Junaid

junaidev@hotmail.com

Smart contracts Event Radar

Ethereum ENS Lookup



Contents

[Introduction 2](#_Toc515022489)

[Tools\APIs\Technology Stack 2](#_Toc515022490)

[Technical Architecture 2](#_Toc515022491)

[1) Restful Services endpoint: 2](#_Toc515022492)

[2) Front-end: 3](#_Toc515022493)

[Testing 4](#_Toc515022494)

[How to run 5](#_Toc515022495)

Smart Contracts Event Radar

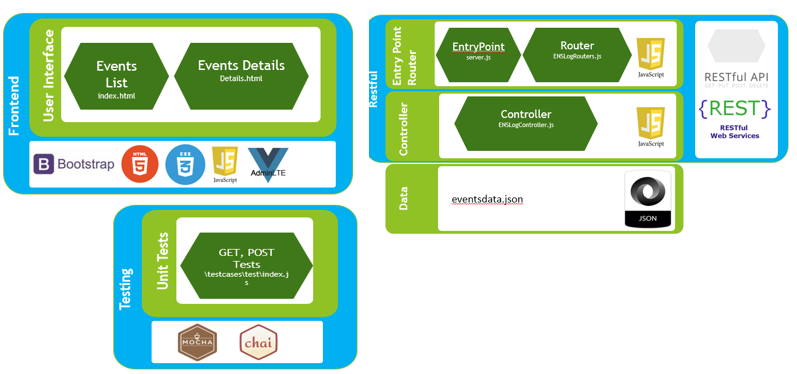
# Introduction

This web app could be used to monitor events of specified contract. In version 1.0.0 the smart contract address and ABI is hardcoded in application, but in future version there will be option to add any contract dynamically. Currently app monitors events of ENS Ethereum Name Service smart contract located at address '0x6090A6e47849629b7245Dfa1Ca21D94cd15878Ef'.

The app is divided into three different modules:

* Restful services endpoint,
* Client code and
* Testing code.

# Tools\APIs\Technology Stack

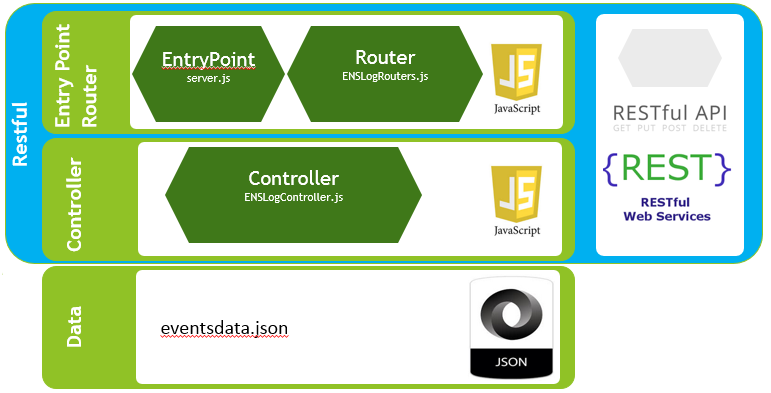
Following APIs\Libs are used for application.

* Node.js based
* Java script
* Express
* JSON
* Mocha
* Chai
* Web3
* HTML \ CSS \ Bootstrap
* Visual Studio Code
* npm package manager

# Technical Architecture

### Restful Services endpoint:

The purpose of backend module is to receive events in json from client code and save in underlying data file. When required data by client code this will send back data in json format. The events occurring in smart contract are saved in json file.

This module is developed using node.js and express based restful services. The service entry point is located in server.js file. The endpoint routs are mapped by using file located in side \api\routes\ENSLogRoutes.js file and controllers are written in \api\controllers\ENSLogController.js. As this is initial stage of project so data is saved in eventsdata.json data file. 

### Front-end:

The front end is based on bootstrap and Admin LTE. For now in version 1.0.0 the event lookup calls are sent to Ethereum node directly by front end. The calls are sent for looking events of ENS smart contract. When an event occurs front end shows notification and sends back event in json format to restful end point for persistence in server side Jason data file. Along with sending data to server the data table of events is also refreshed by querying data from restful end point. There are two files in front end one is index.html for listing all events that are recorded. The second file is details.html which renders details of selected event. In version 1.0.0 the data of event is not formatted like values for gas and timestamp inside event is not formatted.



