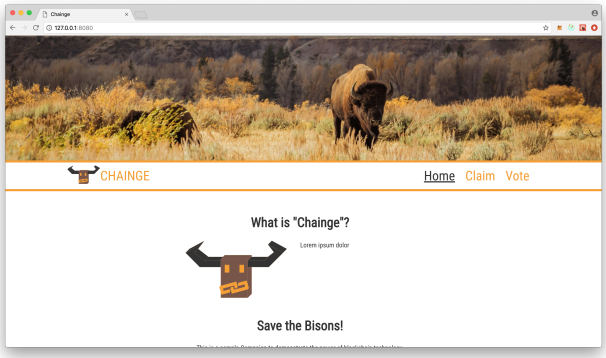
```
// the contract owner is able to set the addresses of the sensor modules that are allowed to send data
function setSensorAccount(address _sensorAccount) public {
    require(msg.sender == owner, "you are not the owner of the contract");
    sensorAccount = _sensorAccount;
}
```

Fig. 11: `setSensorAccount()` function sets the permission to write sensor data to the blockchain

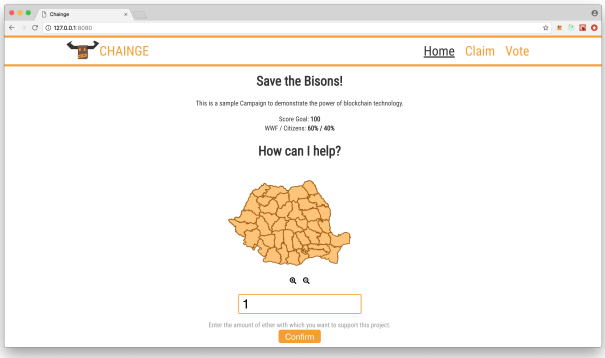
3.2.3 Frontend

The frontend is built in React and retrieves all data directly from the blockchain by using web3.js. As the use of web3.js requires an account, the chrome extension *MetaMask* is required for accessing the website itself, otherwise we wouldn't be able to load any information from the blockchain. As we store almost all data there, the website would be totally useless without having *MetaMask* installed. That is also the reason that a popup tells you to install *MetaMask*, if it is not already active. When the page is ready, the campaign data is loaded from the

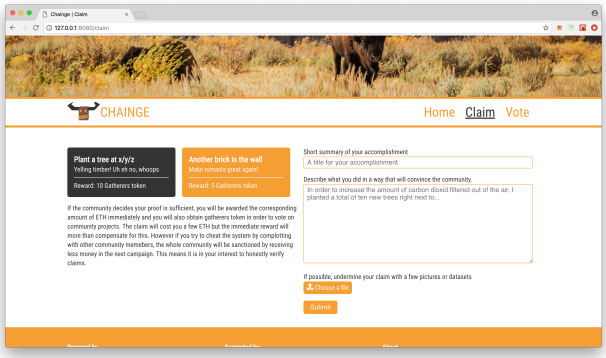
blockchain. As reading data from smart contracts does not cost Ether, there is no transaction and no confirmation necessary. However, as soon as a user tries to donate money to a campaign or use his voting tokens, a *MetaMask* popup shows up, which will handle and confirm the respective transaction to the blockchain.



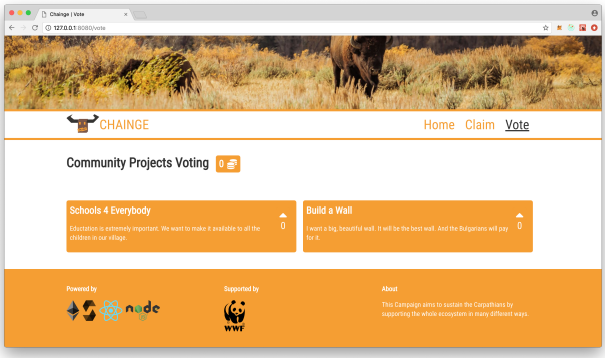
(a) The frontpage



(b) The dontation field



(c) Where claims can be made



(d) Where votes can be casted

Further Development The user experience could be improved by rendering the website server-side, which would allow users to browse the site and look at campaign data without having *MetaMask* installed. Further it would be possible to listen for certain events in the smart contract to continuously refresh the data, if it changed, so the data display on the page is always correct and does not require a manual page refresh by the user.

4 Evaluation

4.1 Blockchain platform choice

The *chainge* system is built on the Ethereum platform because of its popularity and its availability of development tools. There are both advantages and disadvantages of using a widespread, multipurpose blockchain like Ethereum.

First, it is relatively easy to aquire Ether (the virtual currency on Ethereum) for donors and there are also many options to use or exchange it back to fiat currencies. There are a lot of full nodes, which ensure the decentralization and therefore the immutability of the blockchain platform. On the other hand, the funds are subject to the volatile exchange rates of Ether, which sometimes behave rather unpredictable. There are also significant transaction fees for every

write to the blockchain. We should not forget the ongoing public controversy around high energy consumption of "proof-of-work" based networks. Although Ethereum currently transitions to a proof-of-work based consensus mechanism, environmentally focused organisations like WWF might want to avoid this problem.

One option would be to create a dedicated blockchain network, optimized for its usage. Because a high number of independent miners is required to prevent a rollback attack, an option is to create a *charity-chain* together with other organisations interested in deploying similar smart-contract based systems. Miners could be attracted, because they would get "charity-coins", which then could be donated to any participating charity. These participating charities can also act as coin-exchange to allow donors easy access to charity-coins and offering the receiving communities a way to receive fiat currencies to finance community initiatives. Because those charities have no common interests except the usage of this blockchain platform, it can still be regarded as a decentralised system.

4.2 Campaign administration

There is currently no dashboard to create and manage campaigns. Such a dashboard should be built to allow for a user friendly creation of new campaigns, defining impact goals and publishing gatherer actions.

4.3 Smart contract implementation

As time was limited and as we had to get used to a new programming language, we were not able to implement every single feature we wanted to. But the ones that actually were implemented are also working. During the presentation we were able to go through the whole process from starting a campaign and donating money to voting on community projects live on a test net without any issues.

5 Conclusion

Considering that nobody worked with smart contracts before, we were able to implement a lot of features during the limited time of the hackathon. We are especially happy that all implemented features work as intended and that we were able to show a fully working prototype at the end of the hackathon, which incorporated a smart contract running on the test network, a functioning website and a working sensor that was able to communicate with the blockchain. We were not able to provide an implementation of all features described, such as the different proofing mechanisms. However, for these missing features we provided a reference implementation, such as the naive proofing. Even though no actual proofing mechanism is implemented, smart contract developers can get an idea of how these functions are supposed to be used. The function body can be easily adjusted to fit different needs.

By creating an almost self-reliant system with integrated sensors, we achieved most of our goals regarding the hardware. The system can accurately and consistently read data from the attached sensors. Entire self-reliance was not realized, as batteries and LoRaWan-modules were not available for the project. We are convinced that these features could easily be added to the existing prototype. The wide range of available sensors and the modularity of the Raspberry Pi make our concept future-proof. We fully believe that similar self-built sensors could be deployed in large scale projects and function reliably.

Overall, we were very satisfied with our results and are looking forward to developing the solution further.