

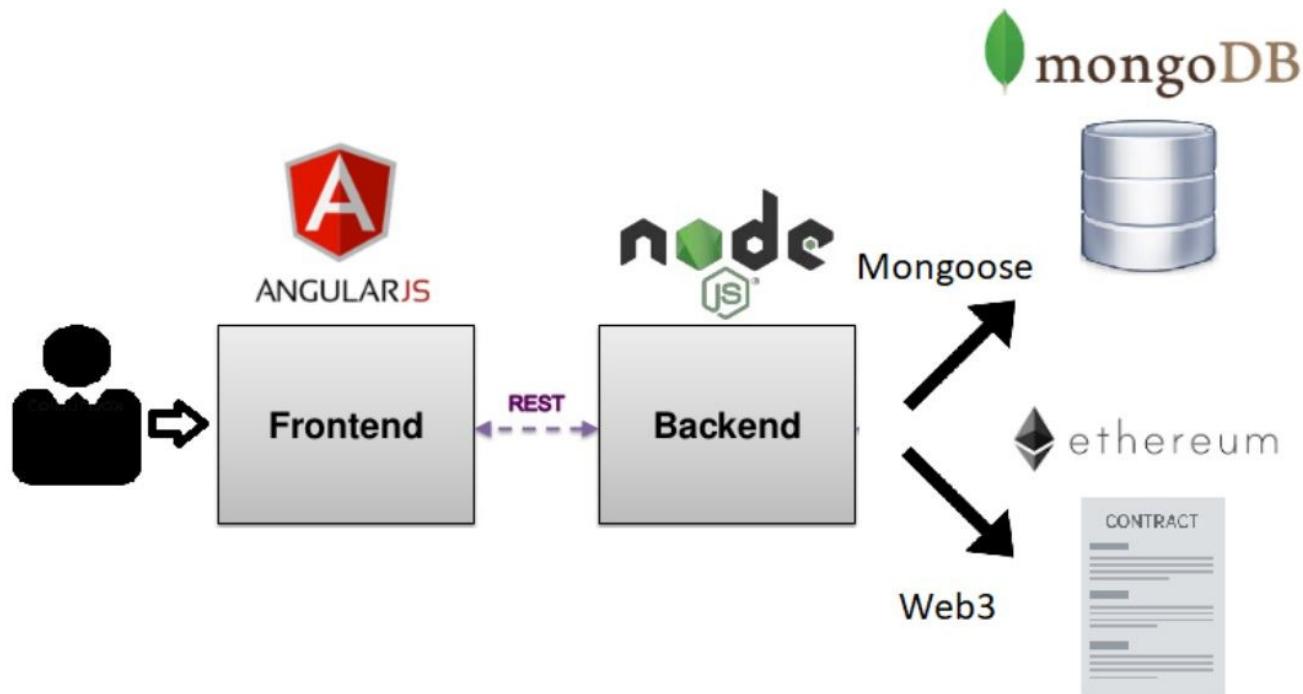
Objectives

- Frontend
 - Allows users (blockchain address) to create new projects
 - Allows project owners to:
 - List their projects
 - Cancel project
 - Withdraw funds from projects
 - Allows backers to:
 - Invest in a project
 - Check in which projects they invested in
 - Claim shares from those projects

Objectives

- Backend + Smart Contract
 - Allows project owners to:
 - withdraw funds when a project has been funded successfully
 - Allows backers to
 - retrieve a share (token) for each project they invest in
- A project has a deadline until which the funding goal must be reached

Concept and design



Frontend - Angular Router

- Maps a route to the correspondent template and the controller that controls it

```
// creates the App and adds the routes handler
angular.module('Blockstarter', [])

.config(function($routeProvider) {

    $routeProvider
        .when('/projects/view/:project', {
            templateUrl: 'templates/projectview.html',
            controller: 'ProjectCtrl'
        })

        ...
        .otherwise({
            redirectTo: '/projects/view'
        });
});
```

The screenshot shows a web application interface with a blue header bar. The header contains navigation links: 'ALL PROJECTS' (with a magnifying glass icon), '+ ADD PROJECT', 'CREATED' (with a calendar icon), 'BACKED' (with a dollar sign icon), and 'LOGOUT'. Below the header is a search bar with the placeholder 'Search...' and a 'Show All' button. The main content area is titled 'Project List'. It displays two projects: 'Chocolate' and 'New Project'. Each project card includes a hex ID, an end date, a goal reached status, a description, and a progress bar.

