

**M03 Team Project**

**Blockchain Based Order and Access Management**

**SS21/22**

submitted by

**Sachith Harshitha Liyanagama**

Matr. Nr. 769617

**Ken Freise**

Matr. Nr. 746041

**Shakil Ahammed**

Matr. Nr. 769709

submitted to

**Prof. Dr.-Ing. Markus Haid**

**Table of Contents**

[1. Project Scope 3](#_Toc76636261)

[1.1. Requirements 3](#_Toc76636262)

[1.2. Deliverables 3](#_Toc76636263)

[2. Concept Overview 5](#_Toc76636264)

[2.1. General Concept Overview 5](#_Toc76636265)

[2.2. Detailed Concept Overview 6](#_Toc76636266)

[2.3. Very Detailed Concept Overview 7](#_Toc76636267)

[3. Blockchain 0](#_Toc76636268)

[3.1. What is Blockchain? 0](#_Toc76636269)

[3.2. What is Ethereum? 0](#_Toc76636270)

[3.3. Smart Contract 0](#_Toc76636271)

[3.3.1. What is it? 0](#_Toc76636272)

[3.3.2. How does it work 0](#_Toc76636273)

[3.3.3. Types of Smart Contracts 0](#_Toc76636274)

[3.4. Blockchain Oracle 0](#_Toc76636275)

[3.4.1. The Oracle Problem 0](#_Toc76636276)

[3.4.2. How does it work? 0](#_Toc76636277)

[3.4.3. Pros & Cons 0](#_Toc76636278)

[3.4.4. Oracles used in Ethereum 0](#_Toc76636279)

[3.5. How to Communicate with blockchain 0](#_Toc76636280)

[4. Components 0](#_Toc76636281)

[4.1. Ganache CLI 0](#_Toc76636282)

[4.2. Smart Contracts 1](#_Toc76636283)

[4.2.1. Main Contract 1](#_Toc76636284)

[4.2.2. Assets Contract 2](#_Toc76636285)

[4.2.3. History Contract 2](#_Toc76636286)

[4.2.4. OpenRequests Contract 2](#_Toc76636287)

[4.2.5. Oracle Service Contract 2](#_Toc76636288)

[4.3. Authentication Code Generator API 3](#_Toc76636289)

[4.3.1. Project Structure 3](#_Toc76636290)

[4.4. Internal Communication Rest API 4](#_Toc76636291)

[4.5. Customer Portal Application 5](#_Toc76636292)

[4.5.1. Project Structure 5](#_Toc76636293)

[4.6. INTRANAV RTLS Rest API 5](#_Toc76636294)

[5. How to Run 6](#_Toc76636295)

[5.1. Pre-Requisites 6](#_Toc76636296)

[5.1.1. Docker 6](#_Toc76636297)

[5.1.2. Ethereum Remix 6](#_Toc76636298)

[5.1.3. NGROK 6](#_Toc76636299)

[5.1.4. Visual Studio Code (DEV only) 6](#_Toc76636300)

[5.1.5. MAVEN (DEV only) 7](#_Toc76636301)

[5.1.6. GIT (DEV only) 7](#_Toc76636302)

[5.1.7. Web3j (DEV only) 7](#_Toc76636303)

[5.1.8. Nethereum Generator (DEV only) 7](#_Toc76636304)

[5.2. Clone the GIT project repository 7](#_Toc76636305)

[5.3. Start Ganache CLI Docker 8](#_Toc76636306)

[5.4. Deploy Smart Contracts 9](#_Toc76636307)

[5.5. Start the Internal Communication Rest API 11](#_Toc76636308)

[5.6. Start the Company Rest API 13](#_Toc76636309)

[5.7. Run the Client Application Web Service 14](#_Toc76636310)

[6. Amazon AWS Integration 15](#_Toc76636311)

[7. Source 16](#_Toc76636312)

[8. Bibliography 16](#_Toc76636313)

# Project Scope

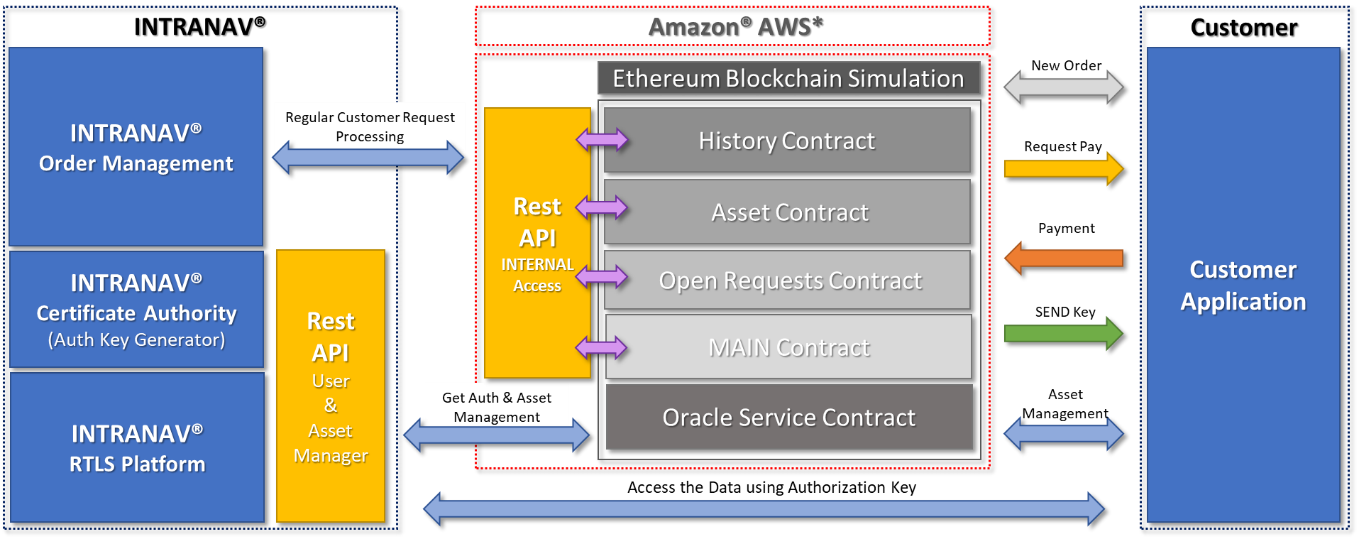


Figure : Project deliverable structure

Lorem ipsum at nusquam appellantur his, labitur bonorum pri no [5]. His no decore nemore graecis. In eos meis nominavi, liber soluta vim cu. Sea commune suavitate interpretaris eu, vix eu libris efficiantur.

