

**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](#_Toc78106005)

[1.1. Requirements 3](#_Toc78106006)

[1.2. Deliverables 3](#_Toc78106007)

[1.3. Timeline 4](#_Toc78106008)

[2. Concept Overview 5](#_Toc78106009)

[2.1. General Concept Overview 5](#_Toc78106010)

[3. Blockchain 6](#_Toc78106011)

[3.1. What is Blockchain? 6](#_Toc78106012)

[3.2. What is Ethereum? 6](#_Toc78106013)

[3.3. Smart Contract 6](#_Toc78106014)

[3.3.1. What is it? 6](#_Toc78106015)

[3.3.2. How does it work 6](#_Toc78106016)

[3.3.3. Types of Smart Contracts 7](#_Toc78106017)

[3.4. Blockchain Oracle 8](#_Toc78106018)

[3.4.1. The Oracle Problem 8](#_Toc78106019)

[3.4.2. How does it work? 8](#_Toc78106020)

[1. Inside Smart Contract call oracle function to get external data. 8](#_Toc78106021)

[3.4.3. Oracles used in Ethereum 9](#_Toc78106022)

[3.5. How to Communicate with blockchain 9](#_Toc78106023)

[4. Components 10](#_Toc78106024)

[4.1. Ganache CLI 10](#_Toc78106025)

[4.2. Smart Contracts 10](#_Toc78106026)

[4.2.1. Main Contract 11](#_Toc78106027)

[4.2.2. Assets Contract 11](#_Toc78106028)

[4.2.3. History Contract 11](#_Toc78106029)

[4.2.4. OpenRequests Contract 11](#_Toc78106030)

[4.2.5. Oracle Service Contract 11](#_Toc78106031)

[4.3. Authentication Code Generator API 12](#_Toc78106032)

[4.3.1. Project Structure 12](#_Toc78106033)

[4.4. Internal Communication Rest API 13](#_Toc78106034)

[4.4.1. Project structure 13](#_Toc78106035)

[4.5. Customer Portal Application 14](#_Toc78106036)

[4.5.1. Project Structure 14](#_Toc78106037)

[4.6. INTRANAV RTLS Rest API 15](#_Toc78106038)

[4.6.1. Project Structure 15](#_Toc78106039)

[5. How to Run 15](#_Toc78106040)

[5.1. Pre-Requisites 15](#_Toc78106041)

[5.1.1. Docker 15](#_Toc78106042)

[5.1.2. Ethereum Remix 16](#_Toc78106043)

[5.1.3. NGROK 16](#_Toc78106044)

[5.1.4. Visual Studio Code (DEV only) 17](#_Toc78106045)

[5.1.5. JAVA 17](#_Toc78106046)

[5.1.6. MAVEN 17](#_Toc78106047)

[5.1.7. GIT 17](#_Toc78106048)

[5.1.8. Web3j (DEV only) 17](#_Toc78106049)

[5.1.9. Nethereum Generator (DEV only) 18](#_Toc78106050)

[5.2. Clone the GIT project repository 18](#_Toc78106051)

[5.3. Start Ganache CLI Docker 18](#_Toc78106052)

[5.4. Deploy Smart Contracts 19](#_Toc78106053)

[5.5. Start the Internal Communication Rest API 22](#_Toc78106054)

[5.6. Start the Company Rest API 24](#_Toc78106055)

[5.7. Run the Client Application Web Service 25](#_Toc78106056)

[6. Amazon AWS Integration 27](#_Toc78106057)

[6.1. How to deploy 27](#_Toc78106058)

[7. Source 27](#_Toc78106059)

[8. Bibliography 28](#_Toc78106060)

[9. Appendix I - Figures 29](#_Toc78106061)

[10. Appendix II - Tables 29](#_Toc78106062)

# Project Scope

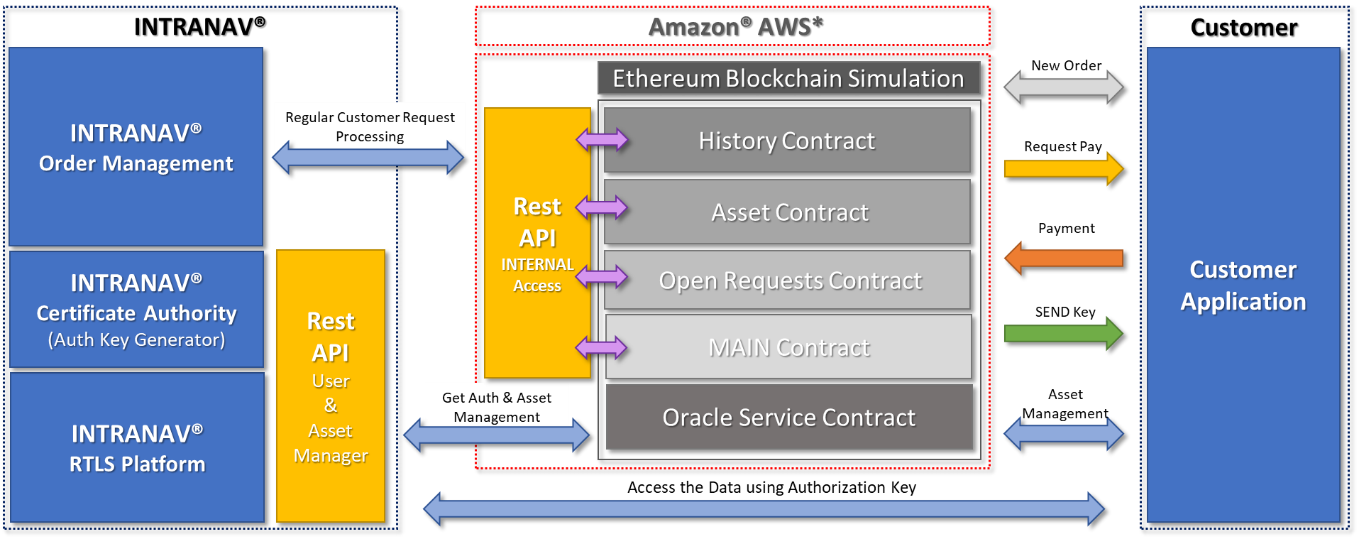


Figure : Project deliverable structure

The project scope was to develop client order management system integration concept using Ethereum Blockchain.

