Prototype

What has been done

Structure

This proof of work includes core functionality code, web app UI wireframes and operational diagrams.
The code is split up into projects (PIPO.Contracts, PIPO.Interface, PIPO.Verify) and test projects, the projects are for the code and test projects contain unit tests implementing the core code.

```
❯ tree -L 2
.
├── Design
│   └── Wireframes.pdf
├── LICENSE
├── Operational\ diagrams
│   ├── CreateAccount.png
│   ├── CreateAccount.umdx
│   ├── RegiserWork.png
│   └── RegiserWork.umdx
├── PIPO
│   ├── PIPO.Contracts
│   ├── PIPO.Interface
│   ├── PIPO.Interface.Test
│   ├── PIPO.Verify
│   ├── PIPO.Verify.Test
│   ├── PIPO.sln
│   ├── PIPO.sln.DotSettings.user
│   ├── Spec
│   ├── TestChains
│   └── launch.sh
├── Project-Specification.pdf
├── README.md
└── prototype.pages

10 directories, 12 files
```

Prerequisites

Dotnet

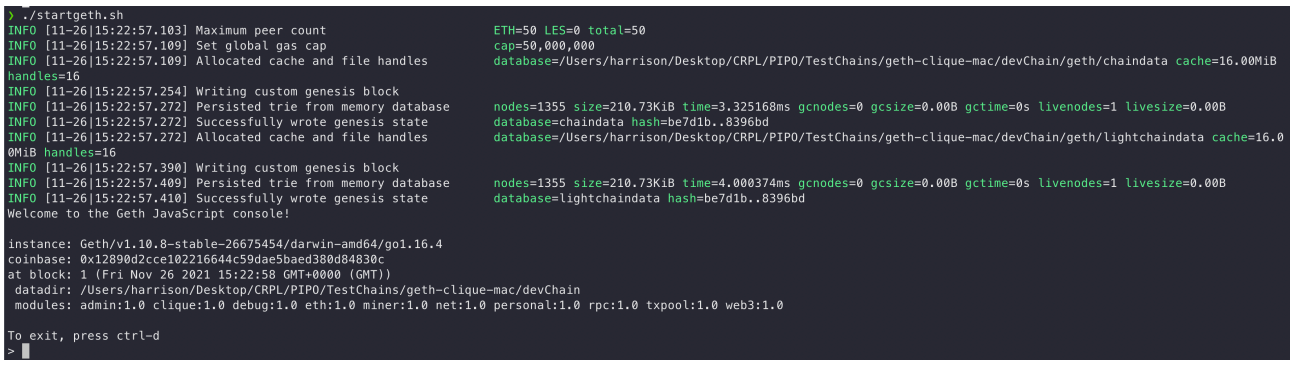
You'll need the latest version of .NET you can get it at: https://dotnet.microsoft.com/download

```
❯ dotnet --version
6.0.100
```

Then run **dotnet build** in the PIPO directory which should install all dependencies and compile successfully if not **dotnet restore** will sort out dependencies.

Running the test blockchain

The tests need a locally hosted private blockchain, I'm using **Geth** for this with a preconfigured account and options (https://github.com/Nethereum/TestChains/). To run this you have to launch ***startgeth.sh*** in ***geth-clique-mac*** or ***startgeth.bat*** in ***geth-clique-windows.***



If you don't see this you might need to run
chmod +x startgeth.sh and ***chmod +x geth***

If this still doesn't work you may need to follow the geth install guide at: https://geth.ethereum.org/docs/install-and-build/installing-geth

Running the tests

The tests can be run from the terminal if you don't have an IDE like visual studio or Jet-brains Rider. You can run all of them with "dotnet test" in the PIPO directory, or individually,

```
dotnet test --filter PIPO.Verify.Test.When_Comparing_Works.Should_Be_The_Same
```

or all the tests in the fixture.

```
dotnet test --filter PIPO.Verify.Test.When_Comparing_Works
```

What has been done?

1. Connect to a local test blockchain

This functionality is tested in the **"When_Connecting_To_Private_Blockchain"** Test fixture.

2. Develop and compile a smart contract

Smart contracts for the Ethereum blockchain are written in the **Solidity** language (these can be found: "*PIPO/PIPO.Contracts/Contracts/*") these need to be compiled into byte code to be deployed onto the chain. The C# library that I'm using **Nethereum** provides a generation tool which takes the .sol files compiles them into byte code and generates all the necessary C# classes avoiding pointless duplication of work. (The c# classes are simply a representation of the smart contract written, meaning without the generation I would have to write the contract effectually twice in two different languages)

https://nethereum.readthedocs.io/en/latest/nethereum-codegen-console/

3. Deploy a contract to a chain

This functionality is tested in the **"When_Deploying_Contract"** Test fixture.

4. Connect to the Ethereum “Mainnet”

This functionality is tested in the **“When_Connecting_To_Mainnet_Blockchain”** Test fixture.

5. Connect to a public “Test chain”

This functionality is tested in the **“When_Connecting_To_Ropsten_Blockchain”** Test fixture.

There are three different chains I can interact with when developing this application, the Ethereum mainnet which is the official chain that everyone uses, a local chain using a tool called **Geth** or a public test chain (I’m using **Ropsten**).

These all have their own uses in development.

1. Mainnet is for deploying a finished product.
2. Testnet is for integration/real world testing as it’s the closest you can get to mainnet.
3. Local chain is for unit testing and prototyping, making sure it theoretically works.

6. Verification algorithm proof of concept

This functionality is tested in the **“When_Verifying_Work”** Test fixture.

The verification proof of concept is built in the “*PIPO.Verify/WorksVerificationService.cs*” that uses a SHA512 hash of the uploaded work to create a unique but quickly comparable value to compare between all other works. (As this method will be checking the work against all other works in the system so has to be quick.)

7. Interact with a smart contact on a chain

This functionality is tested in the **“When_Interacting_With_SimpleOwnership”** Test fixture.

Interaction with a smart contract that’s been deployed onto the chain is done through rpc messages, thankfully the **Nethereum** library has all the endpoint necessary to do this, combined with the generated classes from the smart contract files makes it easy to interact with a deployed contract.

See: */PIPO/PIPO.Interface.Test/When_Interacting_With_SimpleOwnership.cs*

8. Create Ethereum account

This functionality is tested in the **“When_Interacting_With_Blockchain”** Test fixture.

Accounts are simply a pair of keys, private and public which can be generated using **Nethereum** then communicated to the chain.

9. Document outlining systems copyright theoretical structure

The document is in “**PIPO/Spec/PIPO.pdf**”

Source Control

https://github.com/MrHarrisonBarker/Crpl/tree/proto