## Requirements

Suspendisse in odio. In elit diam, cursus vitae, venenatis in, molestie in, leo. Cras ornare. Nulla libero.Phasellus feugiat mattis libero. Sed vehicula aliquam ligula. Nullam lacinia, felis vel dignissim sodales,enim lectus lobortis diam, quis nonummy mauris odio auctor tortor. Integer in dui nec lacus bibendumultrices. Etiam odio elit, aliquam et, porttitor id, interdum cursus, elit. Nulla eleifend tempor mauris.In vel arcu quis pede laoreet vulputate.Morbi pharetra magna a

## Deliverables

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Component | Description | Deliverability |
| 01 | Ganache **Blockchain Simulator** | Implementing and documentation of the dockerized version of the Ganache CLI. |  |
| 02 | Smart Contract(s)  **Main, History, Open Requests, Assets & Oracle Service** | Implementation of the Smart contracts with required structure of the artifact and documentation. |  |
| 05 | Rest API  **For Internal Communication** | [Swagger](https://swagger.io/) driven Java based Rest API (Optimized to work with the Main Smart Contract) |  |
| 06 | Rest API  **For External Communication** | [Swagger](https://swagger.io/) driven Java based Rest API (Optimized to work with the Main Smart Contract) |  |
| 07 | Rest API  **Mock API to represent the INRANAV REST API** | A simple [Swagger](https://swagger.io/) driven Java based Rest API with a built-in AES Token simulator and asset manager. |  |
| 08 | Amazon AWS Deployment | Deploy the above containerized applications in Amazon AWS separately and create the communication in between them. |  |
| 09 | Sample Customer Application | Simple Graphical User Interface Application to showcase the functionality. |  |
| All the components which are meant to host on AWS are created in docker containers. If required, using a PAID AWS account these can be hosted in AWS. | | | |

# Concept Overview

# General Concept Overview

The detailed Flowchart can be found as “.png” in “HDa\_TeamProject\_SS21\00\_Documentation\FlowChart.png”

# Blockchain

## What is Blockchain?

## What is Ethereum?

## Smart Contract

### What is it?

### How does it work

### Types of Smart Contracts

## Blockchain Oracle

### The Oracle Problem

### How does it work?

### Pros & Cons

### Oracles used in Ethereum

## How to Communicate with blockchain

# Components

## Ganache CLI

Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.

The Ganache CLI is the Command line version of Ganache. We will be using a modified version of the Ganache CLI docker with a built in Ethereum Bridge in order to be able to use external oracle services.

## Smart Contracts

We have four Smart Contracts: Main.sol, Assets.sol, OpenRequests.sol and OraService.sol. They have different functionalities. The overview call structure is shown below.

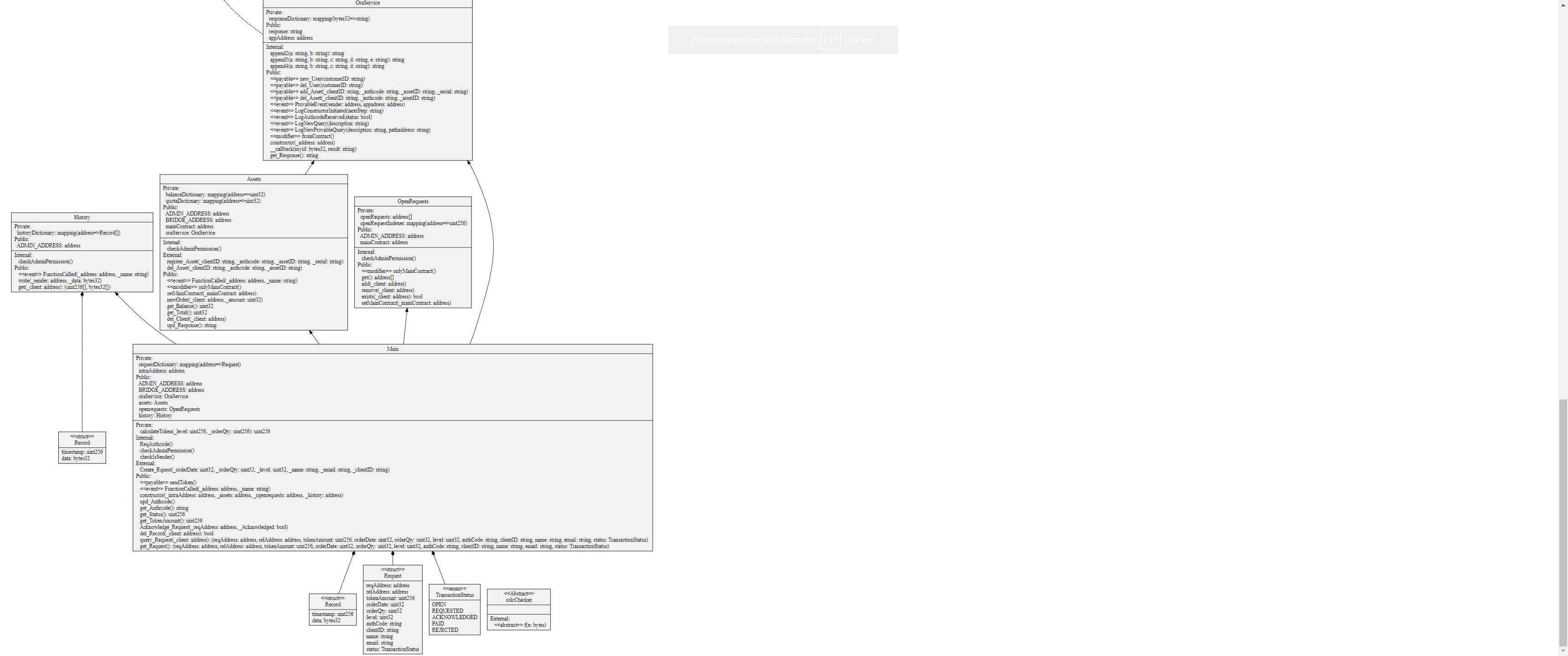


Figure : UML Diagram of the Smart Contract Communication

### Main Contract

Functionality of Main Contract:

* Save the Request a customer created to get the authentication code
* Calculate the price to process request
* Pay the required tokens for the request
* Invoke function of the OraService smart contract to generate and get new authentication code
* Update the new Authcode on Blockchain
* Return the new Authcode to Web Application
* increase Sensor balance if payment was successful
* decrease Sensor balance if Sensor is added

### Assets Contract

Functionality of Asset Contract:

* Adds or deletes an asset
* Update the asset quota and balance with the order amount

### History Contract

* Records history of the order process

### OpenRequests Contract

* Provides a list where open requests can be saved to or deleted from

### Oracle Service Contract

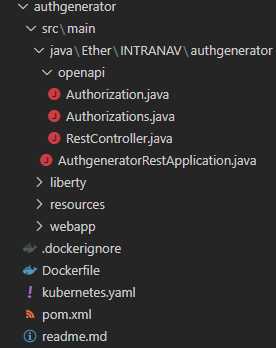
Functionality of Oracle Service Contrace:

* Use Provable Oracle Service to get the Authcode from Rest API
* Save the Authcode on the Blockchain
* Return the new Authcode to Main.sol Contract

## Authentication Code Generator API

The Authentication Code Generator Rest API provides two endpoints. The first one: “/new/{authToken}/{reqID}". This endpoint gets an authToken and reqID as input and is returning the authentication code for the requester id. If the request id is not existing, a new authentication code is created and returned. The second endpoint is: “delete/{authToken}/{reqID}". There the authentication code of the request id is deleted.

## Project Structure



The most important file is RestController.java. This File contains the implementation of the 2 Endpoints the Rest API provides.

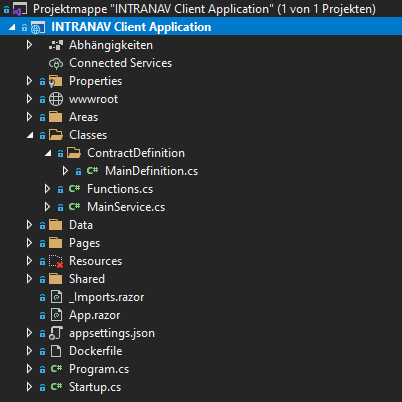
## Internal Communication Rest API

## Customer Portal Application

The Customer Application provides an Interface for the Customer to:

* create and send an Authentication Code request
* pay the required tokens for the request
* get the Authentication code and the Sensor Balance
* add and delete a Sensor.

### Project Structure



In “Functions.cs” the main functions like newRequest() and transfertoken() are implemented. The main functionality of the User Application can be modified here. These functions invoke nethereum functions which are implemented in “MainService.cs”. Netherum is the interface between the Web Application and Ethereum Blockchain/Smart Contracts. In MainDefinitions.cs the names of the SmartContract Functions are defined which nethereum is calling.

## INTRANAV RTLS Rest API

# How to Run

## Pre-Requisites

### Docker

In order to proceed with setup of the project it is required to configure docker on your computer.

Please refer: <https://docs.docker.com/get-docker/>

### Ethereum Remix

Given its ability to debug and test the smart contracts as well as extensive support with the built-in solidity compilers, Ethereum Remix is the suitable Solidity programming IDE for this project.

Please refer: <https://remix-project.org/>

### NGROK

In order to for the blockchain to access external data using the oracle service, the external data source must be exist in the internet. Therefore we will be using the NGROK local tunneling platform to expose the Company Rest API system to the internet so that we could obtain data from it.

Please refer: <https://dashboard.ngrok.com/get-started/setup>

### Visual Studio Code - WikipediaVisual Studio Code (DEV only)

Since the REST API microservices are programmed in JAVA language, Visual Studio Code can be the go to method for programming the APIs. Similarly with the plugins available for open liberty the API can be debugged live without the need of restart.

Please refer: <https://code.visualstudio.com/>

In order to proceed you will also require to install below plugins

1. Java Extension Pack
2. Open Liberty Tools

### JAVA

Java JDK or JDE is necessary to run .java files.

Please refer: <https://www.java.com/de/download/manual.jsp>

### MAVEN

Maven is a build automation tool used primarily for Java projects. Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. 

Please refer: <https://maven.apache.org/install.html>

### GIT

Since the source codes are all stored in GIT repositories it would be convenient to have git installed on the local computer.

Please refer: <https://git-scm.com/>

### Ethereum Transactions - Web3jWeb3j (DEV only)

Web3j is a lightweight, highly modular, reactive, type safe Java and Android library for working with Smart Contracts and integrating with Ethereum blockchains. This allows you to work with Ethereum blockchains, without the additional overhead of having to write your own integration code for the platform.

Please refer: <http://docs.web3j.io/latest/quickstart/>

### Nethereum Generator (DEV only)

Nethereum is the .Net integration library for Ethereum, simplifying smart contract management and interaction with Ethereum nodes whether they are public, like Geth , Parity or private, like Quorum and Besu.

Please refer: <http://docs.nethereum.com/en/latest/getting-started/>

In order to generate the API libraries, we will be using the Console CLI version of the Nethereum.

Please refer : <https://docs.nethereum.com/en/latest/nethereum-codegen-console/>

## Clone the GIT project repository

**Step 1:** Clone the project repo to your computer.

git **clone** **https**://github.com/SachiHarshitha/HDa\_TeamProject\_SS21.git

## Start Ganache CLI Docker

Step 1: Clone the Original Ganache CLI docker source into the path “.\01\_GanacheCLI”.

Should look like this : “.\01\_GanacheCLI\ganach-cli”

git **clone** **https**://github.com/trufflesuite/ganache-cli.git

**Step 2:** Clone the Ganache CLI, Ethereum Bridge integration project into the same path.

Should look like this : “.\01\_GanacheCLI\GanacheCLI-with-EthBridge”

git **clone** **https**://github.com/SachiHarshitha/GanacheCLI-with-EthBridge.git

**Step 3:** Copy “1entrypoint.sh”, “args.js” and “Dockerfile” from “.\01\_GanacheCLI\GanacheCLI-with-EthBridge\” folder and replace into “.\01\_GanacheCLI\ganache-cli\” .