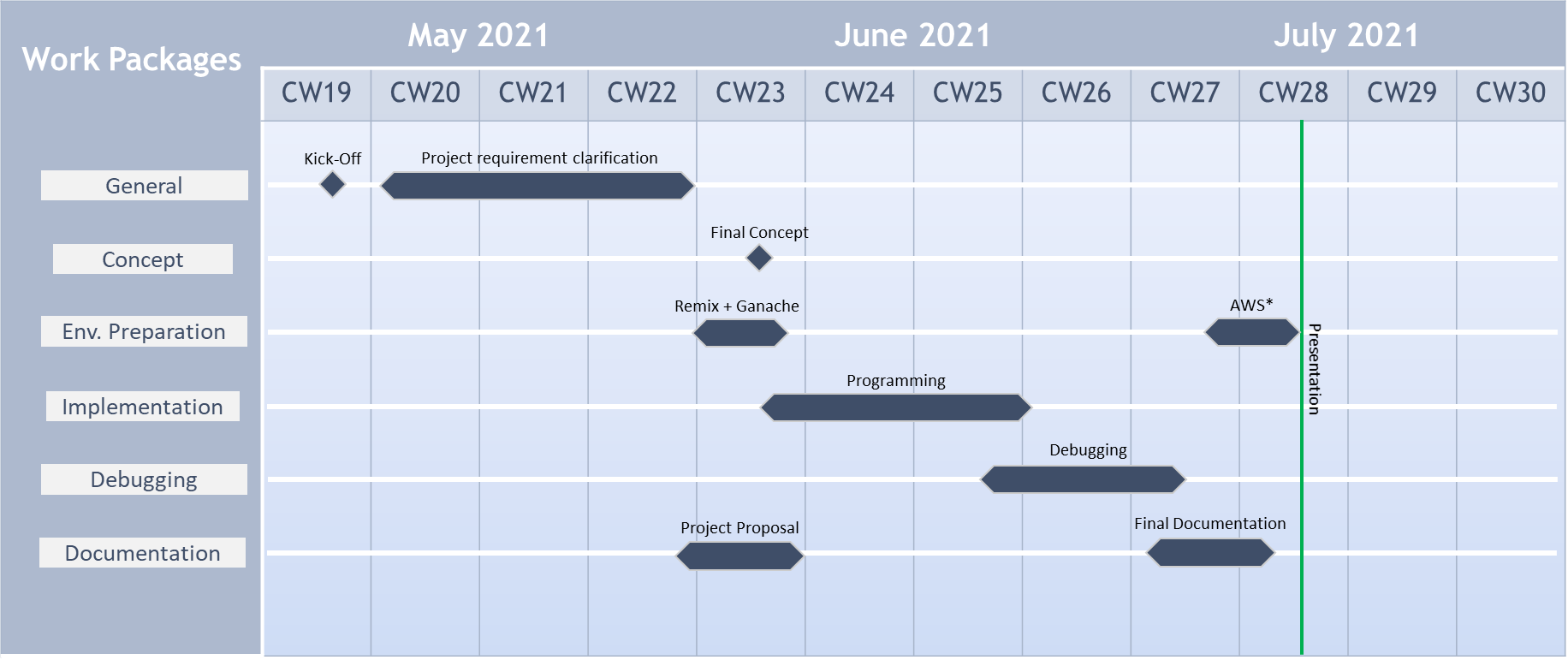
## Requirements

According to the project requirements, it was defined that, Ethereum based smart contracts to be used and based on the concept components other components such as web applications, Rest APIs to be developed as required in order to showcase the full application functionality.

## 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. | | | |

## Timeline



# Concept Overview

## General Concept Overview

Figure : General Concept Overview

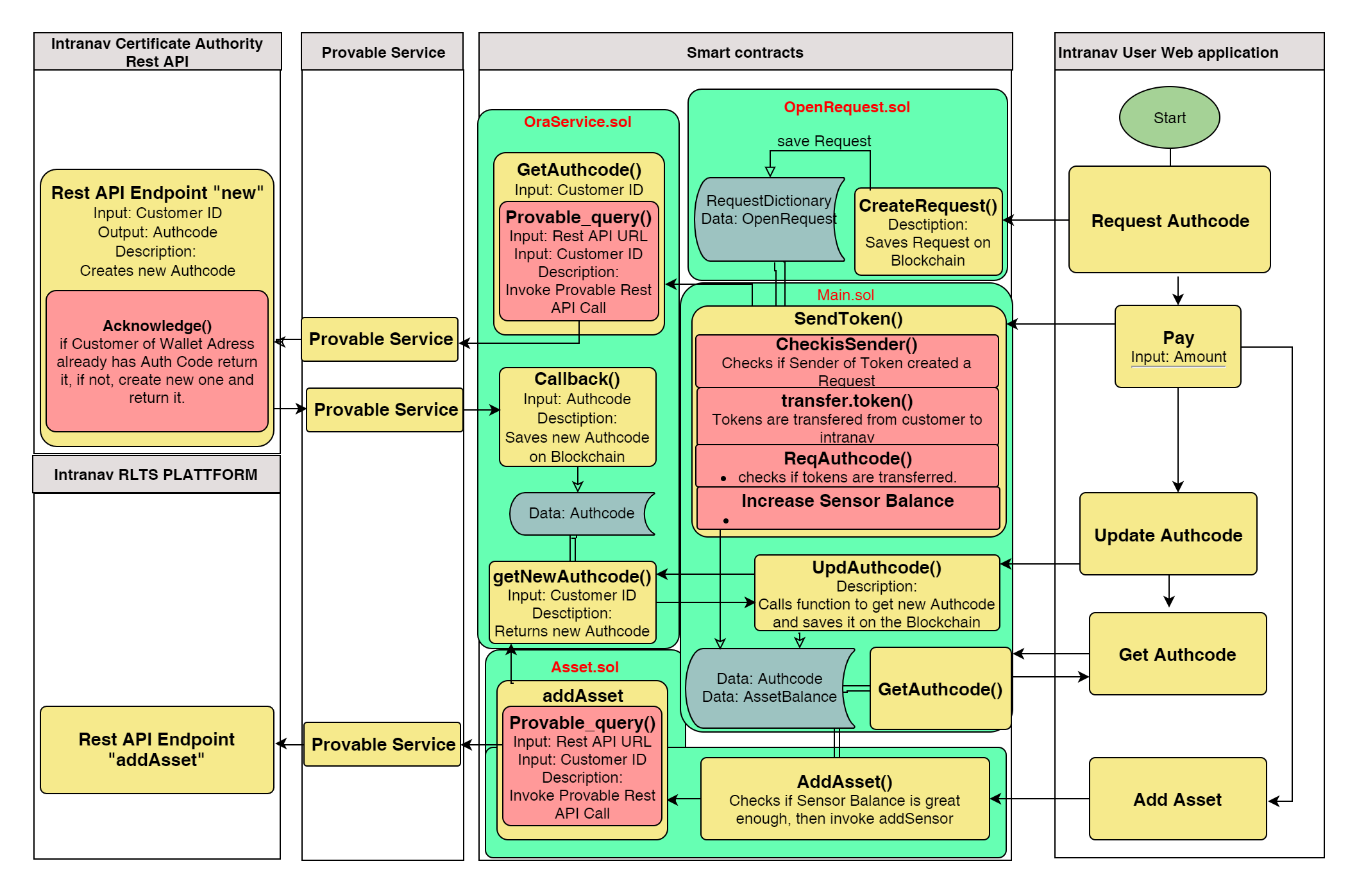


Figure : Detailed Concept Overview

# Blockchain

## What is Blockchain?

Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. An asset can be tangible (a house, car, cash, land) or intangible (intellectual property, patents, copyrights, branding). Virtually anything of value can be tracked and traded on a blockchain network, reducing risk and cutting costs for all involved.

