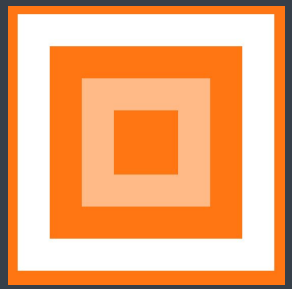


Nethereum + MUD

Working together with Unity, Blazor, Nethereum and MUD

Juan Blanco

juanfranblanco

juanfranblanco

Juan Blanco

Involved in blockchain since 2013 and trying to help the Ethereum ecosystem since 2015 as the creator of Nethereum and the VSCode Solidity extension.

Still excited about Ethereum's potential to change the world, and as such, I believe we will eventually see most applications be part of it.

24 years of experience in .NET and more in IT.

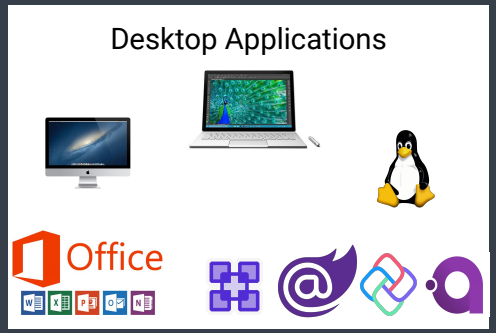
I previously worked as an architect, senior architect, and developer across various sectors, including technology, energy, finance, logistics, and media.

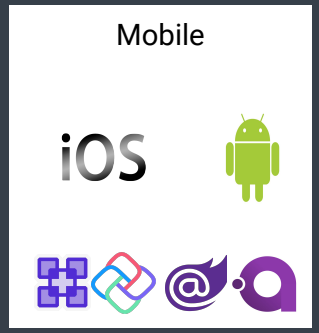
Vision

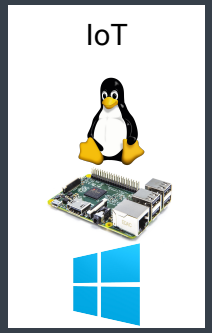
To provide the tools, frameworks and guidance so any .NET developer and application can be created or integrated with Ethereum

Bringing the love of Ethereum to .NET (and vice versa)

.NET Ecosystem and Ethereum







Nethereum features

Core integration

Ethereum node (Json RPC), Smart contract integration, RLP, EVM simulator, signing, ABI Encoding, Chain / log indexing, L2 support

Standards and common Protocols

SIWE, Gnosis Safe, ENS, ERC20, ERC721, ERC1155, EIP-712, …

Other

Merkel, Patricia. .. tries, Data services (etherscan, sourcify), Quorum …

External wallets and signers

Metamask, WalletConnect, Azure Key Vault, AWS Key management, Hardware wallets

Tooling

Nethereum Playground, Code generators, Testing

Templates, Examples and tooling

Getting started templates (smart contracts), Blazor template, SIWE template…
Web, Desktop ,Mobile, Unity examples

MUD Smart contracts, enabling complex Applications & Games

Smart Contract Features:

- *Data Model*: Structured table-based storage, key-value and relational database formats
- *Dynamic Schemas*: Define schemas at runtime, flexibility to evolve post-deployment
- *Typed Data Access*: Auto-generated libraries for type-safe getters and setters, Solidity-compatible encoding
- *Data Indexing*: Events on all data changes, enabling real-time offchain indexing of all the data on chain

MUD Smart contracts, enabling complex Applications & Games

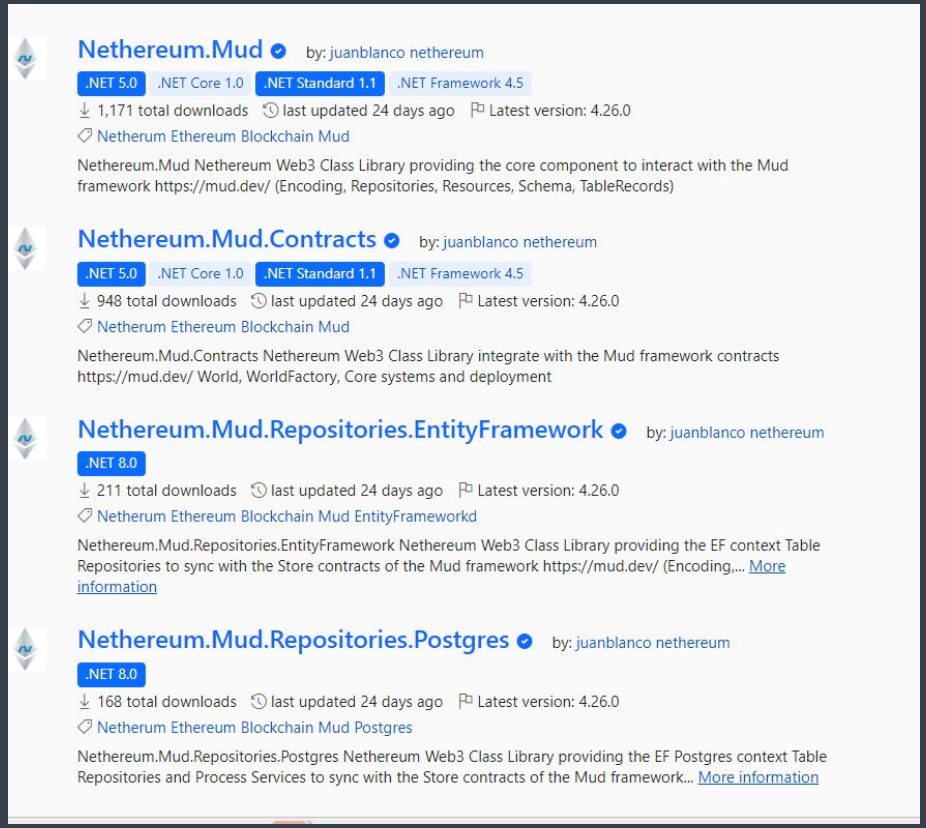
Complex applications and games:

- *Scalability*: Supports advanced data structures, ideal for complex logic in applications
- *Flexibility*: Adapts as needs evolve, allowing new data schemas without redeployment
- *Upgradability:* Upgrade or add new business logic
- *Interoperability*: Data structures can be directly mapped to relational databases, easing offchain integration
- *Delegated authority*: Device wallets / Burner / Session wallets users should not worry about their “main” account when using an application or a game
- *Ever evolving worlds*: New namespaces with new functionality and authority allows you or anyone to expand the “world”

Nethereum.MUD

Nethereum enables .NET developers to seamlessly interact with Ethereum, covering core integration, wallet support, and smart contract interaction