Info: In args.js are wallet addresses defined. The first wallet address is (0xb8B7…). In the smart contracts we use this address as Admin\_Adress. You need this information later when you deploy the smart contracts. **Some of the next steps are only necessary, if you change the args.js file.**

**Step 4**: Build the docker container.

**cd** .\ganache-**cli**\  
docker build --tag trufflesuite/ganache-**cli**-**test** .

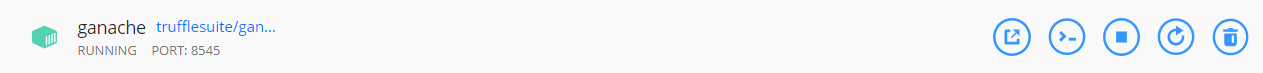
**Step 5:** Run the docker container.

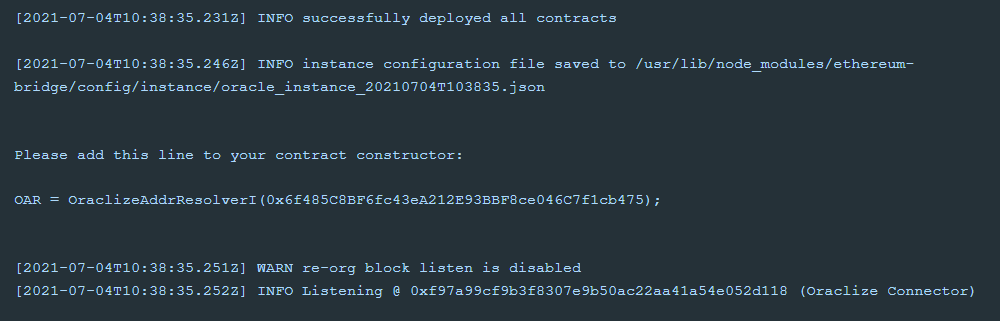
docker run -d --name ganache -p 8545:8545 trufflesuite/ganache-cli-test -l 80000000 -m "argue liberty sock desert drift peasant vivid fox hint document author circle" –v

\*\* If you have a specific mnemonic code for your wallet replace the string value above with your own mnemonic string.

!!! Only important when you use an other wallet address (different args.js used), if not ignore this step!!!

**Step 6:** Open the docker CLI for the ganache docker and copy out the Ethereum Bridge Address. You will need it to deploy the smart contracts.

How to : Open Docker Software, and double click on the running ganache container. 

Example : 

## Deploy Smart Contracts

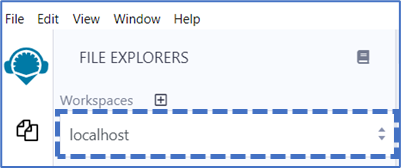
**Step 1:** Open Remix IDE and set workspace to “localhost”

Figure : Set Workspace

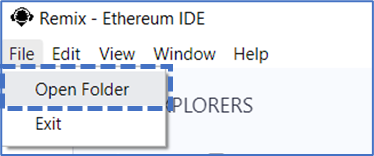


Figure : Open Folder view

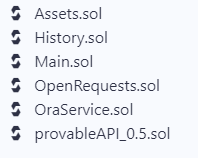
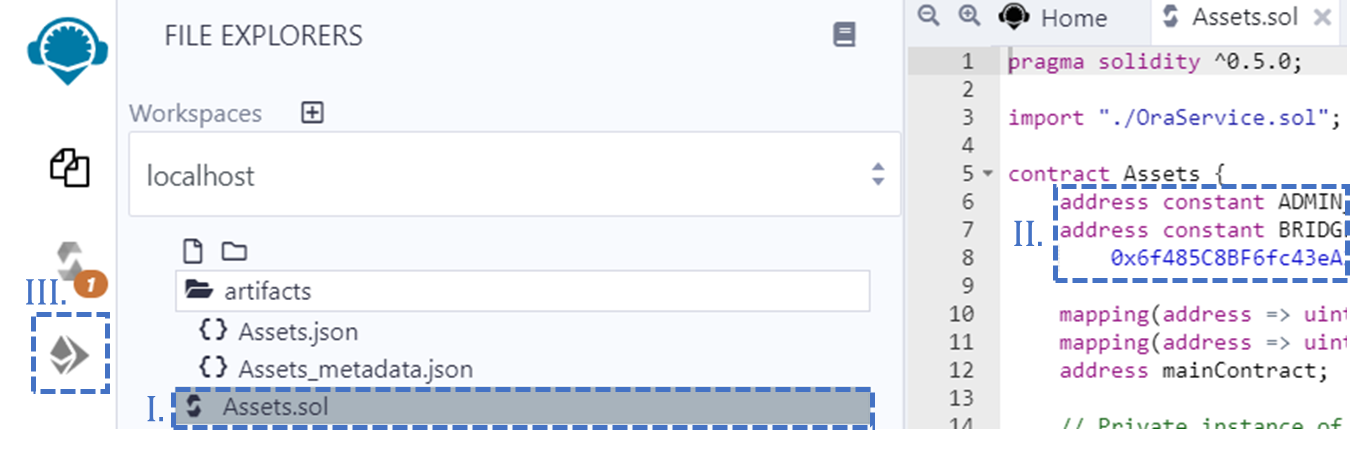
**Step 2:** Open the “\02\_SmartContract” folder from File Menu.

Figure : Smart Contracts

You will be able to see 5 \*.sol files inside the folder as shown.

**Step 3**: Open “Assets.sol” smart contract



!!! Only important when you use other wallet address (different args.js used), if not ignore this step!!!

**Step 4:**

Figure : Assets Smart Contract Deployment – Step 1

please update the below information according to your instance as shown in “Item II”.