## What is Ethereum?

Ethereum is a technology that lets you send cryptocurrency to anyone for a small fee. It also powers applications that everyone can use and no one can take down. It's called the world's programmable blockchain. Ethereum builds on Bitcoin's innovation, with some big differences. Both let you use digital money without payment providers or banks. But Ethereum is programmable, so you can also use it for lots of different digital assets – even Bitcoin!

This also means Ethereum is for more than payments. It's a marketplace of financial services, games and apps that can't steal your data or censor you.

## Smart Contract

### What is it?

Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary’s involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met.

### How does it work

Smart contracts work by following simple “if/when…then…” statements that are written into code on a blockchain. A network of computers executes the actions when predetermined conditions have been met and verified. These actions could include releasing funds to the appropriate parties, registering a vehicle, sending notifications, or issuing a ticket. The blockchain is then updated when the transaction is completed. That means the transaction cannot be changed, and only parties who have been granted permission can see the results.

Within a smart contract, there can be as many stipulations as needed to satisfy the participants that the task will be completed satisfactorily. To establish the terms, participants must determine how transactions and their data are represented on the blockchain, agree on the “if/when...then…” rules that govern those transactions, explore all possible exceptions, and define a framework for resolving disputes.

Then the smart contract can be programmed by a developer – although increasingly, organizations that use blockchain for business provide templates, web interfaces, and other online tools to simplify structuring smart contracts.

### Types of Smart Contracts

Assuming the reader has a basic understanding of contracts and computer programming, and building on from our definition of smart contracts, we can roughly classify smart contracts and protocols into the following major categories.

* Smart legal contracts

These are presumably the most obvious kind. Most, if not, all contracts are legally enforceable. Without going into much technicalities, a smart legal contact is one that involves strict legal recourses in case parties involved in the same were to not fulfill their end of the bargain. As previously mentioned, the current legal framework in different countries and contexts lack sufficient support for smart and automated contracts on the blockchain and their legal status is unclear. However, once the laws are made, smart contracts can be made to simplify processes which currently involve strict regulatory oversight such as transactions in the financial and real estate market, government subsidies, international trade etc.

* DAOs

Decentralized Autonomous Organizations, shortly DAO, can be loosely defined as communities that exist on the blockchain. The community may be defined by a set of rules arrived at and put into code via smart contracts. Every action by every participant would then be subject to these sets of rules with the task of enforcing and reaching at recourse in case of a break being left to the program. Multitudes of smart contracts make up these rules and they work in tandem policing and watching over participants.

A DAO called the Genesis DAO was created by Ethereum participants in may of 2016. The community was meant to be a crowdfunding and venture capital platform. In a surprisingly short period of time they managed to raise an astounding $150 million. However, hacker(s) found loopholes in the system and managed to steal about $50 million dollars’ worth of Ethers from the crowdfund investors. The hack and its fallout resulted in a fork of the Ethereum blockchain into two, Ethereum and Ethereum Classic.

* Application logic contracts (ALCs)

If you’ve heard about the internet of things in conjunction with the blockchain, chances are that the matter talked about Application logic contacts, shortly ALC. Such smart contracts contain application specific code that work in conjunction with other smart contracts and programs on the blockchain. They aid in communicating with and validating communication between devices (while in the domain of IoT). ALCs are a pivotal piece of every multi-function smart contract and mostly always work under a managing program. They find applications everywhere in most examples cited here.

## Blockchain Oracle

### The Oracle Problem

The Oracle Problem is defined as the security, authenticity, and trust conflict between third-party oracles and the trustless execution of smart contracts. The digital world needs to know about the physical world.

Oracles retain an enormous amount of power over smart contracts in how they are executed because the data they provide determines how the smart contracts execute. Therefore, data feeds from third-party sources give that data substantial influence over the execution of a smart contract, removing its trustless nature as part of a decentralized network.

Each node needs to calculate(Mine) same result for same input for a contract method. Because of this you cant use an external api from inside a smart contract to get external data. Because for example with an API to get the time, each node would get another time. To get external data inside a smart contract a blockchain Oracle is used.

### How does it work?

### Inside Smart Contract call oracle function to get external data.

1. The Oracle contract requests the data from Oracle Service
2. The Oracle Service invoke a Rest API
3. The Oracle Service gets the data
4. The Oracle Service publishs the data on the blockchain
5. Oracle Contract callback function gets the data
6. The Oracle Contract returns the data
7. All nodes can mine the same result, since the data is published

Figure : Blockchain Oracle Communication

### Oracles used in Ethereum

There are multiple Oracles exist and it is also possible to create your own modular oracle networks to get any customized data into the blockchain. In example Chainlink and Provable (Formerly known as Oraclize) are two most prominent ones. In our applicaton we used Provable, since the Provable Oracle is created to use external REST API.

## How to Communicate with blockchain

Generally, the Blockchain communication happens via different ways and each of them are unique to the blockchain architecture. Ethereum supports JSON RPC API communication and there are available API libraries developed for this task as well.

* JSON RPC API

Ethereum JSON-RPC APIs use a namespace system. RPC methods are clustered into several categories, depending on their usage. All method names are composed of the namespace, an underscore, and the actual method name within the namespace.

For example, the eth\_call method resides in the eth namespace.

It will request data either from the node, execute an EVM function and return a response, or transmit data to the Ethereum network.

* Different Languages and available API libraries for Ethereum Communication

|  |  |
| --- | --- |
| **Language** | **API Library** |
| .Net Framework (C#, F#) | Nethereum |
| Java, JavaScript | web3J, web3.js & Ethers.js |
| Golang | Geth (Go-Ethereum) |
| Others (If no API library exists) | JSON-RPC |