Project	ID	Ends	Goal Reached	Description	Progress (%)
Chocolate	0xf08ba1e78ebfb72d07682fc60384366e6cfa456e	13 Jul 2017 at 22:43	false	I love chocco.	34.0%
New Project	0x33fb80cb64f79d379521b0f221f4651889ace9a4	13 Jul 2017 at 22:32	false	Another project, not chocolate	0%

Frontend - Angular Controllers

- A controller for each view
- The views are loaded by the Angular router

```
// Controller module, holds a controller for each view
angular.module("Blockstarter.controllers", [])

.controller('AppCtrl', function($scope, Api, ... ) {
  ...
})

...
controller('ProjectCtrl', function($scope, Api, ... ) {
  ...
});
```

Frontend - Api Connection

- Each controller gets the data from the server through this module
- Seamless communication
- Promises support

```
// creates Api module and sets all the methods to interact with the server
angular.module('Blockstarter.api', ['Blockstarter.config'])

.factory('Api', function($http, CONFIG, $window, $timeout, $q) {

    this.getAllProjects = () => {
        return $http
            .get(CONFIG.endpoint + '/projects')
            .then(response => { return response.data; })
            .catch(error => { return error; });
    };

    ...

    return this;
});
```

Backend - Technologies



Server



Persistence



ethereum

Smart Contracts dev
on testrpc client



Fast deployment and
compilation

Backend - Api

Method	Route	Params	Description
POST	/api/v1/projects	next slide	Create a project
POST	/api/v1/projects/fund	- project - backer - amount	Fund a project
GET	/api/v1/projects		Get all projects
GET	/api/v1/projects/creator/:creator		Get all projects created by a creator
GET	/api/v1/projects/backer/:backer		Get all projects funded by a backer
GET	/api/v1/projects/:project		Show project information
POST	/api/v1/projects/withdraw	- project - creator - amount	Withdraw funds from the project
POST	/api/v1/projects/claim-shares	- project - backer - token	Claim project shares
POST	/api/v1/projects/show/shares	- project - backer	Show the shares owned by a backer

Backend - Api

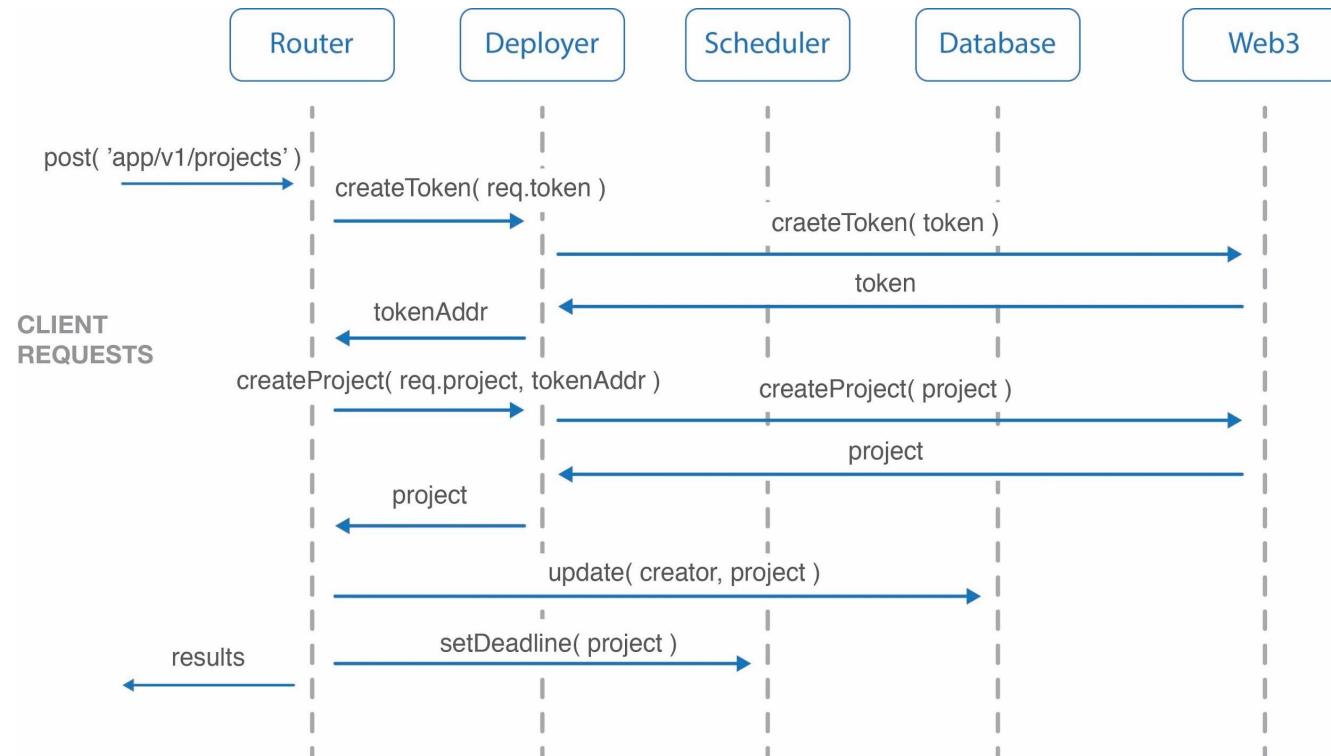
Method	Route	Params	Description
POST	/api/v1/projects	next slide	Create a project
POST	/api/v1/projects/fund	- project - backer - amount	Fund a project
GET	/api/v1/projects		Get all projects
GET	/api/v1/projects/creator/:creator		Get all projects created by a creator
GET	/api/v1/projects/backer/:backer		Get all projects funded by a backer
GET	/api/v1/projects/:project		Show project information
POST	/api/v1/projects/withdraw	- project - creator - amount	Withdraw funds from the project
POST	/api/v1/projects/claim-shares	- project - backer - token	Claim project shares
POST	/api/v1/projects/show/shares	- project - backer	Show the shares owned by a backer