# Testing

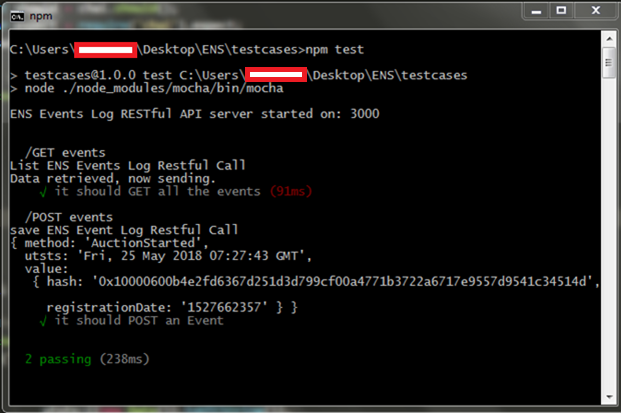
For demonstration of unit testing the unit tests are written by using chai and mocha java script libraries. These libs are used for TDD and automated unit testing.



There are two tests written in testcases\test\index.js. One testcase is for POST call and saving event JSON. Second test case is for GET call, for getting list of already persisted events. The test cases could be invoked by using following command:

|  |
| --- |
| npm test |

Before test cases invocation make sure that required libraries are installed by using npm and server is not running at oprt 3000. The test cases code will run server for itself.

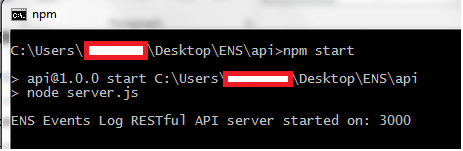


# How to run

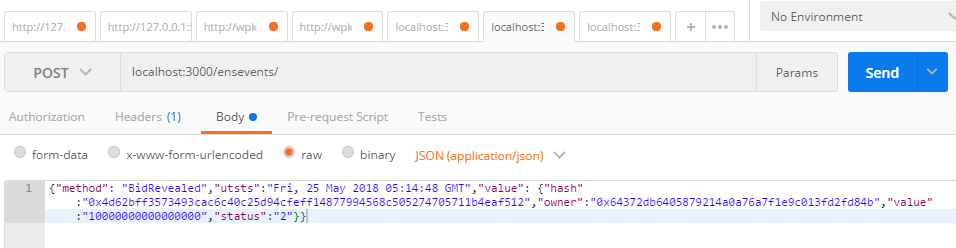
For running application first make sure that all libs are installed by using npm. After that First browse to api folder and issue following command:

|  |
| --- |
| api\ npm start |

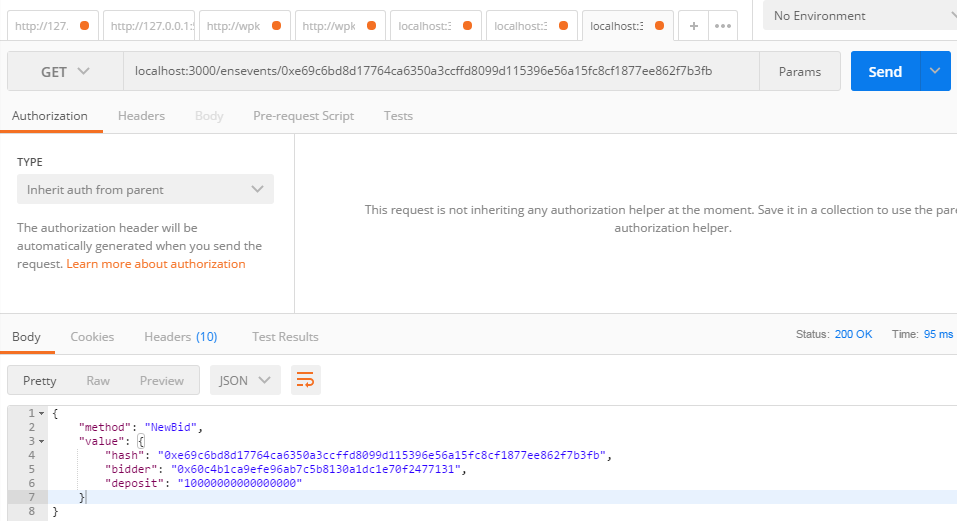
If everything is properly setup following screen short will be shown:



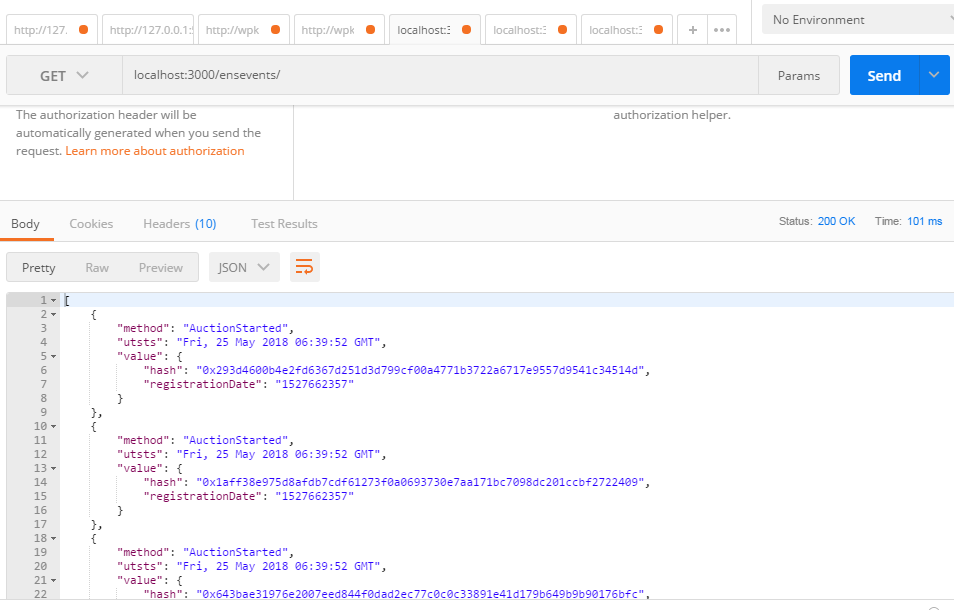
This indicates that server is running at port 3000. The Restful calls could also be verified by using post man. Following is post call demonstration for saving an event in server json.



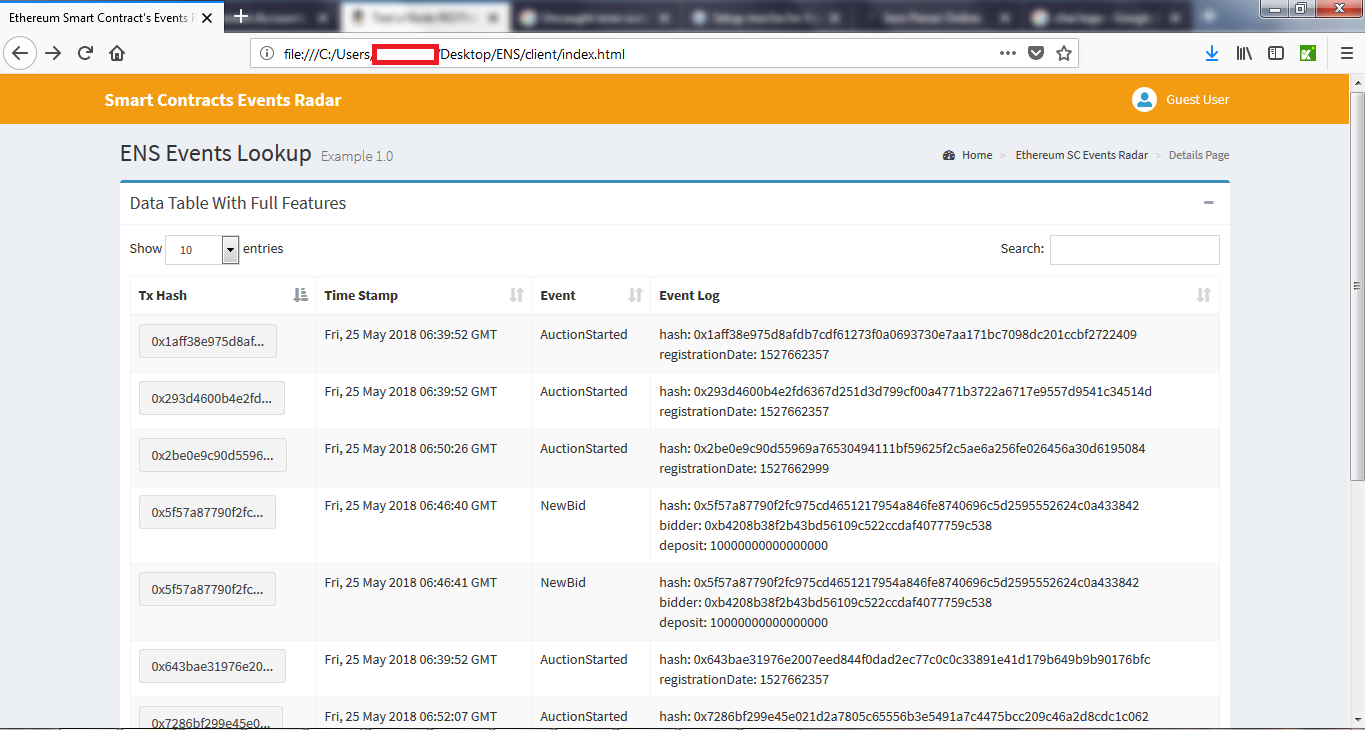
Following is restful get call demonstration for getting specified event from server json.