Table : Different Programming Languages and the available API libraries

# 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.

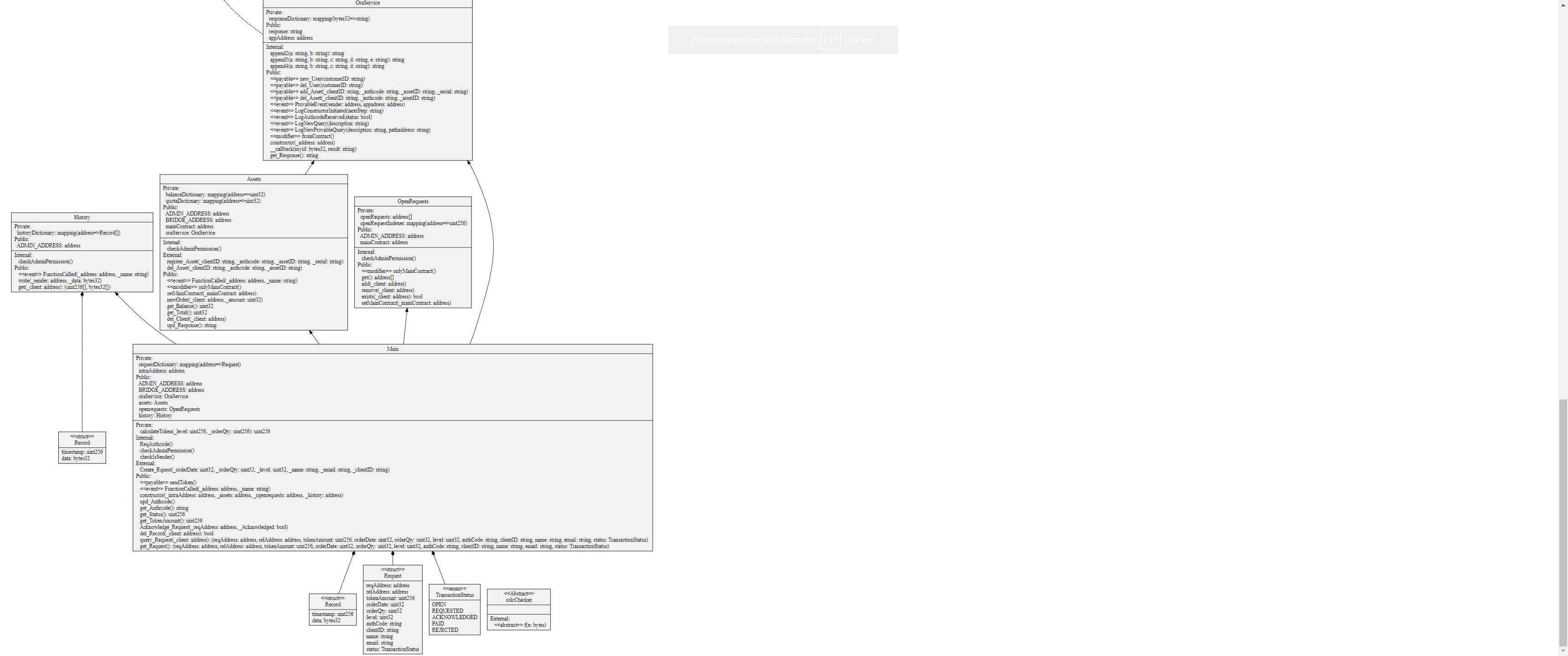


Figure : UML Diagram of the Smart Contract Communication

### 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

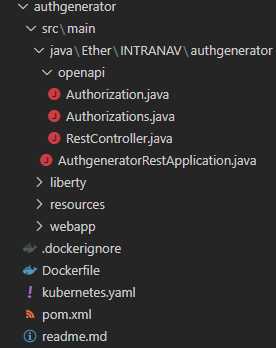
Functionality of Oracle Service Contrace:

* 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

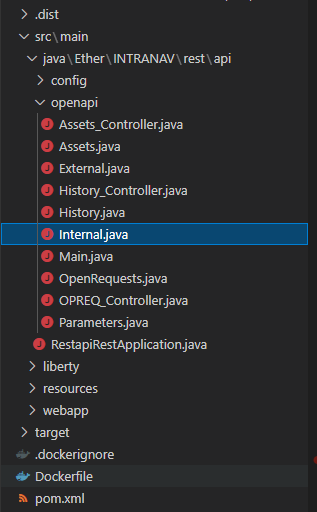


The most important file is RestController.java. This File contains the implementation of the 2 Endpoints the Rest API provides.

## Internal Communication Rest API

We used the Openliberty Framework to create this REST API. The source is in 04\_InternalRestAPI. To communicate with the blockchain web3j library is used. This API acknowledges the Open Requests.

## Project structure



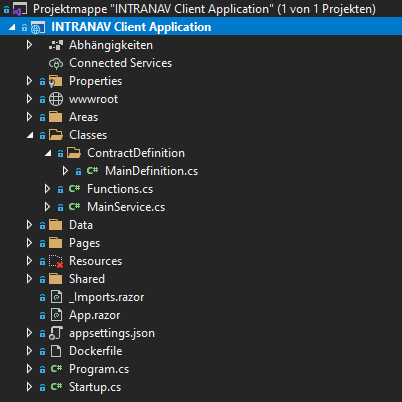
In internal.java the REST API endpoints are defined.

## 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

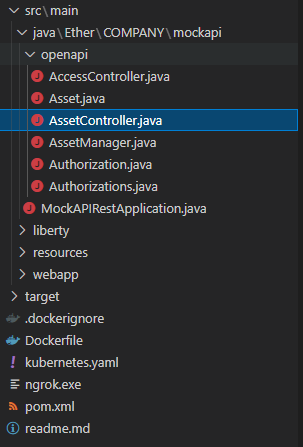


In “Functions.cs” the main functions like newRequest() and transfertoken() are implemented. The main functionality of the User Application can be modified here. These functions invoke nethereum functions which are implemented in “MainService.cs”. Netherum is the interface between the Web Application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

## INTRANAV RTLS Rest API

We emulated the Intranav RTLS API because the existing one needs too much informations to register an asset, this would exceed the size of the smart contract. Our simulated REST API is shown here.

# Project Structure



The REST API Endpoints to register an Asset are defined in AssetControler.java

# Decision Matrix

|  |  |
| --- | --- |
| Ganache CLI | We used the command line version of ganache (Ganache CLI), because we wanted to containerize the Ethereum simulation. The command line version of ganache can be easily containerized. |
| 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. Further Informations at 6.1.2. |
| Docker | To run the project on different Operating Systems, we containerized the used REST APIS and web application. Docker can be installed on different Operating Systems. The containers can be deployed in docker. |
| 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. |
| Web3j | 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. Web3j is used at the REST APIs and Web application. |
| Nethereum Generator | 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. The library is used at the web application. |
| Visual Studio Code | 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. |
| Openliberty framework | Openliberty is an open application framework. It supports full MicroProfile standard. The MicroProfile OpenAPI is used at this project to create the Rest APIs inside the Openliberty framework. Additionally, Openliberty deploys to every major cloud platform, including Docker, Kubernetes, and Cloud Foundry. |

# 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/>

There are other IDEs available for Solidity Smart Contract development. Most used ones are [Atom](https://atom.io/), [Truffle](https://www.trufflesuite.com/) and [VS code](#_Visual_Studio_Code) with solidity extension. A simple comparison can be seen below.

|  |  |
| --- | --- |
| **Remix IDE** | **VS Code with Plugins** |
| All-in-One solution | Require multiple plugin installations and configurations |
| Can import any source libraries locally or over the network. | Does not support Direct import of supporting contracts over external sources. (Ex. Github) |
| Automation is not currently possible. | Some testing can be automated. |
| Good for small-mid ranged projects. | Enterprise grade big solidity projects with multiple connections would benefit. |
| Do not support code formatting. | There are plugins available to support code formatting. |
| Ethereum Specific development with multiple developers backing the process. | Single developers developing the plugins separately and therefore takes time for bug fixing. |

Table : Remix IDE vs VS-Code for Solidity Programming

### 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.

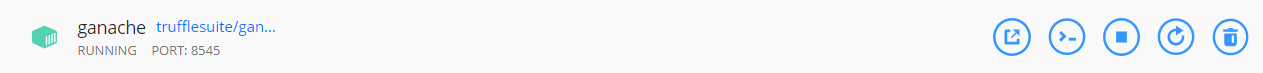
How to : Open Docker Software, and double click on the running ganache container. 

