Advanced Distributed Systems & Blockchain

A charity DAPP by Yanick, Benjamin and Etienne

Application Requirements

- Simplistic UI
- Straightforward logic
- (Feedback and summary)

User Requirements

- Fast and easy way to donate
  - Select organization
  - Select amount
  - Send → Metamask as industry standard
- Insight of amount a particular organization accumulated after transaction

Organization Requirements

- Be able to check balances and withdraw whenever they want.

UI Requirements

- Reusability
- Simplicity

UI Donator

TOGETHER WE CAN GET THINGS DONE.

Aerzte ohne Grenzen

Amount (ETH)

Send

THANK YOU FOR YOUR GENEROUS DONATION!

AERZTE OHNE GRENZEN

0.1012001 ETH | 54.073237432000006 CHF

MOZILLA FOUNDATION

0.4 ETH | 213.72800000000004 CHF

SWISS RED CROSS

0 ETH | 0 CHF

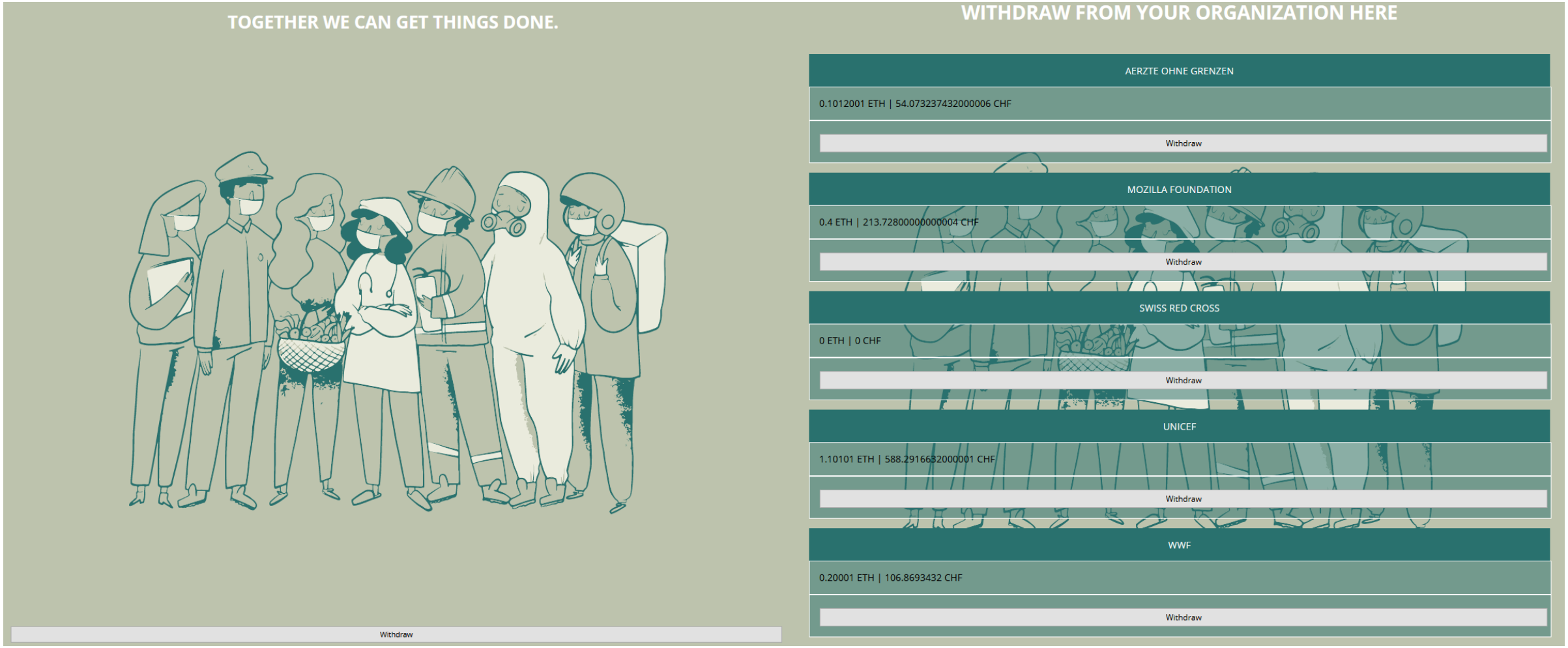
UNICEF

1.10101 ETH | 588.2916632000001 CHF

WWF

0.20001 ETH | 106.8693432 CHF

UI Organization



Architecture Requirements

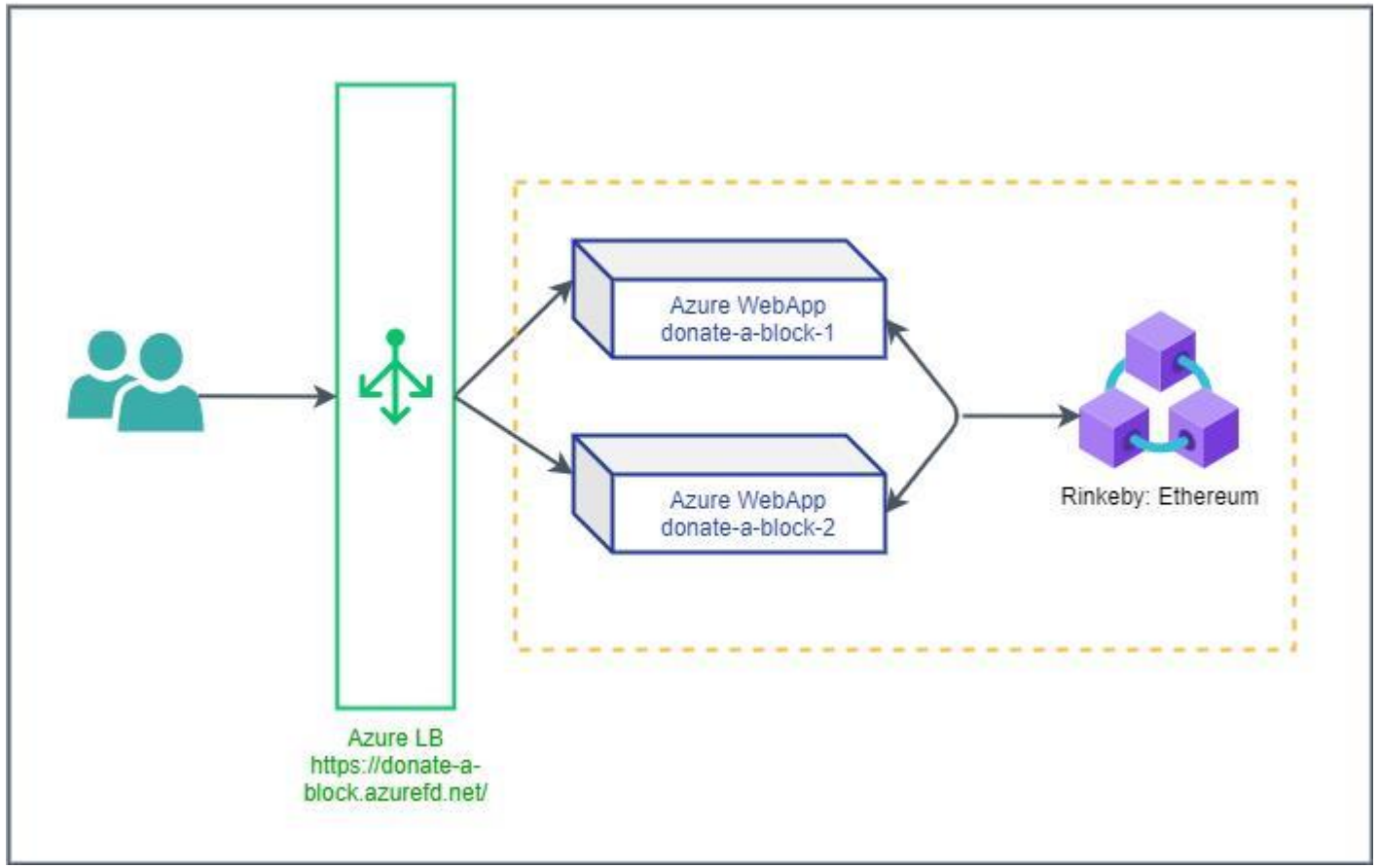
- Frontend
  - Lightweight
  - Easy to run/maintain
  - Preferably working with templates
- Backend
  - Loadbalancing
  - Rinkeby Testnet