Backend - Create Project (1)

Request Body:

```
{  
  token: {  
    initialSupply: 100, // number of shares  
    tokenName: "Example Token",  
    tokenSymbol: "Symbol",  
    creator: "0x ..." // creator address  
  },  
  project: {  
    title: "Example",  
    description: "First Project With Token",  
    goal: 100, // in ethers  
    duration: 120, // in minutes  
    sharesAvailable: 50, // number of tokens available (50 %)  
    creator: "0x ..." // creator address  
  }  
}
```

Backend - Create Project (2)



Backend – Extension

- Implementing a scheduler
 1. For each project created a timeout is set
 2. On timeout killContract() is called.
 3. If the campaign is closed and the goal is not reached, the project contract is killed.
 4. Otherwise, the backes can claim their tokens and the creator withdraw the funds.

```
this.setScheduler = (project, data) => {
  let time = project.deadline - (new Date().getTime());
  console.log(time);
  let timerObj = setTimeout(this.killContract, time, data);
  return timerObj;
}
```

Backend - Contracts

- *ShareToken.sol*
 - Create shares (tokens) that will be associated to a project
 - Specify initial supply and tokens available to backers
 - Map with share balances
 - At first the owner owns all shares and after the project is funded the owner transfers the shares to the backers on request

Backend - Contracts

- *Project.sol*
 - Receive Ether from anonymous Backers
 - Update the funding status of the project
 - Show funding status of the project
 - Kill the project and redistribute the funds (only Creator)
 - Withdraw funds (only Creator)
 - Retrieve a share (only Backer)
 - Using modifiers in order to restrict the access
 - Using events in order to keep track and give feedback to the backend

Backend - Contracts

```
mapping (address => uint256) balanceOf;

// Initializes contract with initial supply tokens to the creator of the contract
function ShareToken(uint256 _initialSupply, string _tokenName, uint8 _decimalUnits, string _tokenSymbol) {
    if (_initialSupply == 0) {
        _initialSupply = 100;
    }
    creator = msg.sender;
    initialSupply = _initialSupply;
    balanceOf[creator] = _initialSupply;                                // Give the creator all initial tokens
    name = _tokenName;                                                 // Set the name for display purposes
    symbol = _tokenSymbol;                                            // Set the symbol for display purposes
    decimals = _decimalUnits;                                         // Amount of decimals for display purposes
}

// Send coins
function transfer(address _to, uint256 _value) {
    balanceOf[creator] -= _value;                                       // Subtract from the creator
    balanceOf[_to] += _value;                                           // Add the same to the recipient
    Transfer(creator, _to, _value, initialSupply, name, symbol);
}
```

Backend - Contracts

```
// Support array used to iterate over the mapping
address[] iterator;
// Map of backers and amounts
mapping(address => Backer) fundings;

...
```

1

```
// Contract killer
function kill() goalNotMet {
    // Give back the money to all the backers
    for (uint i=0; i< iterator.length; i++) {
        address key = iterator[i];
        key.send(fundings[key].amount);
    }
    selfdestruct(msg.sender);
}
```

2

```
// Fallback function (send money)
function() payable campaignOpen {
    ...
    if (fundings[sender].exists) {
        fundings[sender].amount += amount;
    } else {
        fundings[sender] = Backer(amount, true, false);
        iterator.push(sender);
    }
    ...
}
```

3

Backend - Contracts

```
modifier onlyOwner() {
    if (msg.sender == owner) _;
}

modifier onlyAllowedBacker() {
    if (fundings[msg.sender].exists && !fundings[msg.sender].claimed) _;
}
```

```
function claimShares() onlyAllowedBacker campaignClosed returns(bool result) { ... }
```

```
// Events
event SomeoneBacked(address backer, uint amount, bool goalReached);
event ShareClaimed(address backer, uint tokens);
event WithdrawnFunds(bool successful, uint fundsLeft);
```

Design Decisions

Selecting event-based approach over array structure with promise.

- Waiting for the event completion ensure the success of funding.

Problems with integrating the Meteor framework with API.

- We selected AngularJS, because we had everything in ReactJS

MongoDB for storing the project and owner information

- Every project has only one owner. In this scenario MongoDB fits well.

Scheduler for implementing Extension

Circuit breaker

Backend – APIs

- Create project

1. Deploy token contract and assign shares
2. Deploy project contract including token contract address
3. Update Creator Mongodb with the newly created project
4. Set scheduler for deadline and return result

- Fund project

1. Call fund project function in the contract
2. Update Backer mongodb with the newly backed project
3. Return result