Figure : Ganache Docker Container in Docker GUI.

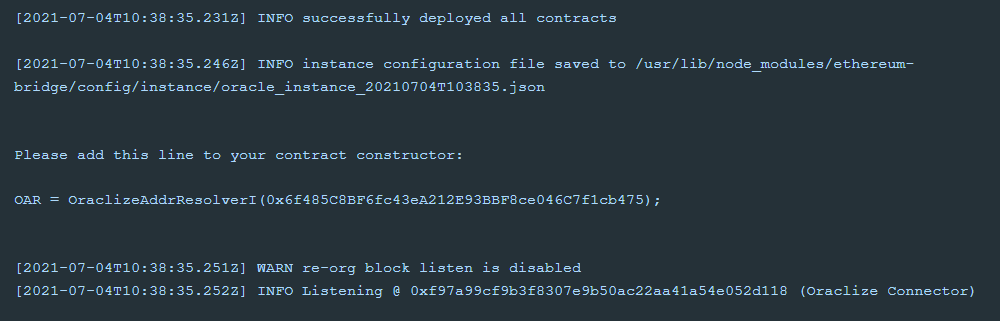
Example : 

Figure : Docker CLI log from Ganache Docker with Ethereum Bridge Address

## Deploy Smart Contracts

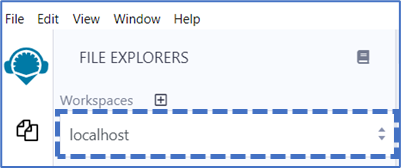
**Step 1:** Open Remix IDE and set workspace to “localhost”

Figure : Set Workspace

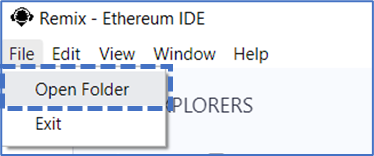


Figure : Open Folder view

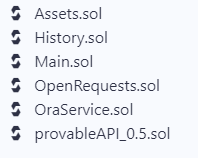
**Step 2:** Open the “\02\_SmartContract” folder from File Menu.

Figure : Smart Contracts

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

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

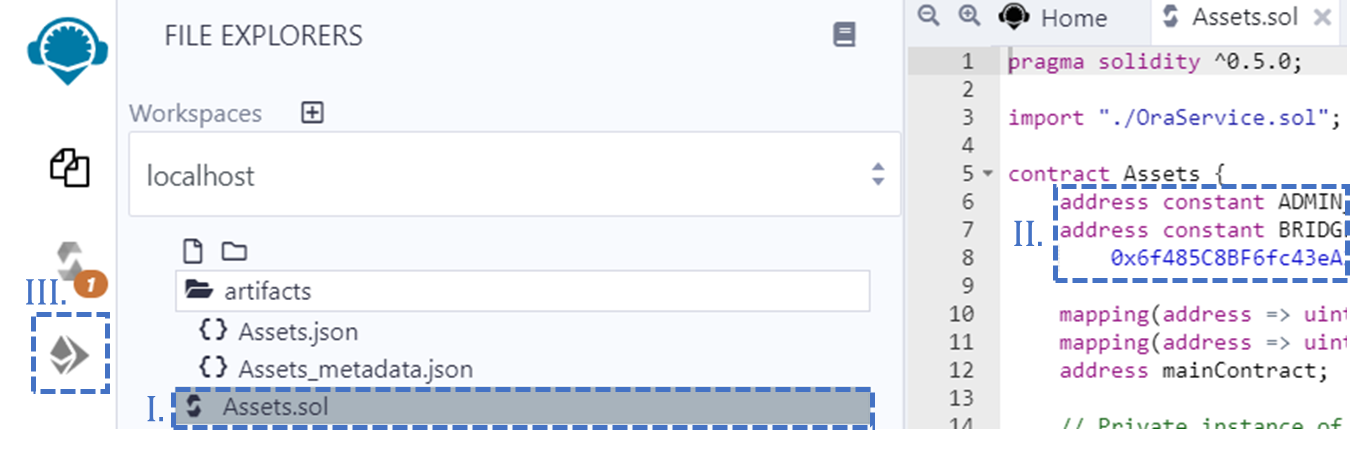


Figure : Assets Smart Contract Deployment – Step 1

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

**Step 4:**

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.

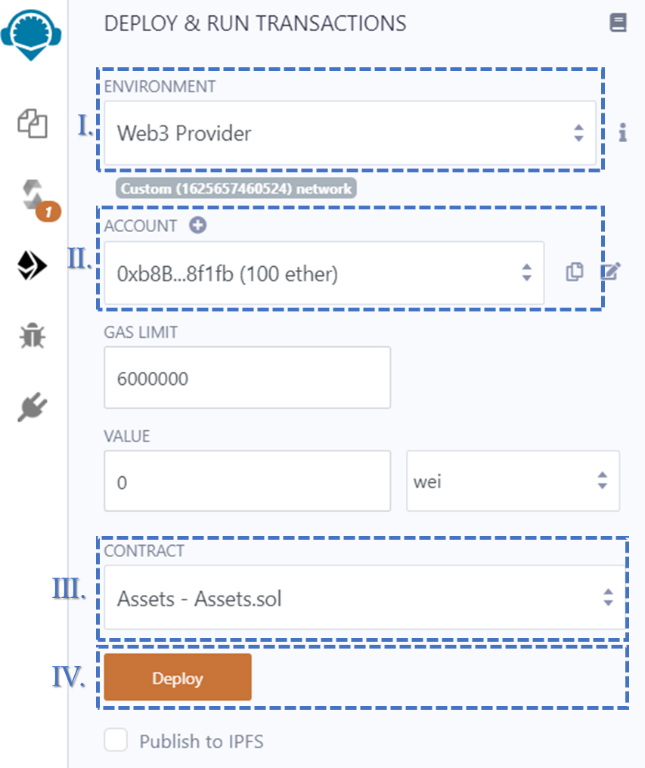
1. **Set Environment**

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.