**address** constant ADMIN\_ADDRESS = 0x79bD771Dc8CaBdaFf066aFfeB86455f9d5602E65;

//ADMIN Account which the Internal API uses

**address** constant **BRIDGE\_ADDRESS** =  
0x6f485C8BF6fc43eA212E93BBF8ce046C7f1cb475;

// Ethereum-Bridge Account address

**Step 5:**

Now go to the compile tab and compile the contract, then deploy the contract using the “Deploy & Run Transactions” tab in REMIX IDE.

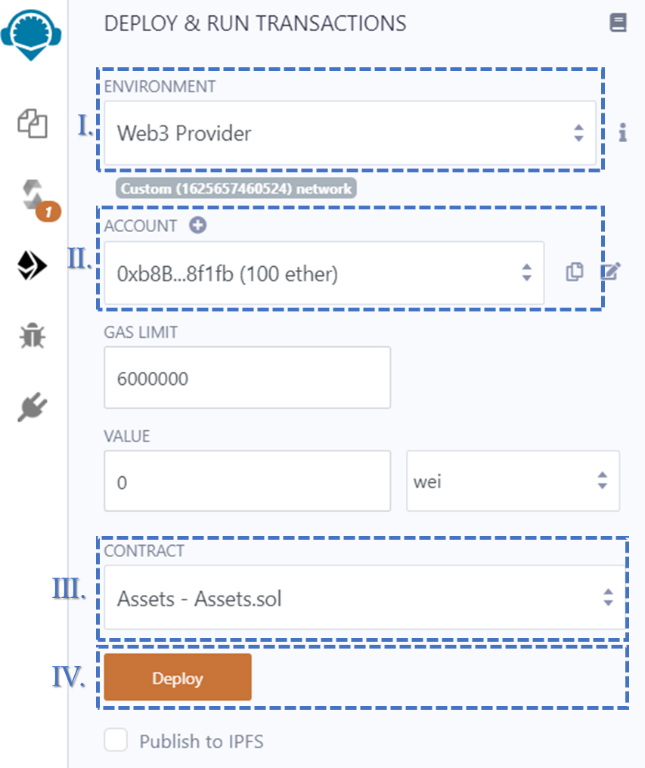
1. **Set Environment**

Figure : Deploy Contract

Select the “**Web3 Provider**” as the environment.

In the coming up “**External Node Connection** “window, click “**OK** “.

1. **Set Account**

Select the first account in the blockchain. (it’s the (0xbb87) account, which is defined as admin address in the smart contracts.

1. **Check the contract**

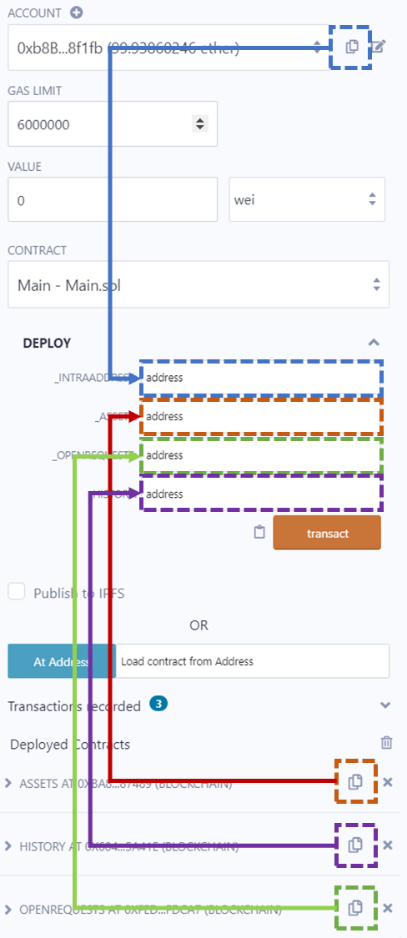
Make sure the “**Assets.sol**” contract is selected.

1. **Deploy the contract**

Click “Deploy” to deploy the contract into the blockchain.

**Step 4**: Repeat the above “Step 3” for “History.sol” and “OpenRequests.sol” smart contracts.

**Step 5:** Deploy the Main Contract.

Since the Main Contract is larger in size, please Enable “Contract Optimization” in Compile Tab.

Repeat the above “Step 3” according to the address values in your instance.

When you are required to deploy the contract, it will require certain parameters to be passed into the constructor as shown below. Copy the address values from below “Deployed Contracts” section and provide it into the appropriate address parameter.

Please note that use the First Account in the “Accounts” drop down list.

**AVOID** **using** **the** **second** account since **it** is fixed to **the** Ethereum Bridge.

**Step 6:** Set the Main Contract address in “Assets.sol” and “OpenRequests.sol” contract instances.

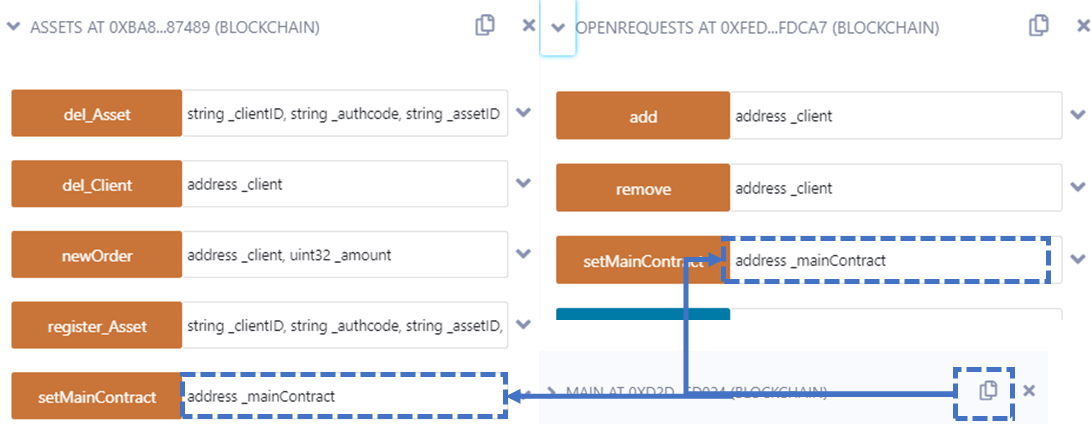


Figure : Set Main Contract address into Assets and OpenRequests.