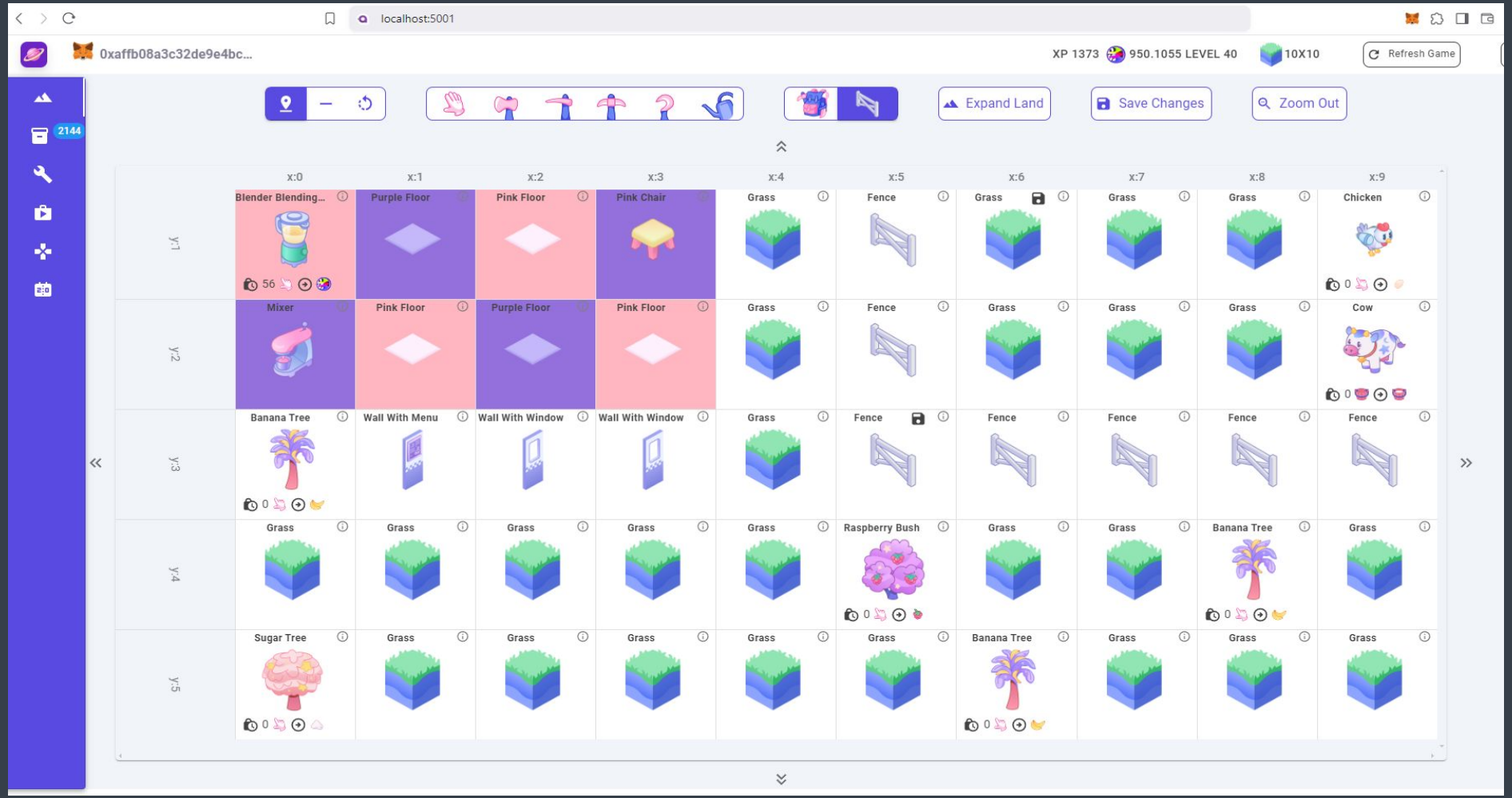
MUD expands it and the whole Ethereum ecosystem by providing data-typed storage and modular logic, enabling more complex games and applications



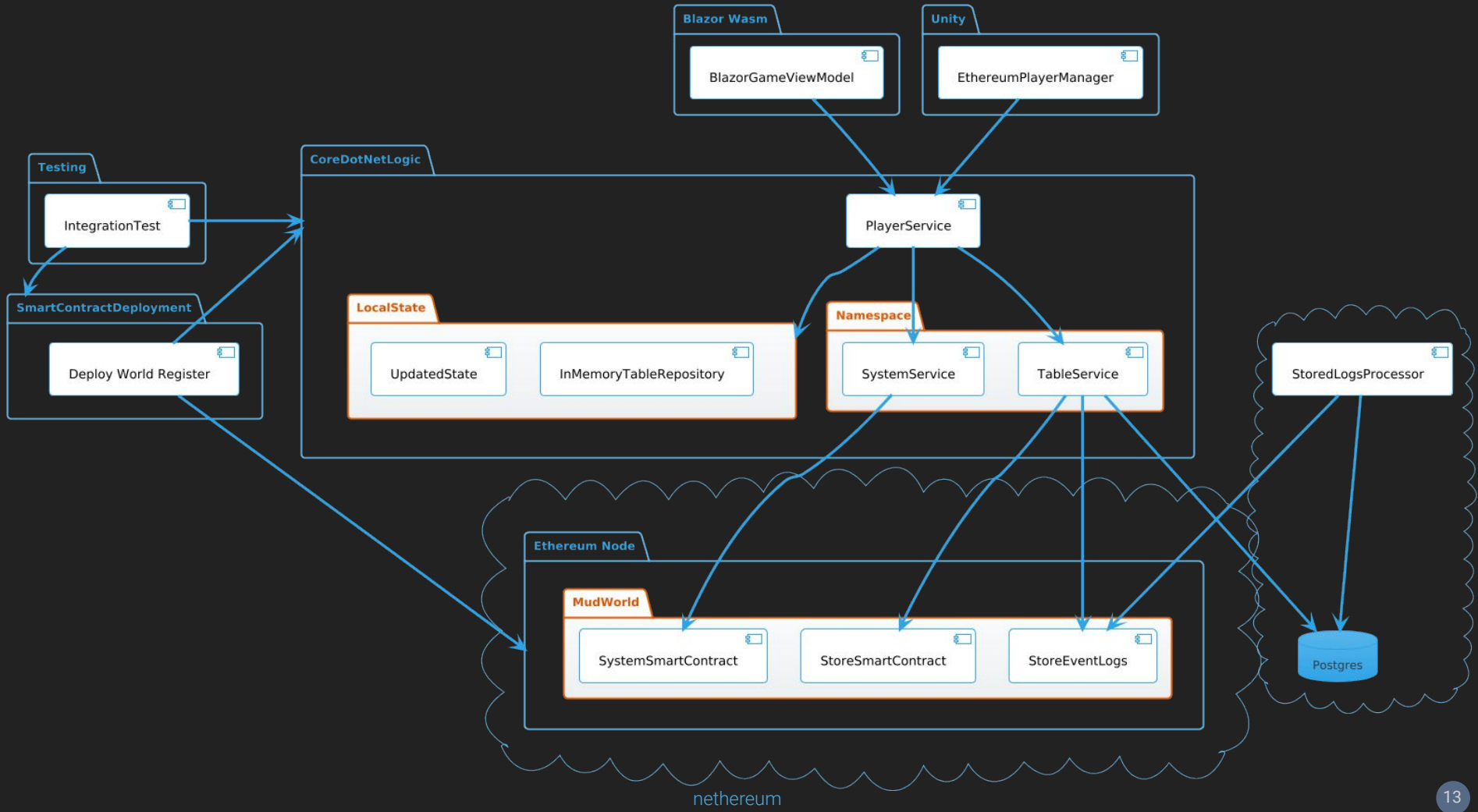
Dogfooding the implementation

The integration of Nethereum.Mud with Cafe Cosmos, and early community feedback has allowed to dogfood the implementation, testing it and refining it in a real world scenario.


CAFE
COSMOS
24:30:06


localhost:5001
0xaffb08a3c32de9e4bc...
XP 1373
950.1055 LEVEL 40
10X10
Refresh Game
Expand Land
Save Changes
Zoom Out
2144
x:0 x:1 x:2 x:3 x:4 x:5 x:6 x:7 x:8 x:9
y:1 y:2 y:3 y:4 y:5
Blender Blending...
Purple Floor
Pink Floor
Pink Chair
Grass
Fence
Chicken
Mixer
Cow
Banana Tree
Wall With Menu
Wall With Window
Raspberry Bush
Sugar Tree


0xAfF...b0494
950.10545830...
00:04:11
1
2
3
4
5
6
7
8


Blazor Wasm
BlazorGameViewModel
Unity
EthereumPlayerManager
Testing
IntegrationTest
CoreDotNetLogic
PlayerService
SmartContractDeployment
Deploy World Register
LocalState
UpdatedState
InMemoryTableRepository
Namespace
SystemService
TableService
StoredLogsProcessor
Ethereum Node
MudWorld
SystemSmartContract
StoreSmartContract
StoreEventLogs
Postgres
nethereum

Let’s explore how each component works by incorporating some functionality.