Architecture General

- Handlebars
- Node
- Express
- Solidity

arvindkalra/express-box template

Architecture Loadbalancer



Additional Functionality

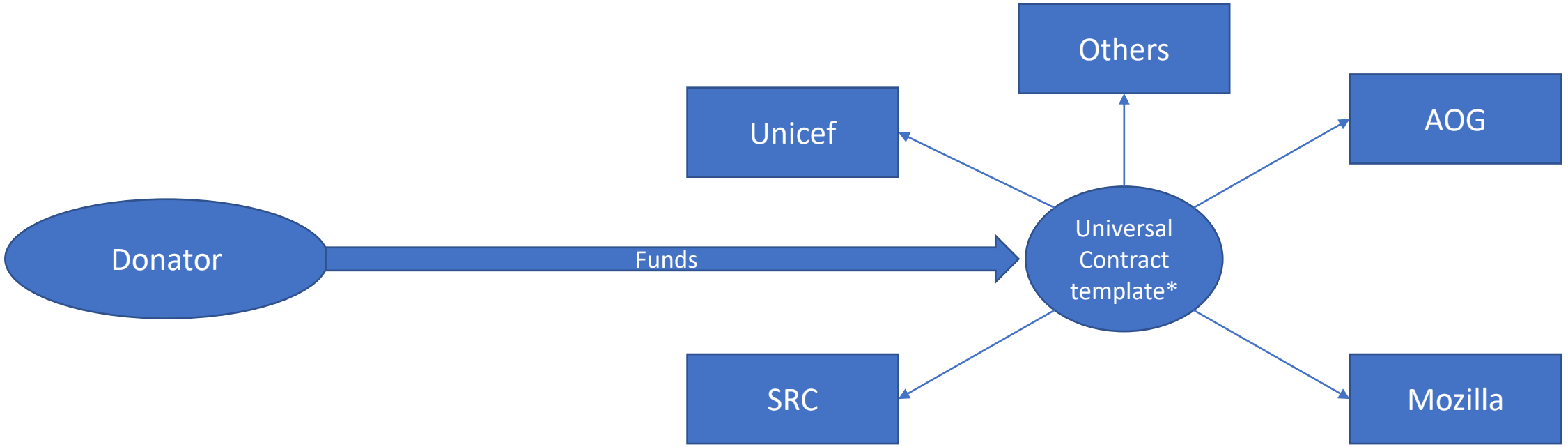
- CoinGecko API GET Request for up-to-date exchange rates
  - ETH → CHF

Smart Contract Requirements

- Small & simple
- Reusable

Smart Contract

- Each organization has their own contract to hold the donated funds
- Only owner of the organizations contract may withdraw funds



* Every organization has the *same* contract and thus their own access to funds without any accessability by others

Contract – Template

```
contract Main {

    struct Donation{
        address donor;
        uint amountDonated;
    }

    address payable public recipient;
    string public description;

    //map amount of money donated by donors address
    mapping(address => uint) public donorsAmount;

    uint public donors;
    uint public amountTotal;

    Donation [] public donations;

    constructor(
        string memory _description, address payable _recipient
    ) public {
        recipient = _recipient;
        description = _description;
    }

    modifier restricted()
        require(
            msg.sender == recipient, "No permission"
        );_;
    }
```

Considerations:

- Recipient is a payable address aka. an organization
- Amount donated is mapped to sending address
- Minimal information (description, address) linked to a particular organization
- Restriction for withdrawl → only organization can withdraw funds

Contract – Template

```
function donate() public payable {
    donorsAmount[msg.sender] = msg.value;
    amountTotal += msg.value;
    donors++;

    _createDonation(msg.sender, msg.value);
}

function _createDonation(address donor, uint amount)
private {
    Donation memory newDonation = Donation({
        donor: donor,
        amountDonated: amount
    });
    donations.push(newDonation);
}
```

Considerations:

- Donate:
  - Updates donation amount of sender & total amount of donators
  - Updates the total amount for particular organization
  - Creates a rekord of the donation
- Donation Rekord:
  - Generates new Donation object on the basis of the Donation struct
  - Adds this new Donation with sender and amount to the rekord Object, which holds all the transactions

Contract – Template

```
function withdraw() public restricted{
    uint balance = getBalance();
    require(
        balance >= 0, "Balance is too small"
    );
    recipient.transfer(address(this).balance);
}

function getBalance() public view
returns(uint){
    return address(this).balance;
}
```

Considerations:

- Withdraw:
  - Get Balance and check if amount is greater than zero
  - Transfer amount to particular organizations address

- Get Balance:
  - Get balance of the the associated address of an organization

Encountered Problems

- Handlebar usage for differentiating organization and user/donator didn’t work as expected

  →Two seperate summary views. One with withdraw and one without.

Feedback

- Nice Testat requirements
- Learned alot about how «easy» it is to work on smart contracts
- Good topic, i.e. working with finance
- No restrictions on HOW things should be done
- Good examples in lecture