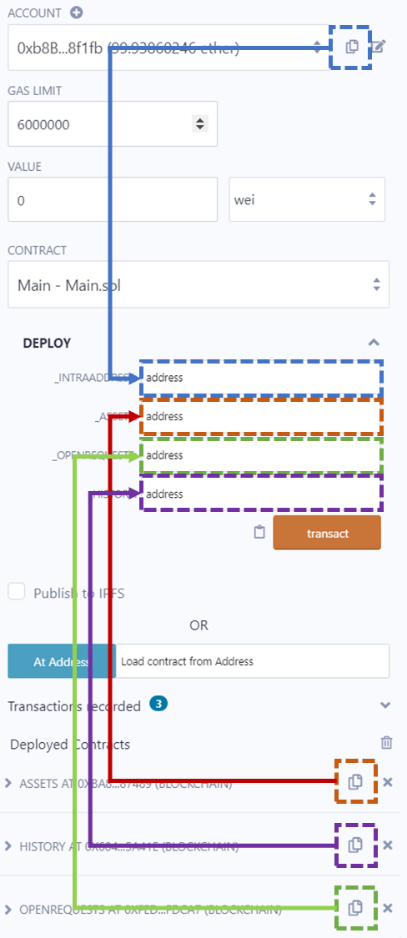
Since the Main Contract is larger in size, please Enable “Contract Optimization” in Compile Tab.

Figure : Main Contract Deployment

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.

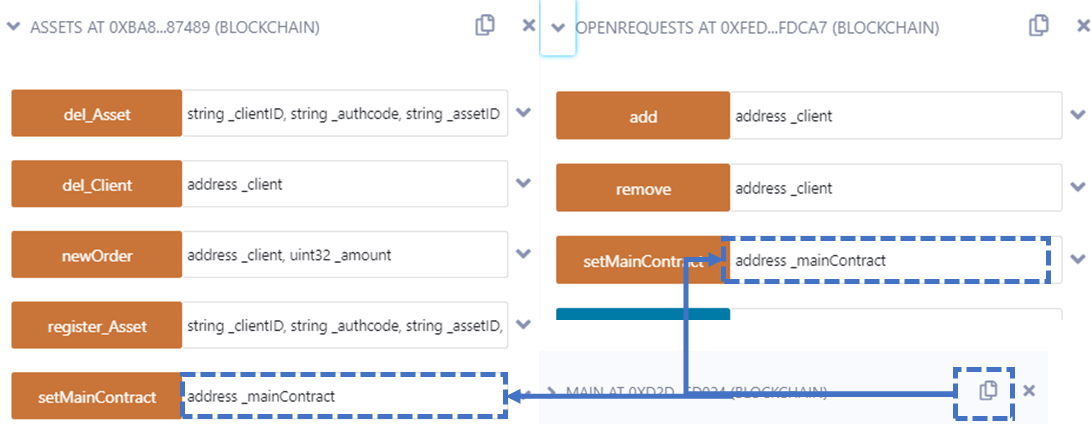


Figure : Set Main Contract address into Assets and OpenRequests.

## 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.

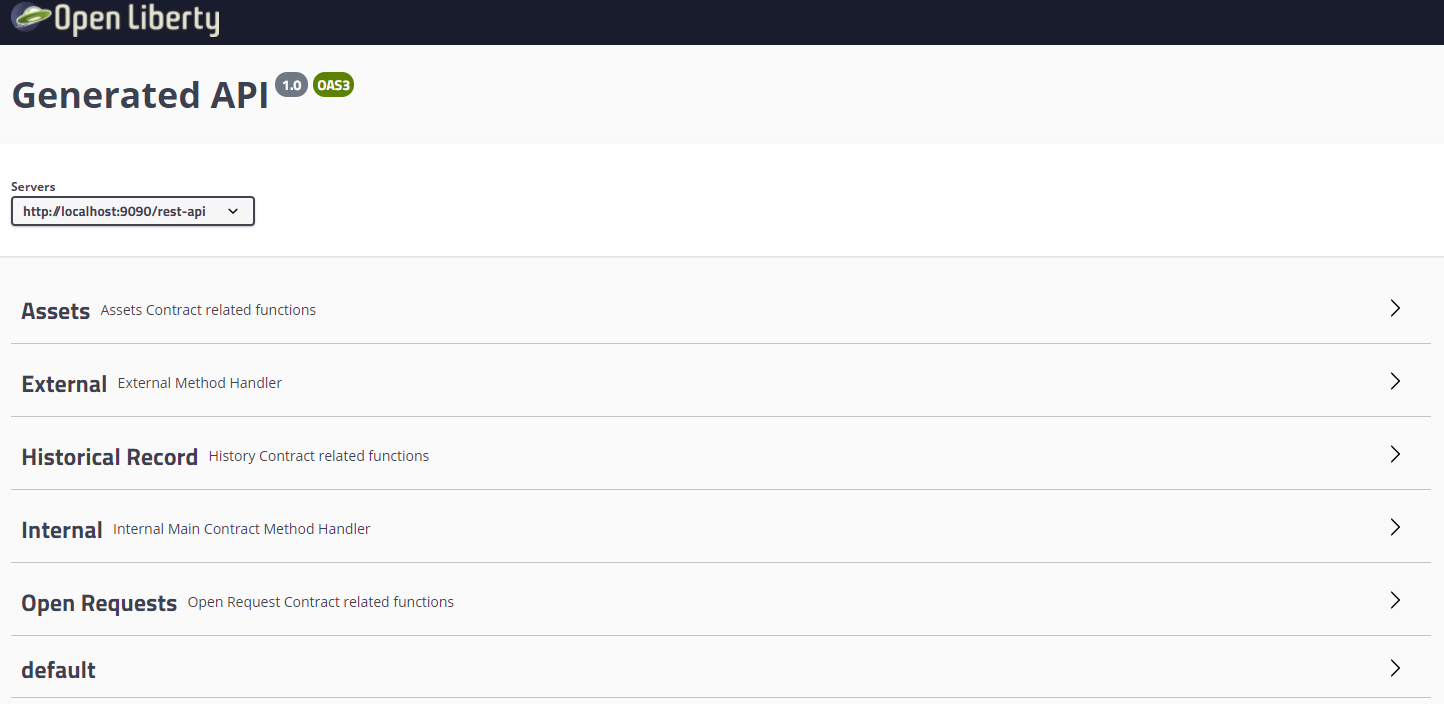


Figure : Internal Communication Rest API

## 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 :

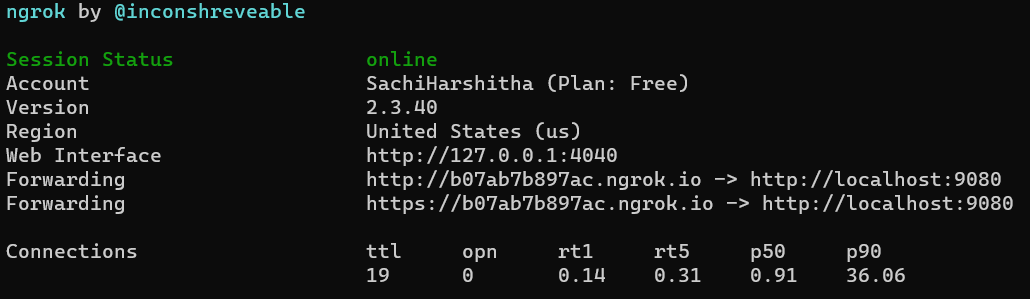


Figure : NGROK local tunneling CLI window.