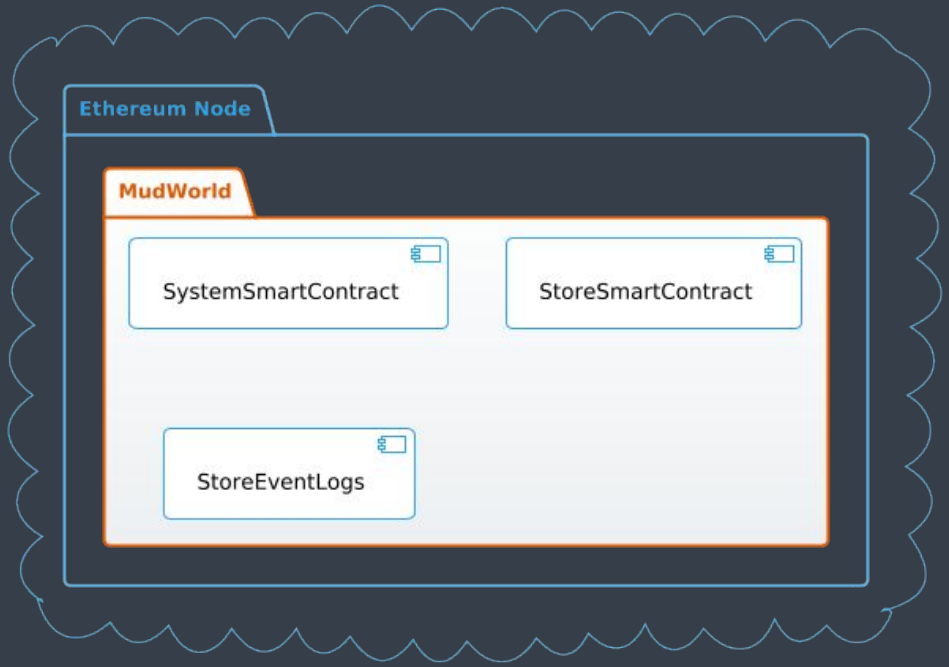
Wouldn’t be great to have a “Crazy Oven” here?

I have an oven…. but not a “Crazy Oven”

And not enough materials to craft a new one.

What if we could purchase the "crazy oven" instead of crafting it from a **catalogue**?

Ethereum smart contracts



Mud Config

```
import { defineWorld } from "@latticexyz/world";

export default defineWorld({
  tables: {
        CatalogueItem:{
          schema:{
            catalogueId: "uint256",
            itemId: "uint256",
            price: "uint256",
            exists: "bool",
          },
          key: ["itemId"]
        }
  },
});
```

Code generation of tables using MUD tablegen

Note: When working on windows it is easier to install pnpm in the git bash or wsl2

Catalogue System - Smart Contract

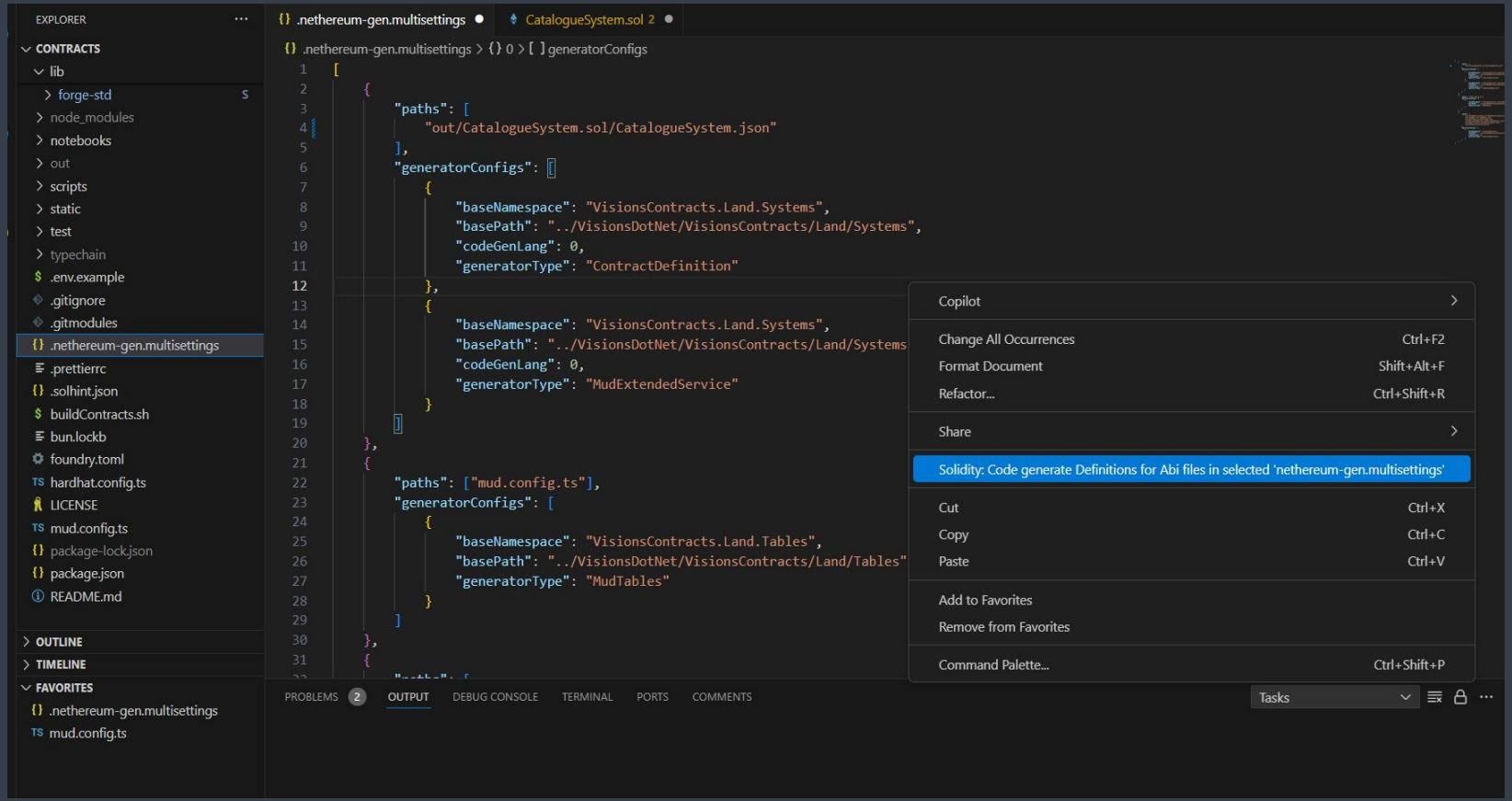
```
// SPDX-License-Identifier: (Apache-2.0 AND CC-BY-4.0)
pragma solidity ^0.8.0;
import {CatalogueItem, CatalogueItemData, LandInfo, ConfigAddresses} from "../codegen/index.sol";
import {LibInventoryManagement} from "./libraries/LibInventoryManagement.sol";
import {LibLandManagement} from "./libraries/LibLandManagement.sol";
import { System } from "@latticexyz/world/src/System.sol";

contract CatalogueSystem is System {

  function purchaseCatalogueItem(uint256 landId, uint256 itemId, uint256 quantity) {
        CatalogueItemData memory catalogueItem = CatalogueItem.get(itemId);
        require(catalogueItem.exists, "Catalogue: item does not exist");
        uint256 price = catalogueItem.price;
        uint256 totalCost = price * quantity;
        uint256 balance = LandInfo.getTokenBalance(landId);
        require(balance >= totalCost, "Catalogue: insufficient balance to purchase item");
        LibInventoryManagement.increaseQuantity(landId, itemId, quantity);
        LibLandManagement.decreaseLandTokenBalance(landId, totalCost);
    }
...

```

Code generation of Nethereum tables and system services

- File named **.nethereum-gen.multisettings**
- Different settings for different set of abis or other files like “mud.config.ts”
- Includes the namespace, path, language and type of generation
- “ContractDefinition” is the standard Nethereum service, deployment etc.
- “MudExtendedService” extends this extra functionality for MUD
- “MudTables”, the tables
- **Once you are done “Right click in vscode”..**