## Start the Internal Communication Rest API

**Step 1:** Redirect to “\04\_InternalRestAPI” path and edit the contract addresses.

**THIS STEP SHOULD NOT BE NECCECCARY, BUT CHECK IF THE ADRESSES ARE CORRECT.**

The “Parameters.java” JAVA file can be found here under the path, “.\04\_InternalRestAPI\src\main\java\Ether\INTRANAV\rest\api\openapi”.

Open the file in your text editor and replace the below information.

/\*\*  
 \* Set the Parameters below according to the Simulation Environment  
 \*/  
**public** class Parameters {  
  
// Main Smart Contract Address  
**public** **static** **final** **String** MAIN\_CONTRACT\_ADDRESS = "0xd2DF0bc9B2e9eb28c260daDf7D2ACC6db23fD024";  
// ASSETS Smart Contract Address  
**public** **static** **final** **String** ASSETS\_CONTRACT\_ADDRESS = "0xbA89f50904FeB82245C9c238dDb8A88420b87489";  
// Open Requests Smart Contract Address  
**public** **static** **final** **String** OPREQUESTS\_CONTRACT\_ADDRESS = "0xfEdC7803a40c39ae4bfef1430677B5f3B6cFDca7";  
// History Smart Contract Address  
**public** **static** **final** **String** HISTORY\_CONTRACT\_ADDRESS = "0x604865472362ba89C115dB5a7e01A31e8E75a41E";  
// Private KEY of the Client Account used in the project  
**public** **static** **final** **String** CLIENT\_PRIVATE\_KEY = "0xbf562bcbb6792187dc5cdcf645ff07595d68ce9f234f173785ca1eae097c61e1";  
// Private KEY of the ADMIN Account used in the project  
**public** **static** **final** **String** ADMIN\_PRIVATE\_KEY = "0x77f4d8047405cc8dbeed5c831a6a8123c3102948b235b5aca7a94be529cd743e";  
// Address of the Ganache Docker. In order to access another docker value must be "http://host.docker.internal:PORT/"  
**public** **static** **final** **String** GANACHE = "http://host.docker.internal:8545/";  
}

**Step 2 :** Compile the maven project.

cd .\ 04\_InternalRestAPI \  
**mvn** install -f ".\pom.xml"

**Step 3:** Build the docker container

docker build -t rest-api:1.0-SNAPSHOT .

**Step 4:** Start the Docker Container

docker run -d --name InternalAPI -p 9090:9090 rest-api:1.0-SNAPSHOT

Now if you open the path <http://localhost:9090/openapi/ui/> you will be able to see the Internal Communication API is running.

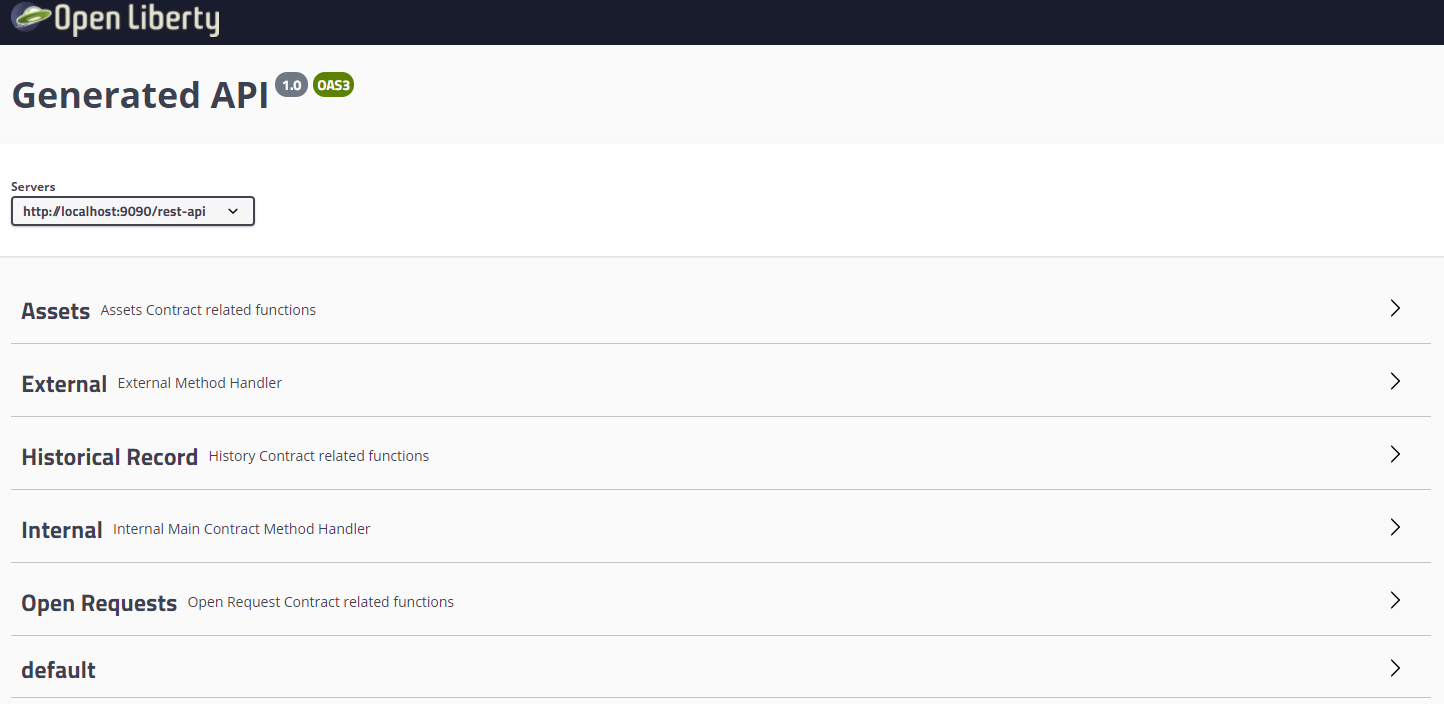


Figure : Internal Communication Rest API

## Start the Company Rest API

**Step 1:** Redirect to “\03\_CompanyRestAPI” path and compile the maven project.

cd .\03\_CompanyRestAPI\  
**mvn** install -f ".\pom.xml"

**Step 2:** Build the docker container

docker build -t company-mock-api:1.0-SNAPSHOT .