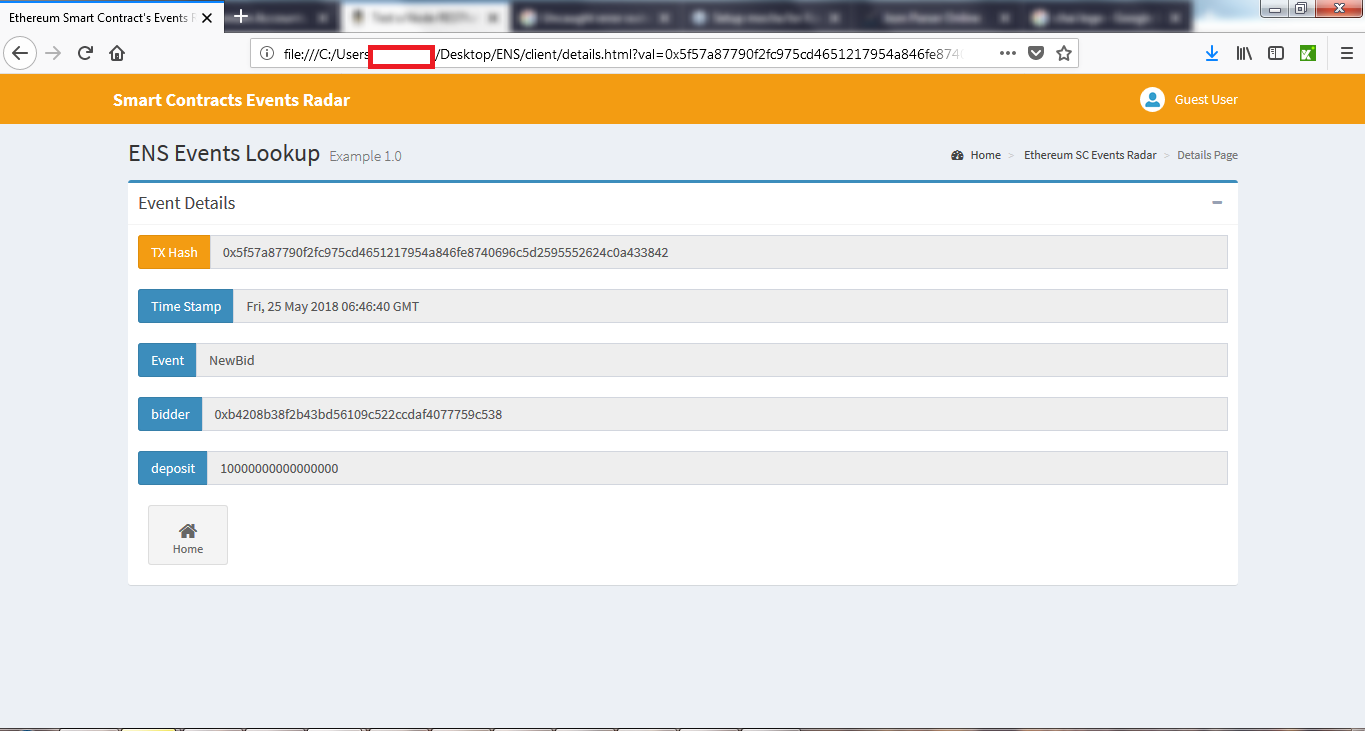
Following is restful get call demonstration for getting list of all events from server json.



Once server is started and restful services are up and running after that client could be launched in any web browser. It could be hosted on any webserver or could be launched directly in web browser. The client uses web3.js for lookup events, if any event occurs it sends data to server endpoint and also re renders its data table.



The details screen:



Possible Improvements

* Creating a standalone library for events lookup and moving that code to server side so no event is missed.
* Host client and server code on same root path so there is no need of CORS ( Cross Origin Resource Sharing ) permission in server.js
* Instead of using val in query sting param use data inside post