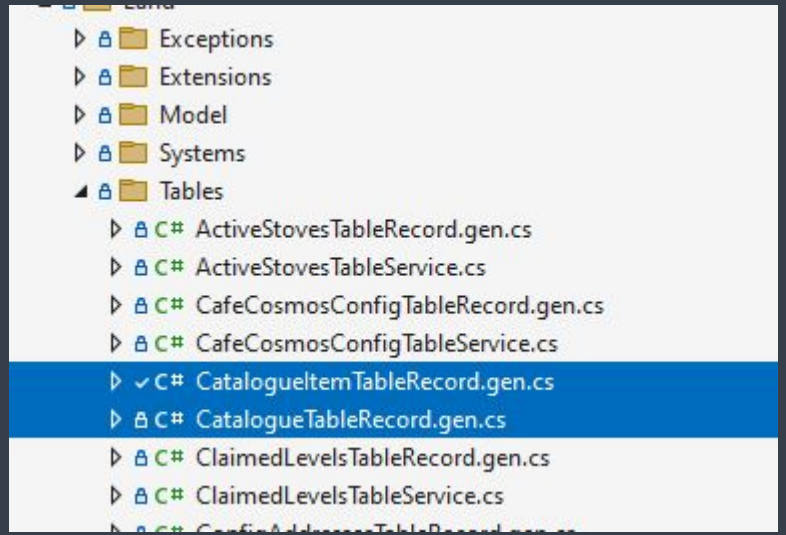
```
[
    {
        "paths": [
            "out/CatalogueSystem.sol/CatalogueSystem.json",
             ...
        ],
        "generatorConfigs": [
            {
                "baseNamespace": "VisionsContracts.Land.Systems",
                "basePath": "../VisionsDotNet/VisionsContracts/Land/Systems",
                "codeGenLang": 0,
                "generatorType": "ContractDefinition"
            },
            {
                "baseNamespace": "VisionsContracts.Land.Systems",
                "basePath": "../VisionsDotNet/VisionsContracts/Land/Systems",
                "codeGenLang": 0,
                "generatorType": "MudExtendedService"
            }
        ]
    },
    {
        "paths": ["mud.config.ts"],
        "generatorConfigs": [
            {
                "baseNamespace": "VisionsContracts.Land.Tables",
                "basePath": "../VisionsDotNet/VisionsContracts/Land/Tables",
                "generatorType": "MudTables"
            }
        ]
    },

  ..

```

Code generation of Nethereum tables and system services

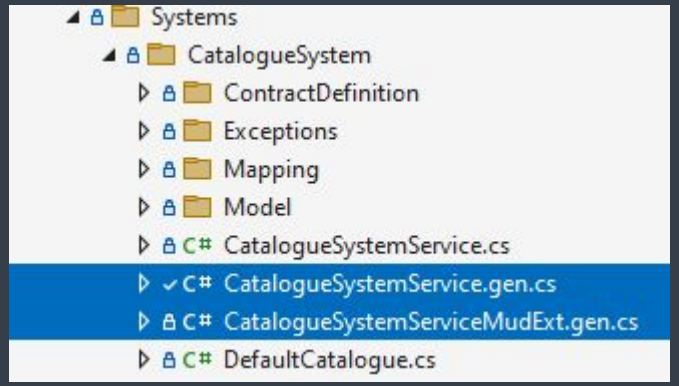


Code generation of Nethereum tables and system services

Tables



Services



TableRecord, TableService, InMemoryTableRepository

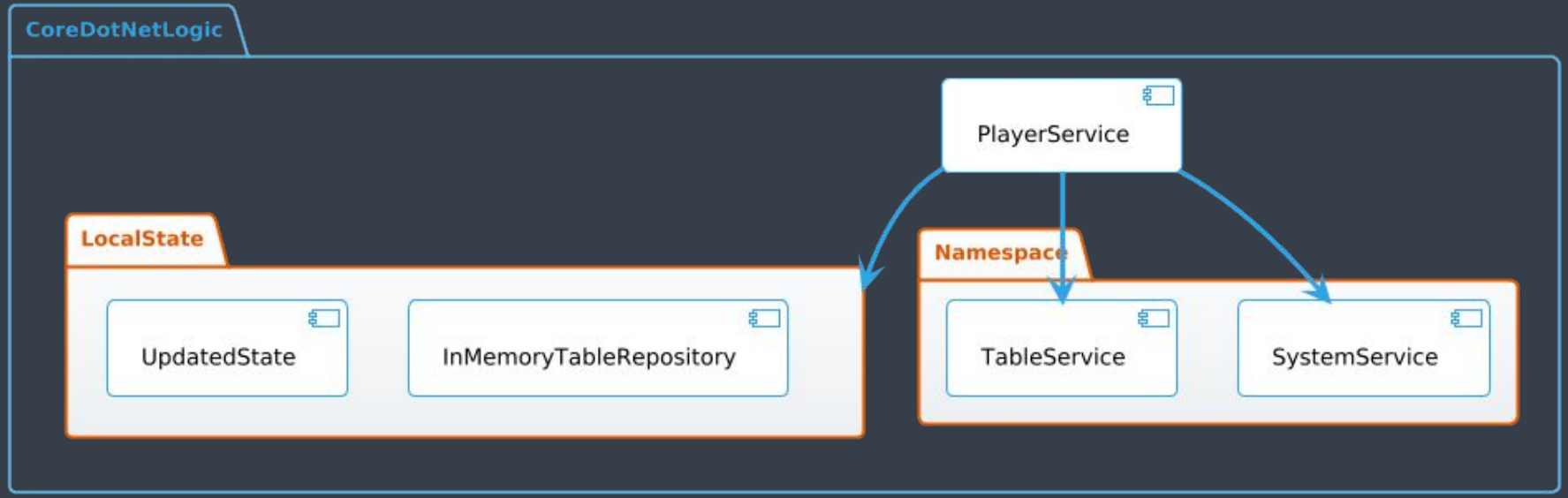


Table Record

- A table record is mapped directly to the Mud.config schema.
- It initialises with both the table name “CatalogueItem” and namespace if specified.
- It includes separately Keys and Values, each property decorated with the Parameter attribute the same way as Nethereum Function Messages, Events etc, although it uses a different packed encoder / decoder.

```
using Nethereum.ABI.FunctionEncoding.Attributes;
using Nethereum.Mud;
using Nethereum.Mud.Contracts.Core.Tables;
using Nethereum.Web3;
using System.Collections.Generic;
using System.Numerics;

namespace VisionsContracts.Land.Tables
{
    public partial class CatalogueItemTableRecord :
                            TableRecord<CatalogueItemTableRecord.CatalogueItemKey,
                                        CatalogueItemTableRecord.CatalogueItemValue>
    {
        public CatalogueItemTableRecord() : base("CatalogueItem")
        {

        }

        public partial class CatalogueItemKey
        {
            [Parameter("uint256", "itemId", 1)]
            public virtual BigInteger ItemId { get; set; }
        }

        public partial class CatalogueItemValue
        {
            [Parameter("uint256", "catalogueId", 1)]
            public virtual BigInteger CatalogueId { get; set; }
            [Parameter("uint256", "price", 2)]
            public virtual BigInteger Price { get; set; }
            [Parameter("bool", "exists", 3)]
            public virtual bool Exists { get; set; }
        }
    }
}
```

Table Service

- The table service maps the table record, key and value types to a service.
- This enables to set records or get records from the chain, or query table repositories, logs or other providers using the table resource id.
- Partial class so it can be extended with custom logic.

```
using Nethereum.ABI.FunctionEncoding.Attributes;
using Nethereum.Mud;
using Nethereum.Mud.Contracts.Core.Tables;
using Nethereum.Web3;
using System.Collections.Generic;
using System.Numerics;

namespace VisionsContracts.Land.Tables
{
    public partial class CatalogueItemTableService : TableService
                                        <CatalogueItemTableRecord,
                                         CatalogueItemTableRecord.CatalogueItemKey,
                                         CatalogueItemTableRecord.CatalogueItemValue>
    {

        public CatalogueItemTableService(IWeb3 web3, string contractAddress) :
        base(web3,  contractAddress) { }
    }
}
```

Table Service

- It enables to register the table in MUD, so the schema is accessible.

```
var web3 = TestAccounts.GetAccount1Web3();
var namespaceAddress = "0x000000000";
var catalogueItemTableService = new CatalogueItemTableService(web3, namespaceAddress);

//mud resource
var resourceId = catalogueItemTableService.Resource.Name;

//register the table
var transactionHash = await catalogueItemTableService.RegisterTableRequestAsync();
```

Table Service

- You can set the record directly onto the table
- Get it from the chain by its key
- Get all the data related to that table using the logs

```
// set or get the record  directly with the chain
transactionHash = await catalogueItemTableService.SetRecordRequestAsync(
    new CatalogueItemTableRecord.CatalogueItemKey() { ItemId = 1 },
    new CatalogueItemTableRecord.CatalogueItemValue()
    {
        CatalogueId = 0,
        Exists = true,
        Price = Web3.Convert.ToWei(5)
    });

var catalogueRecord = await catalogueItemTableService.GetTableRecordAsync(
    new CatalogueItemTableRecord.CatalogueItemKey { ItemId = 1 });

//get data from logs
var recordsFromLog = await catalogueItemTableService.GetRecordsFromLogsAsync(fromBlockNumber: 100,
toBlockNumber: 500);
```