**Step 3:** Start the Docker Container

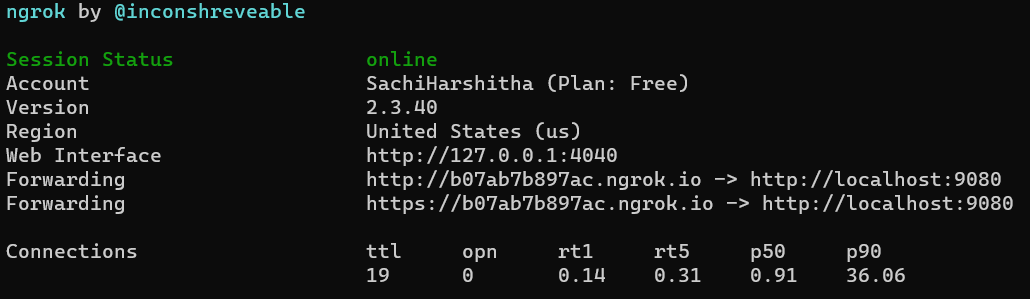
docker run -d --name system -p 9080:9080 company-mock-api:1.0-SNAPSHOT

**Step 4:** Expose the port 9080 and generate an external web address for the Rest API.

.\ngrok http 9080

**Step 5:** Check if above steps succeed by opening the link generated in the ngrok console window.

Example :



The address mentioned above MUST be append with “/openapi/ui”

Ex. <https://b07ab7b897ac.ngrok.io/openapi/ui/>

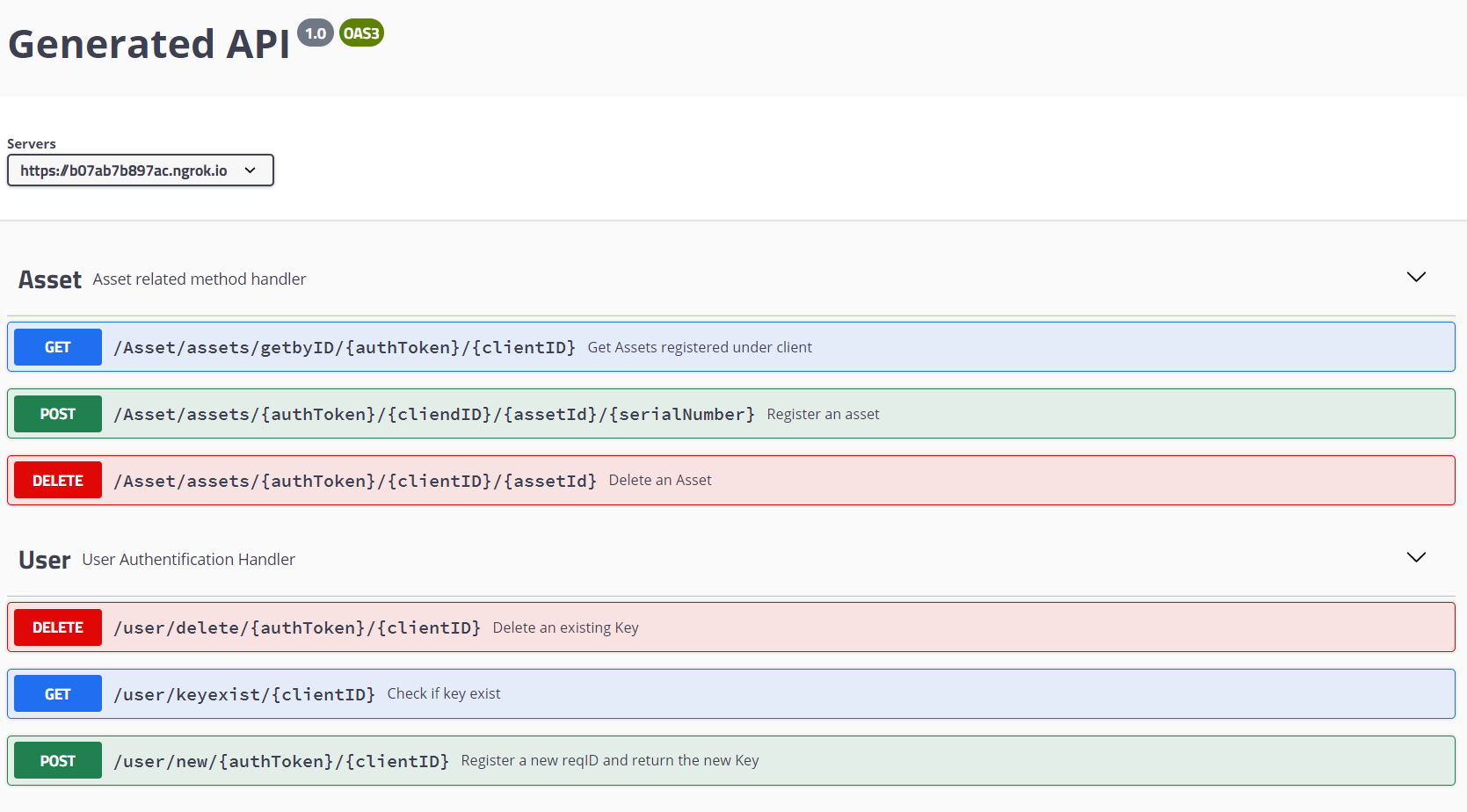


Figure : Sample COMPANY Rest API

## Run the Client Application Web Service

**Step 1:** Redirect to “\05\_ClientWebApp\20\_Release\” path and run the “INTRANAV Client Application.exe” file.

Example:

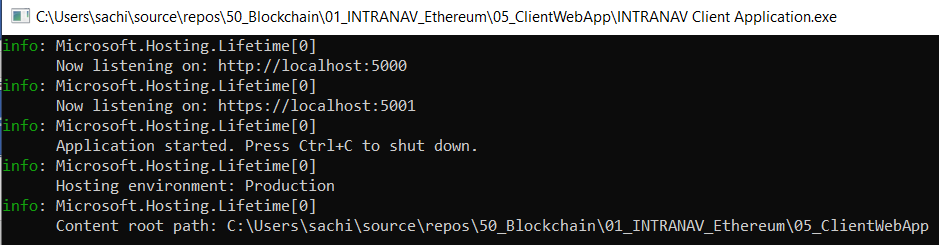


Figure : CMD window of Blazor server app

Now you if you open the address in your web browser <https://localhost:5001/> you will be proceeding to the web application as shown below.