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

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

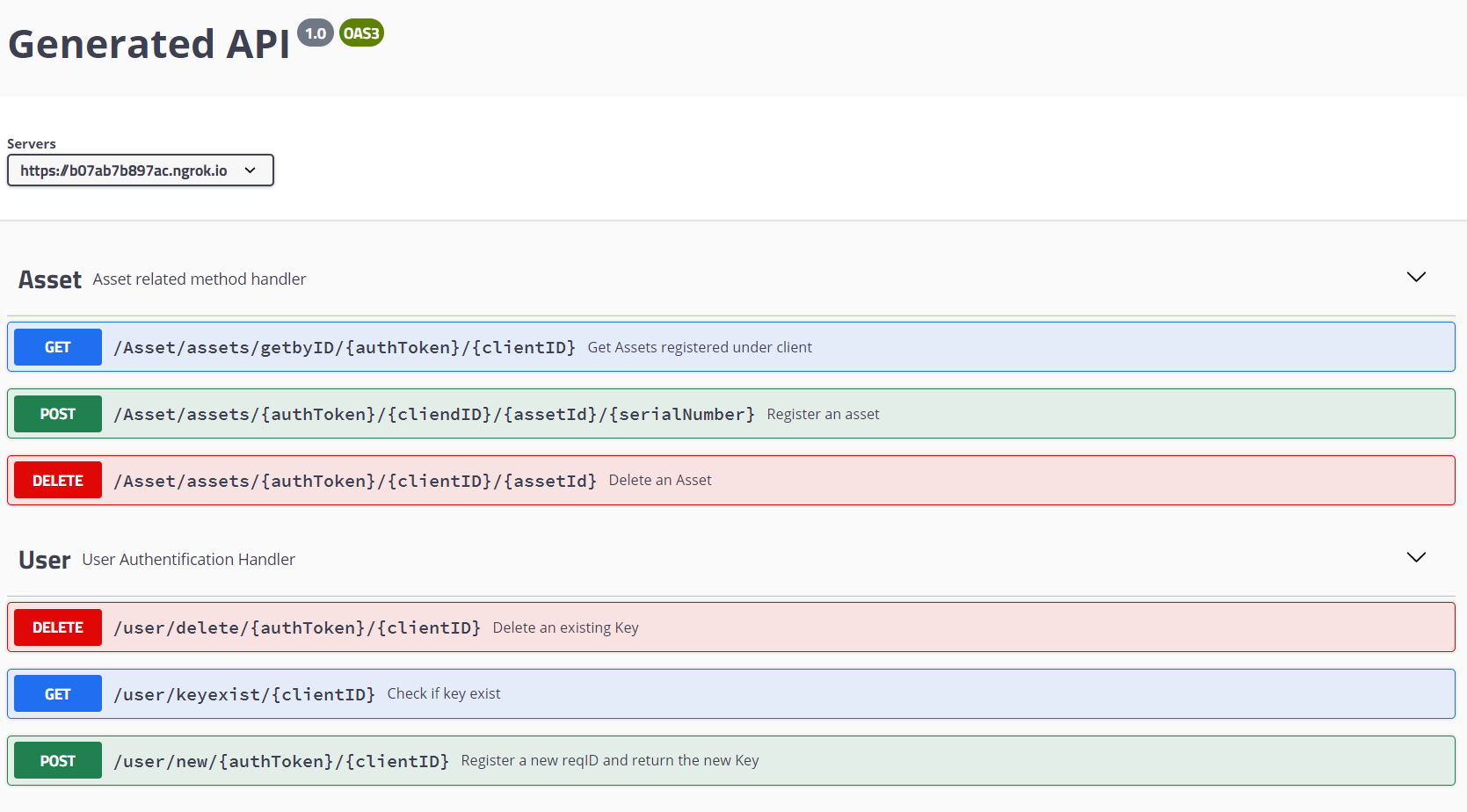


Figure : Sample COMPANY Rest API

## Run the Client Application Web Service

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

Example:

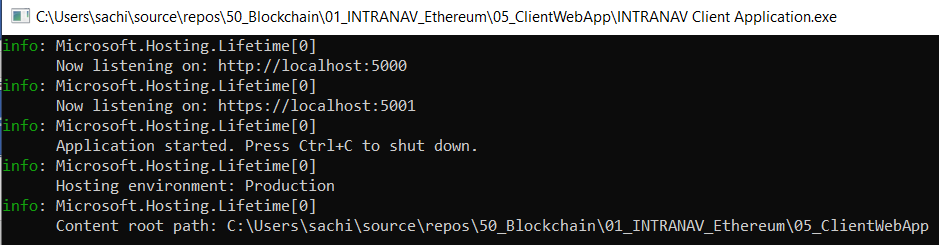


Figure : CMD window of Blazor server app

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.