Table Repository and Log Processing

- A table repository is a common abstraction to store all records from the MUD store
- In combination with the StoreEventsLogProcessingService it can store all the record changes in the smart contracts store
- The table service can be used to retrieve all the records for that table from the repository

```
var storeEventsLogProcessingService = new StoreEventsLogProcessingService(web3, namespaceAddress);

var tableRepository = new InMemoryTableRepository();

await storeEventsLogProcessingService.ProcessAllStoreChangesAsync(
   tableRepository, fromBlockNumber: 100, toBlockNumber: 500);

var catalogueItems = await catalogueItemTableService.GetRecordsFromRepository(tableRepository);


```

Table Repository and Log Processing (Database)

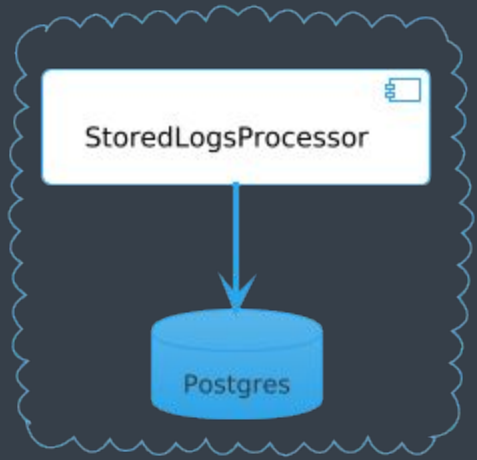


Table Repository and Log Processing (Database)

- Nuget packages: **Nethereum.Mud.Repositories.EF and Nethereum.Mud.Repositories.Postgres**
- It includes a log indexer using the Nethereum Log Processor a Mud specific EF context (generic EF) and a specialised Postgres one to store records
- All the data is stored onto the *storedrecords* table (similar to MUD out of the box indexer)

```
info: MudPostgresStoreRecordsProcessingService[0]
      Total Logs found: 7336
info: MudPostgresStoreRecordsProcessingService[0]
      Logs processed completed to block number: 19934
info: MudPostgresStoreRecordsProcessingService[0]
      Starting logs processing from block number: 19935
info: MudPostgresStoreRecordsProcessingService[0]
      Total Logs found: 7323
info: MudPostgresStoreRecordsProcessingService[0]
      Logs processed completed to block number: 20234
info: MudPostgresStoreRecordsProcessingService[0]
      Starting logs processing from block number: 20235
info: MudPostgresStoreRecordsProcessingService[0]
      Total Logs found: 7350
info: MudPostgresStoreRecordsProcessingService[0]
      Logs processed completed to block number: 20534
info: MudPostgresStoreRecordsProcessingService[0]
      Starting logs processing from block number: 20535
info: MudPostgresStoreRecordsProcessingService[0]
      Total Logs found: 5595
```

public.storedrecords/MUDStore/postgres@PostgreSQL

No limit

Query | Query History | Scratch

```
1 SELECT * FROM public.storedrecords
2 ORDER BY tableid ASC, key ASC, address ASC
```

Data Output | Messages | Notifications

	tableid [PK] bytea	key [PK] bytea	address [PK] bytea	rowid integer	key0 bytea	key1 bytea	key2 bytea	key3 bytea	blocknumber numeric (1000)	logindex integer	isdeleted boolean	static_data bytea	encoded_lengths bytea	dynamic_data bytea
1	[binary data]	[binary data]	[binary data]	17448	[binary dat...	[null]	[null]	[null]	22671	93	false	[binary data]	[binary data]	[binary data]
2	[binary data]	[binary data]	[binary data]	17596	[binary dat...	[null]	[null]	[null]	22717	97	false	[binary data]	[binary data]	[binary data]
3	[binary data]	[binary data]	[binary data]	17628	[binary dat...	[null]	[null]	[null]	22717	143	false	[binary data]	[binary data]	[binary data]
4	[binary data]	[binary data]	[binary data]	17770	[binary dat...	[null]	[null]	[null]	22717	334	false	[binary data]	[binary data]	[binary data]
5	[binary data]	[binary data]	[binary data]	17694	[binary dat...	[null]	[null]	[null]	22717	234	false	[binary data]	[binary data]	[binary data]
6	[binary data]	[binary data]	[binary data]	17444	[binary dat...	[null]	[null]	[null]	22671	87	false	[binary data]	[binary data]	[binary data]
7	[binary data]	[binary data]	[binary data]	17764	[binary dat...	[null]	[null]	[null]	22717	325	false	[binary data]	[binary data]	[binary data]
8	[binary data]	[binary data]	[binary data]	17744	[binary dat...	[null]	[null]	[null]	22717	299	false	[binary data]	[binary data]	[binary data]
9	[binary data]	[binary data]	[binary data]	17748	[binary dat...	[null]	[null]	[null]	22717	305	false	[binary data]	[binary data]	[binary data]
10	[binary data]	[binary data]	[binary data]	17714	[binary dat...	[null]	[null]	[null]	22717	260	false	[binary data]	[binary data]	[binary data]
11	[binary data]	[binary data]	[binary data]	17734	[binary dat...	[null]	[null]	[null]	22717	286	false	[binary data]	[binary data]	[binary data]
12	[binary data]	[binary data]	[binary data]	17644	[binary dat...	[null]	[null]	[null]	22717	165	false	[binary data]	[binary data]	[binary data]
13	[binary data]	[binary data]	[binary data]	17754	[binary dat...	[null]	[null]	[null]	22717	312	false	[binary data]	[binary data]	[binary data]
14	[binary data]	[binary data]	[binary data]	17538	[binary dat...	[null]	[null]	[null]	22717	12	false	[binary data]	[binary data]	[binary data]
15	[binary data]	[binary data]	[binary data]	17462	[binary dat...	[null]	[null]	[null]	22671	114	false	[binary data]	[binary data]	[binary data]
16	[binary data]	[binary data]	[binary data]	17702	[binary dat...	[null]	[null]	[null]	22717	244	false	[binary data]	[binary data]	[binary data]

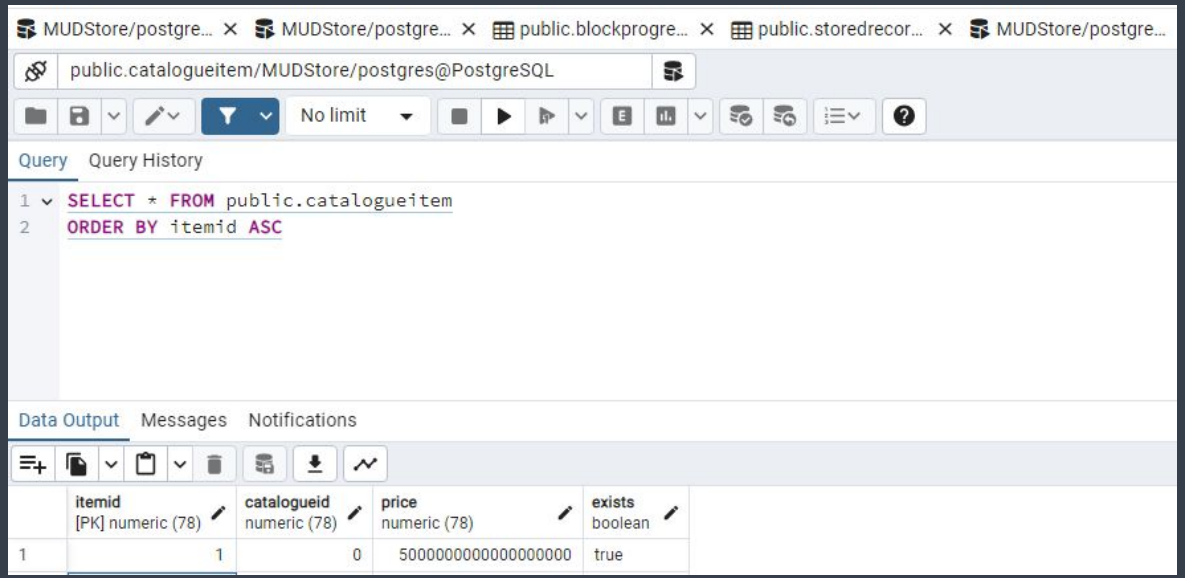
How do I query this encoded data? EF / Postgres repository and Predicate builder

- EF / Postgres repository is the same abstraction to querying data as the InMemory repository.
- A custom predicate builder has been created for each type of storage (EF, Postgres) to query (Equal, NotEqual, And, Or ..) against all the encoded table keys.
- Same predicate builder can be used to send requests through a rest api that wraps the repository to support none direct calls to the database.