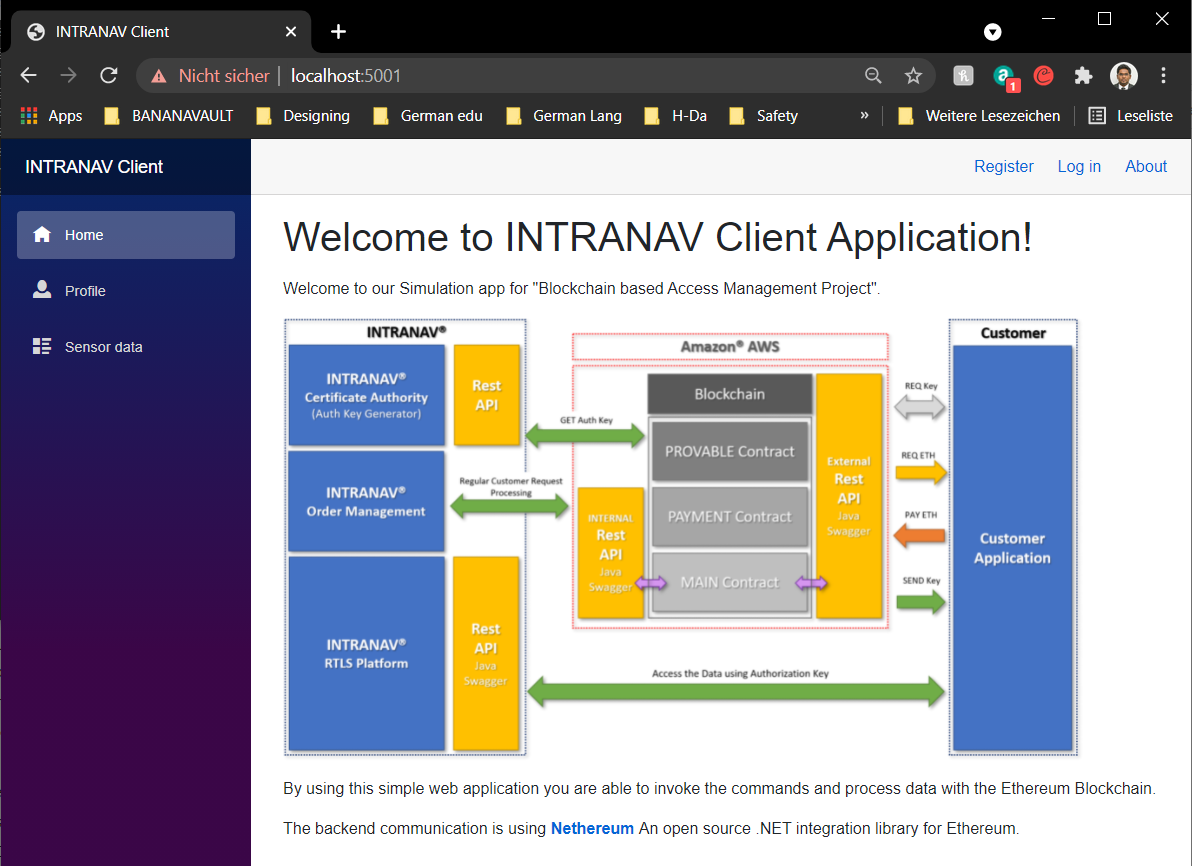
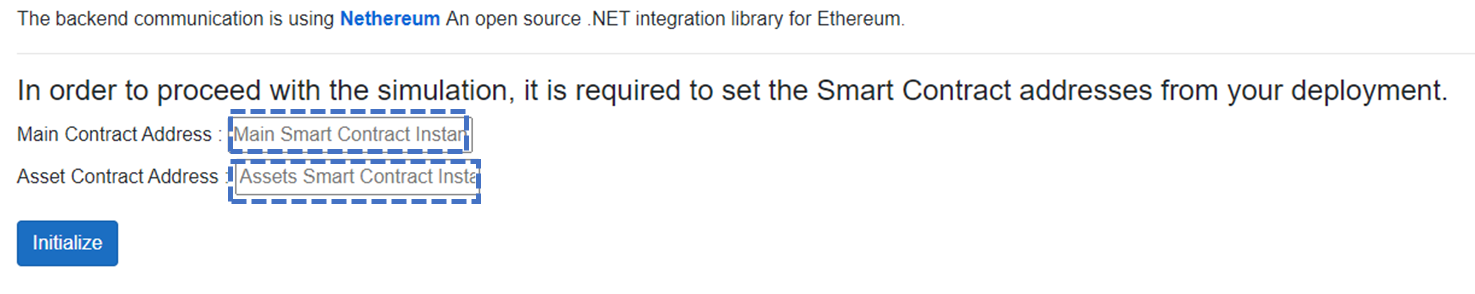


Figure : Client App User Interface

**Step 2:** Initialize the application



Copy and paste the Main and Assets smart contract instance address from above section “4.4 Deploy Smart Contracts”. Then click “Initialize”.

# Amazon AWS Integration

In order to be able to deploy the blockchain and the application into the Amazon AWS platform, we have created the applications with the ability to be containerized. Which then can be easily deployed on AWS using Kubernetes cluster or docker containers directly.

Amazon AWS integration sometimes would be costly. Therefore, it is important to categorize the application based on resource usage as well as available support in order to select the matching AWS system.

Given the size and resource usage we have categorized the components as below.

|  |  |
| --- | --- |
| Item | AWS system |
| Internal Communication API | Kubernetes Cluster |
| INTRANAV REST API Simulation | Fargate Cluster |
| Ganache CLI environment | Kubernetes Cluster |
| Client Application | Elastic Beanstalk |

# Source

Ganache CLI, Ethereum bridge modification: <https://github.com/SachiHarshitha/GanacheCLI-with-EthBridge>

# Bibliography

3