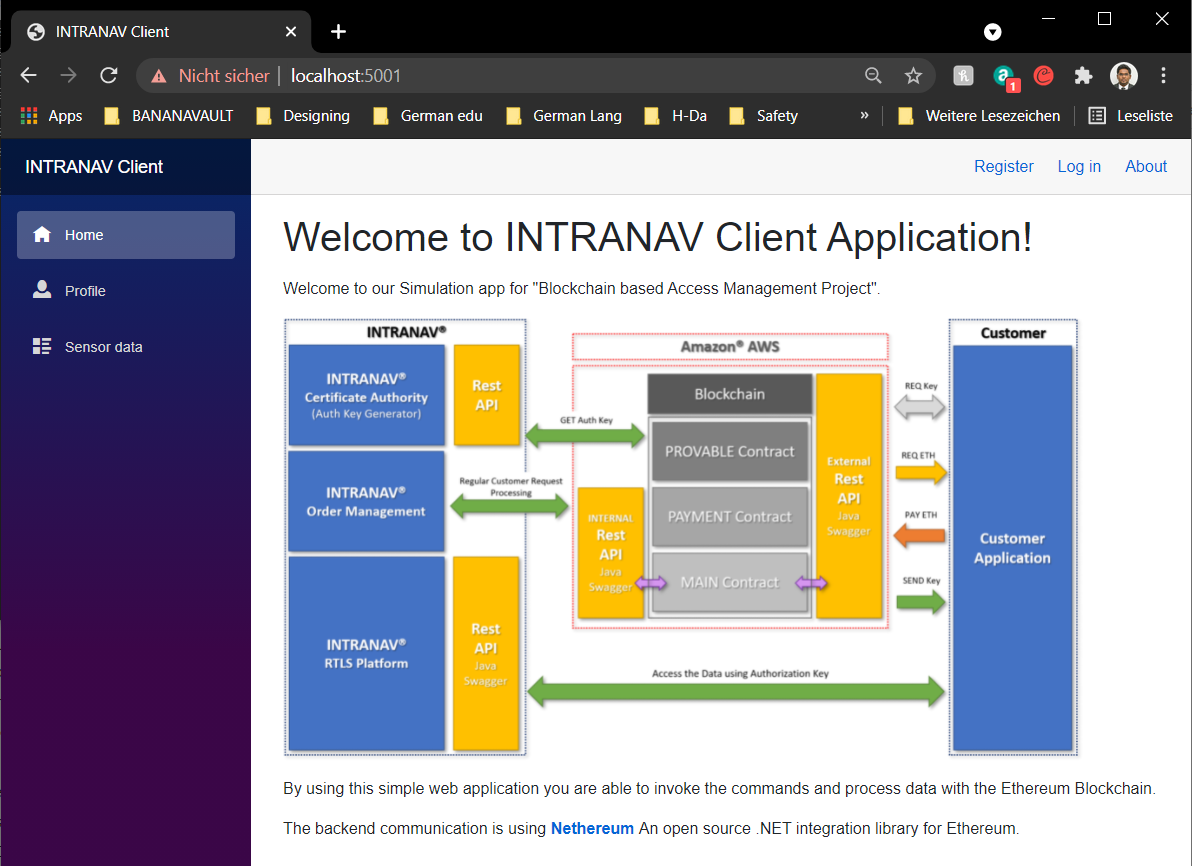


Figure : Client App User Interface

**Step 2:** Initialize the application

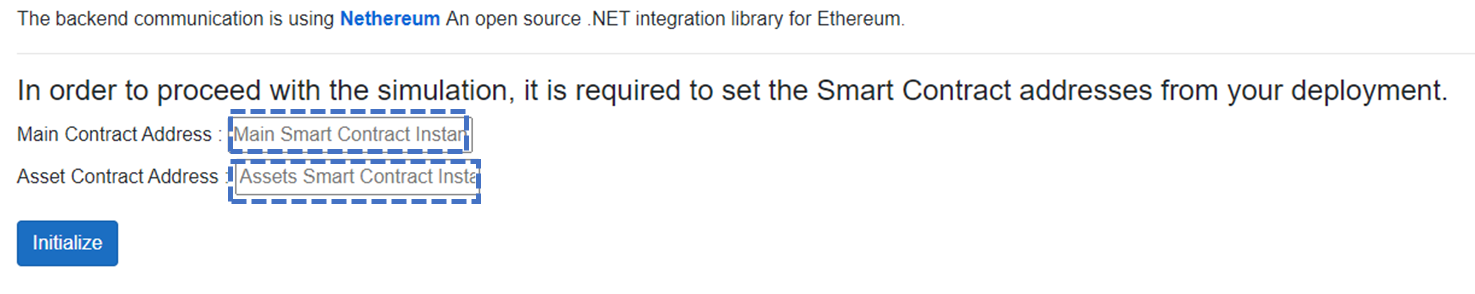


Figure : Client Application Smart Contract address definition.

Copy and paste the Main and Assets smart contract instance address from above section “5.4 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.

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

Table : Application Components and the AWS end points.

## How to deploy

Based on the specific AWS systems used in different applications the deployment do differ from each other.

For Fargate Cluster please refer to:

<https://aws.amazon.com/getting-started/hands-on/deploy-docker-containers/>

For Kubernetes Cluster with Docker Container please refer to:

<https://openliberty.io/guides/cloud-aws.html#configuring-the-aws-cli>

For Elastic Beanstalk with dot net Blazor application please refer to:

<https://aws.amazon.com/blogs/developer/run-blazor-based-net-web-applications-on-aws-serverless/>

# Source

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

Project Data:

<https://github.com/SachiHarshitha/HDa_TeamProject_SS21>

# Bibliography

1. <https://www.ibm.com/topics/what-is-blockchain>
2. <https://ethereum.org/en/what-is-ethereum/>
3. <https://www.ibm.com/topics/smart-contracts>
4. <https://www.ibm.com/topics/smart-contracts>
5. <https://ethereum.org/en/developers/docs/smart-contracts/>
6. <https://medium.com/@teexofficial/what-are-oracles-smart-contracts-the-oracle-problem-911f16821b53>
7. <https://ostechnix.com/blockchain-2-0-explaining-smart-contracts-and-its-types/>

# Appendix I - Figures

[Figure 1 : Project deliverable structure 3](#_Toc78105483)

[Figure 2 : General Concept Overview 5](file:///C:\Users\sachi\source\repos\50_Blockchain\HDa_TeamProject_SS21\00_Documentation\M03_TeamProject_Haid_UWB-Report.docx#_Toc78105484)

[Figure 3 : Detailed Concept Overview 5](#_Toc78105485)

[Figure 4 : Blockchain Oracle Communication 8](file:///C:\Users\sachi\source\repos\50_Blockchain\HDa_TeamProject_SS21\00_Documentation\M03_TeamProject_Haid_UWB-Report.docx#_Toc78105486)

[Figure 5 : UML Diagram of the Smart Contract Communication 10](#_Toc78105487)

[Figure 6 : Ganache Docker Container in Docker GUI. 19](#_Toc78105488)

[Figure 7 : Docker CLI log from Ganache Docker with Ethereum Bridge Address 19](#_Toc78105489)

[Figure 8 : Set Workspace 19](file:///C:\Users\sachi\source\repos\50_Blockchain\HDa_TeamProject_SS21\00_Documentation\M03_TeamProject_Haid_UWB-Report.docx#_Toc78105490)

[Figure 9 : Open Folder view 20](file:///C:\Users\sachi\source\repos\50_Blockchain\HDa_TeamProject_SS21\00_Documentation\M03_TeamProject_Haid_UWB-Report.docx#_Toc78105491)

[Figure 10 : Smart Contracts 20](file:///C:\Users\sachi\source\repos\50_Blockchain\HDa_TeamProject_SS21\00_Documentation\M03_TeamProject_Haid_UWB-Report.docx#_Toc78105492)

[Figure 11 : Assets Smart Contract Deployment – Step 1 20](#_Toc78105493)

[Figure 12 : Deploy Contract 21](file:///C:\Users\sachi\source\repos\50_Blockchain\HDa_TeamProject_SS21\00_Documentation\M03_TeamProject_Haid_UWB-Report.docx#_Toc78105494)

[Figure 13 : Main Contract Deployment 22](file:///C:\Users\sachi\source\repos\50_Blockchain\HDa_TeamProject_SS21\00_Documentation\M03_TeamProject_Haid_UWB-Report.docx#_Toc78105495)

[Figure 14 : Set Main Contract address into Assets and OpenRequests. 22](#_Toc78105496)

[Figure 15 : Internal Communication Rest API 24](#_Toc78105497)

[Figure 16 : NGROK local tunneling CLI window. 25](#_Toc78105498)

[Figure 17 : Sample COMPANY Rest API 25](#_Toc78105499)

[Figure 18 : CMD window of Blazor server app 26](#_Toc78105500)

[Figure 19 : Client App User Interface 26](#_Toc78105501)

[Figure 20 : Client Application Smart Contract address definition. 26](#_Toc78105502)

# Appendix II - Tables

[Table 1 : Different Programming Languages and the available API libraries 9](#_Toc78105539)

[Table 2 : Remix IDE vs VS-Code for Solidity Programming 16](#_Toc78105540)

[Table 3 : Application Components and the AWS end points. 27](#_Toc78105541)