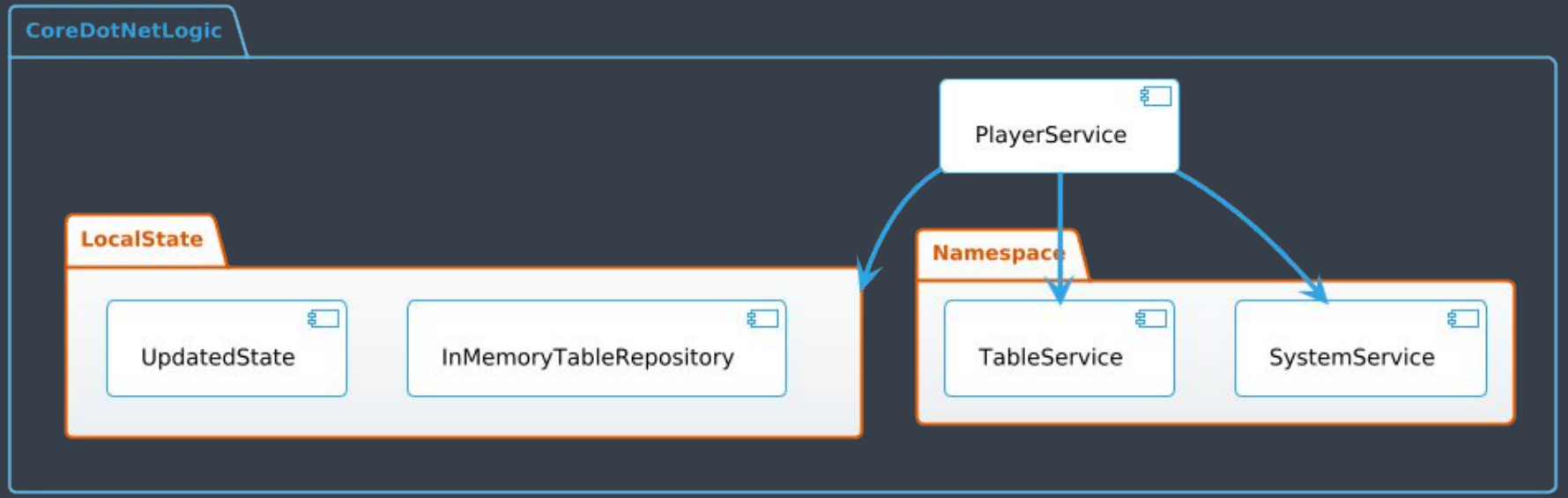
```
1
2  //getting the data from postgres
3  var context = new MudPostgresStoreRecordsDbContext();
4  var postgresRepository = new MudPostgresStoreRecordsTableRepository(context);
5
6  var predicateBuilder = catalogueItemTableService.CreateTablePredicateBuilder();
7  var predicate = predicateBuilder.Equals(x => x.ItemId, 1).OrEqual(x => x.ItemId, 2).Expand();
8
9  var recordsFromPostgres = catalogueItemTableService.GetRecordsFromRepositoryAsync(predicate,
   postgresRepository);
10
11
12  //getting data from the rest api
13 var storedRecordRestApiClient = new StoredRecordRestApiClient(
14             new RestHttpHelper(new HttpClient()), "https://localhost:7034");
15
16 var recordsFromRestApi = await catalogueItemTableService.GetRecordsFromRepositoryAsync(predicate,
   storedRecordRestApiClient);
17
18
```

Table Repository and Log Processing (Database)

- I want to see all the data, create reports, create custom apis, complex queries, etc..
- Normaliser / Normalizer process creates tables based on the schema and decodes the data of the records
- Once it is running and tables created, custom indexes, relationships, and optimisation can be added for performance



SystemService, Namespaces



SystemService (Or contract service) Code generated

- This is the same contract service that is created for standard Nethereum integration with smart contracts, it includes the deployment and simple access to functions, events and errors
- The contract service is a partial class so it can be extended with extra functionality.

```
1    public partial class CatalogueSystemService: ContractWeb3ServiceBase
2    {
3
4        public static Task<string> DeployContractAsync(
5                    Nethereum.Web3.IWeb3 web3,
6                    CatalogueSystemDeployment catalogueSystemDeployment)
7        {
8            return web3.Eth
9                        .GetContractDeploymentHandler<CatalogueSystemDeployment>()
10                       .SendRequestAsync(catalogueSystemDeployment);
11       }
12
13
14       public Task<string> PurchaseCatalogueItemRequestAsync(
15                   BigInteger landId,
16                   BigInteger itemId,
17                   BigInteger quantity)
18       {
19           var purchaseCatalogueItemFunction = new PurchaseCatalogueItemFunction();
20           purchaseCatalogueItemFunction.LandId = landId;
21           purchaseCatalogueItemFunction.ItemId = itemId;
22           purchaseCatalogueItemFunction.Quantity = quantity;
23
24           return ContractHandler.SendRequestAsync(purchaseCatalogueItemFunction);
25       }
26   }
27
28
```

SystemService (Or contract service) Code generated Mud extended

- Mud needs extra functionality and information to identify the smart contract (system)
- This new code generated class it includes the system resource and register all the function signatures.
- It includes also deployment helpers to Create2 and other helper functions to get the functions are required to be registered.

```
1
2   //System resource
3   public class CatalogueSystemServiceResource : SystemResource
4   {
5       public CatalogueSystemServiceResource() : base("Catalogue") { }
6   }
7
8   //Extended partial class for MUD methods
9   public partial class CatalogueSystemService: ISystemService<CatalogueSystemServiceResource>
10  {
11      public IResource Resource => this.GetResource();
12
13      public ISystemServiceResourceRegistration SystemServiceResourceRegistrator
14      {
15          get
16          {
17              return this.GetSystemServiceResourceRegistration
18                                  <CatalogueSystemServiceResource, CatalogueSystemService>();
19          }
20      }
21
22
23      public List<FunctionABI> GetSystemFunctionABIs()
24      {
25          return GetAllFunctionABIs();
26      }
27
28      public string CalculateCreate2Address(string deployerAddress,
29                                            string salt,
30                                            params ByteCodeLibrary[] byteCodeLibraries)
31      {
32          return new CatalogueSystemDeployment()
33                      .CalculateCreate2Address(deployerAddress, salt, byteCodeLibraries);
34      }
35
36      ...
37  }
38
```

Namespace, table service and system services

- All tables and services, belong to a namespace class (in this scenario the land namespace) that matches each namespace in MUD
- The namespace has functionality expands across all these resources, like deployment of all systems, registration of all tables, finding typed errors across all systems, or registration of delegate authority.

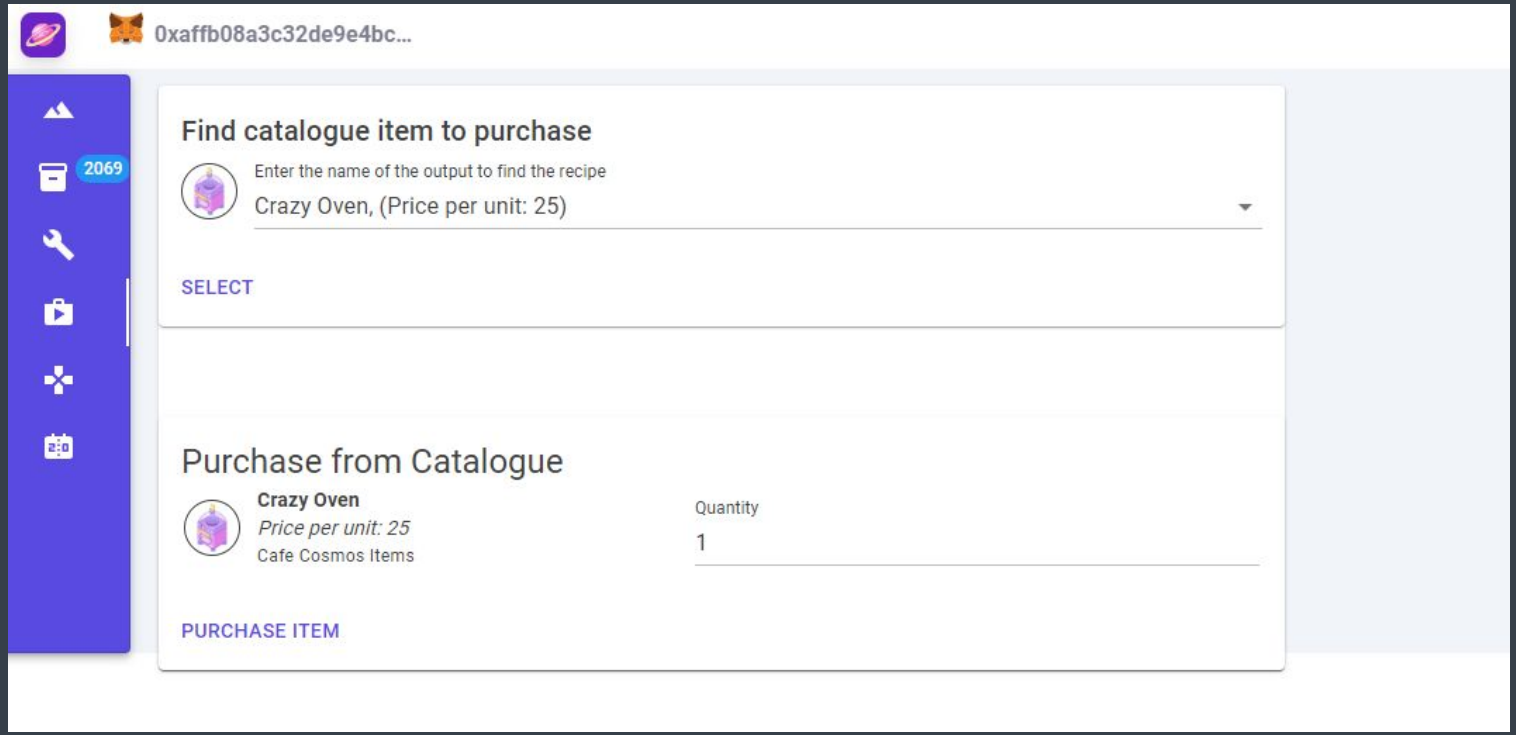
```
public class LandNamespaceResource: NamespaceResource
{
    public LandNamespaceResource() : base("") { }
}

public class LandNamespace : NamespaceBase<LandNamespaceResource, LandSystems,
LandTablesServices>
    public LandNamespace(IWeb3 web3, string contractAddress) : base(web3, contractAddress)
    {
        Systems = new LandSystems(web3, contractAddress);
        Tables = new LandTablesServices(web3, contractAddress);

    }
}

public class LandTablesServices : TablesServices
{
    public CatalogueItemTableService CatalogueItems { get; private set; }
    public LandTablesServices(IWeb3 web3, string contractAddress) : base(web3, contractAddress)
    {
        CatalogueItems = new CatalogueItemTableService(web3, contractAddress);
        TableServices = new List<ITableServiceBase>
        {
            CatalogueItems,
        };
    }
}

public class LandSystems: SystemsServices
{
    public CatalogueSystemService Catalogue { get; private set; }

    public LandSystems(IWeb3 web3, string contractAddress) : base(web3, contractAddress)
    {
        Catalogue = new CatalogueSystemService(web3, contractAddress);

        SystemServices = new List<ISystemService>
        {
           Catalogue
        };
    }
}
```

Player Service - Purchasing

- Core communication logic is through player service (you could have others)
- The player service creates the “Land” Namespace to interact directly with the systems, and deal with other complexities
- When the method is called the data is synchronised with the local state.. But lets do the purchase first and get that oven..

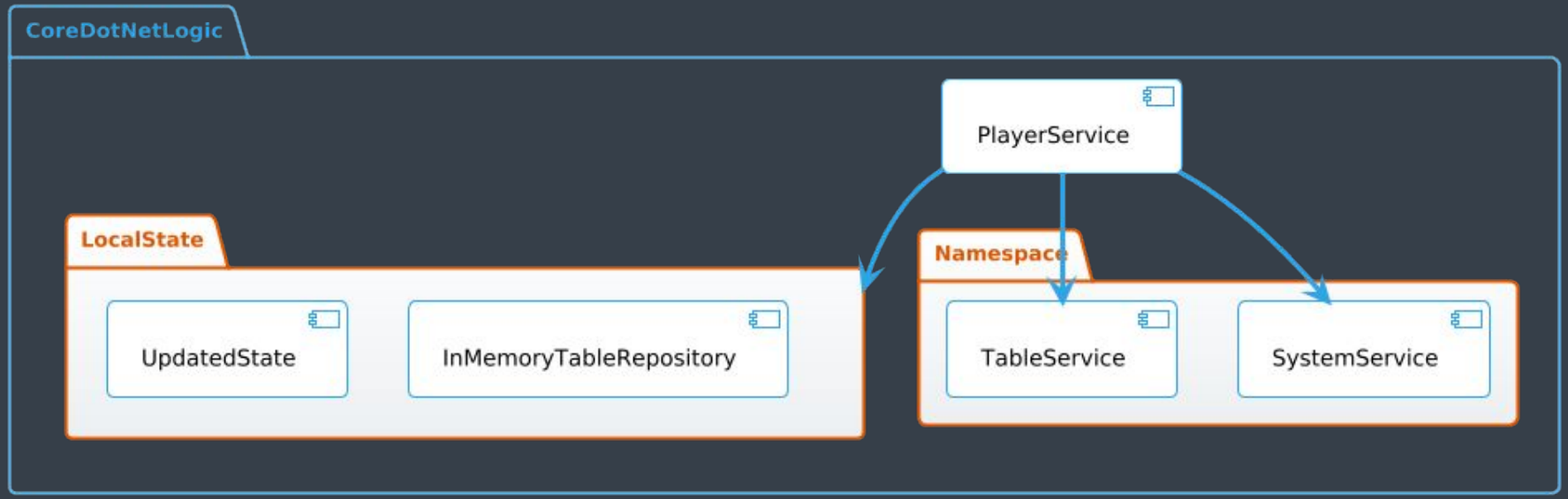
```
1   public class PlayerService
2   {
3       public LandNamespace LandNamespace { get; private set; }
4
5       public PlayerService(Nethereum.Web3.IWeb3 web3,
6                            VisionsContractAddresses contractAddresses,
7                            string playerAddress)
8       {
9           LandNamespace = new LandNamespace(web3, contractAddresses.LandAddress);
10      }
11
12
13      public async Task<TransactionReceipt> PurchaseItemFromCatalogueAndWaitForReceiptAsync(
14                      BigInteger itemId, BigInteger quantity, PlayerLocalState playerLocalState)
15      {
16          var receipt = await LandNamespace
17            .Systems
18            .Catalogue
19            .PurchaseCatalogueItemAndValidateBalanceRequestAndWaitForReceiptAsync(SelectedLandId,
    itemId, quantity, playerLocalState.PlayerLandInfo.TokenBalance);
20
21           await ProcessAllChangesFromReceiptAsync(playerLocalState, receipt);
22          r
23            eturn receipt;
24      }
25
26
```

Catalogue Front End (wasm / Blazor)




0xAfF...b0494
920.135701620...
00:07:11
1
2
3
4
5
6
2
7
8

Local state


CoreDotNetLogic
PlayerService
LocalState
UpdatedState
InMemoryTableRepository
Namespace
TableService
SystemService

Local State

- The local state contains the original state of the game in the *InMemoryTableRepository*
- This is created first when the user starts to play with their state of the game (Land, Inventory, etc)
- All the changes during play made before saving onto the chain are stored in model objects, enabling to continue the game without saving automatically.
- Everytime an action is made this is added on the UpdateLandOperations, which will be later on submitted in batch.

```
1 public class PlayerLocalState
2 {
3     public InMemoryTableRepository InMemoryTableRepository { get; private set; }
4
5     public List<ISystemCallMulticallInput> UpdateLandOperations { get; private set; }
6
7     public List<InventoryItem> UpdatedInventoryItems { get; private set; }
8
9     public List<LandItem> UpdatedLand { get; private set; }
10
11 }
12
13
14
```

What happens when I craft something, or anything on the land?

Local State Game Logic

- When the “craft” action is performed on the game, this action is performed against the local state, and validated as it contains similar logic than the smart contracts.
- A *SystemCallMulticallInput* is created per action, this is a Mud batch item that contains the System Resource Id and the Nethereum Function Message to execute.

```
public class PlayerLocalState
{
    ....

    public void CraftItem(CraftingRecipe craftingRecipe)
    {
       if (DefaultCraftingRecipes.IsValidRecipe(craftingRecipe.Recipe))
       {
           if (craftingRecipe.HasGotInventoryItemsToCraftRecipe(this.UpdatedInventoryItems))
           {

               var inventoryOutput = UpdatedInventoryItems.FirstOrDefault(
                                       x => x.ItemId == craftingRecipe.Output.Id);
               if (inventoryOutput != null)
               {
                   inventoryOutput.Count = inventoryOutput.Count + 1;
               }
               else
               {
                   UpdatedInventoryItems.Add(craftingRecipe.Output.Id, 1);
               }

               var craftRecipeFunction =
craftingRecipe.ConvertToLandCraftRecipeFunction(LandId);
               UpdateLandOperations.Add(
                   new SystemCallMulticallInput<
                         Land.Systems.CraftingSystem.ContractDefinition.CraftRecipeFunction,
                          CraftingSystemServiceResource>
                 (craftRecipeFunction, null));
           }
           else
           {
               throw new CraftRecipeNoEnoughItemsInInventoryException();
           }
       }
       else
       {
           throw new CraftRecipeNotFoundException();
       }
    }

}
```

Saving..

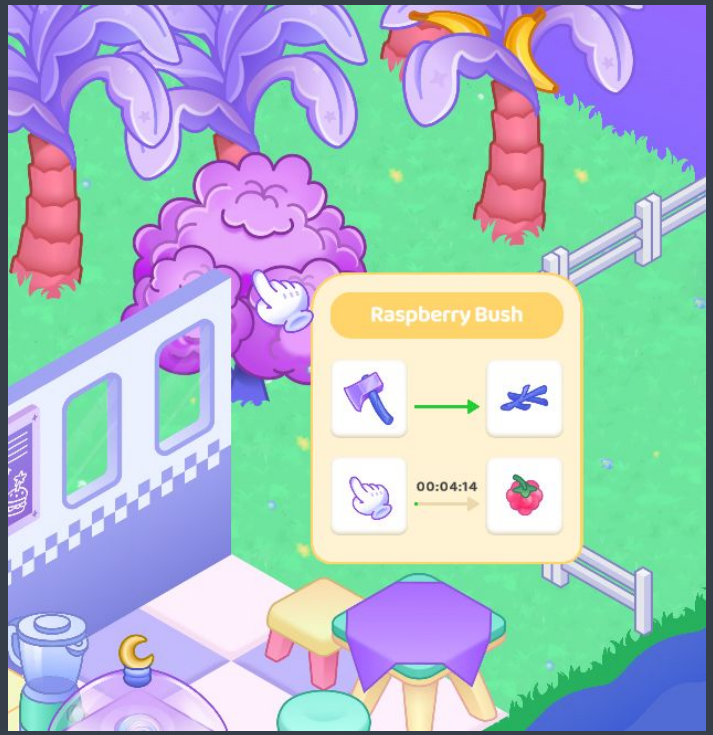
Saving / Executing all player actions on chain

- The game periodically or on user demand will “save the player state”, this executes all the actions that we have collected whilst playing the game and re-execute them in order.
- It is “key” that the transactions don’t fail, so we ensure that the logic match.
- Once it is saved the logs from the transaction are synced with the **InMemoryTableRepository** and recreated again the object model for the user, with the chain specific changes on the state

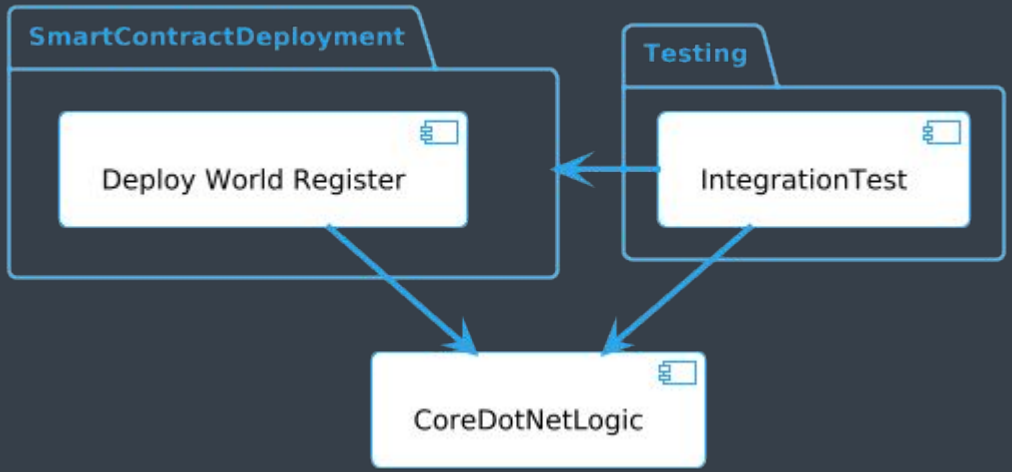
```
public class PlayerService
{

public LandNamespace LandNamespace { get; private set; }

public PlayerService(Nethereum.Web3.IWeb3 web3,
                     VisionsContractAddresses contractAddresses, string playerAddress)
{
    LandNamespace = new LandNamespace(web3, contractAddresses.LandAddress);
    PlayerAddress = playerAddress;
}

 public async Task<TransactionReceipt> SavePlayerStateAndWaitForReceiptAsync
                                                    (PlayerLocalState playerLocalState)
 {
    var receipt = await
                  LandNamespace.Systems.LandItemInteraction
                      .UpdateLandRequestAndWaitForReceiptAsync(playerLocalState);
        await ProcessAllChangesFromReceiptAsync(playerLocalState, receipt);

       return receipt;
 }


 private async Task ProcessAllChangesFromReceiptAsync(
                         PlayerLocalState playerLocalState,
                         TransactionReceipt receipt)
 {
        await StoreEventsLogProcessingService.
                     ProcessAllStoreChangesFromLogs(playerLocalState.InMemoryTableRepository,
                                                                                receipt);
           var landItems = await LandNamespace
                               .GetAllLandItemVOsFromLandTableRepositoryAsync(
                                           playerLocalState.InMemoryTableRepository);
        var inventoryItems = await LandNamespace
                               .GetAllInventoryItemsFromTableRepositoryAsync(
                                           playerLocalState.InMemoryTableRepository);

        playerLocalState.Initialise(inventoryItems, landItems);
    }
}
```

What changes can come from the chain?

- Logic that prevents from cheating, creating bots, etc.
- Placement time
- XP points changes
- Level progress
- Quests completed, rewards



Deployment and Testing



Deployment

- MUD deployment with Nethereum is the same as standard MUD, first the World Factory is deployed and then the World.
- Once we have a world address, we will register the Namespace and Tables
- Finally we can register all the systems and function selectors automatically in a Batch call
- As your world evolves, you can register or update systems individually, or register new tables.

```
logger.LogInformation("Deploying World Factory...");
var worldFactoryDeployerService = new WorldFactoryDeployService();
var worldFactoryAddresses = await worldFactoryDeployerService.
                    DeployWorldFactoryContractAndSystemDependenciesAsync
                                        (web3, create2DeployerAddress, salt);

logger.LogInformation("Deploying World...");
var worldEvent = await worldFactoryDeployerService.DeployWorldAsync
                                        (web3, salt, worldFactoryAddresses);
var worldAddress = worldEvent.NewContract;
var landNamespace = new LandNamespace(web3, worldAddress);

logger.LogInformation("Registering land namespace...");
await landNamespace.RegisterNamespaceRequestAndWaitForReceiptAsync();
logger.LogInformation("Registering land tables...");

await
landNamespace.Tables.RegisterAllTablesRequestAndWaitForLastTxnReceiptAsync();
logger.LogInformation("Deploying Land Systems...");
await landNamespace.Systems.DeployAllCreate2ContractSystemsRequestAsync
                                        (create2DeployerAddress, salt, null);

logger.LogInformation("Registering Land Systems...");

await landNamespace.Systems.BatchRegisterAllSystemsRequestAndWaitForReceiptAsync
                                        (create2DeployerAddress, salt);
```

Testing

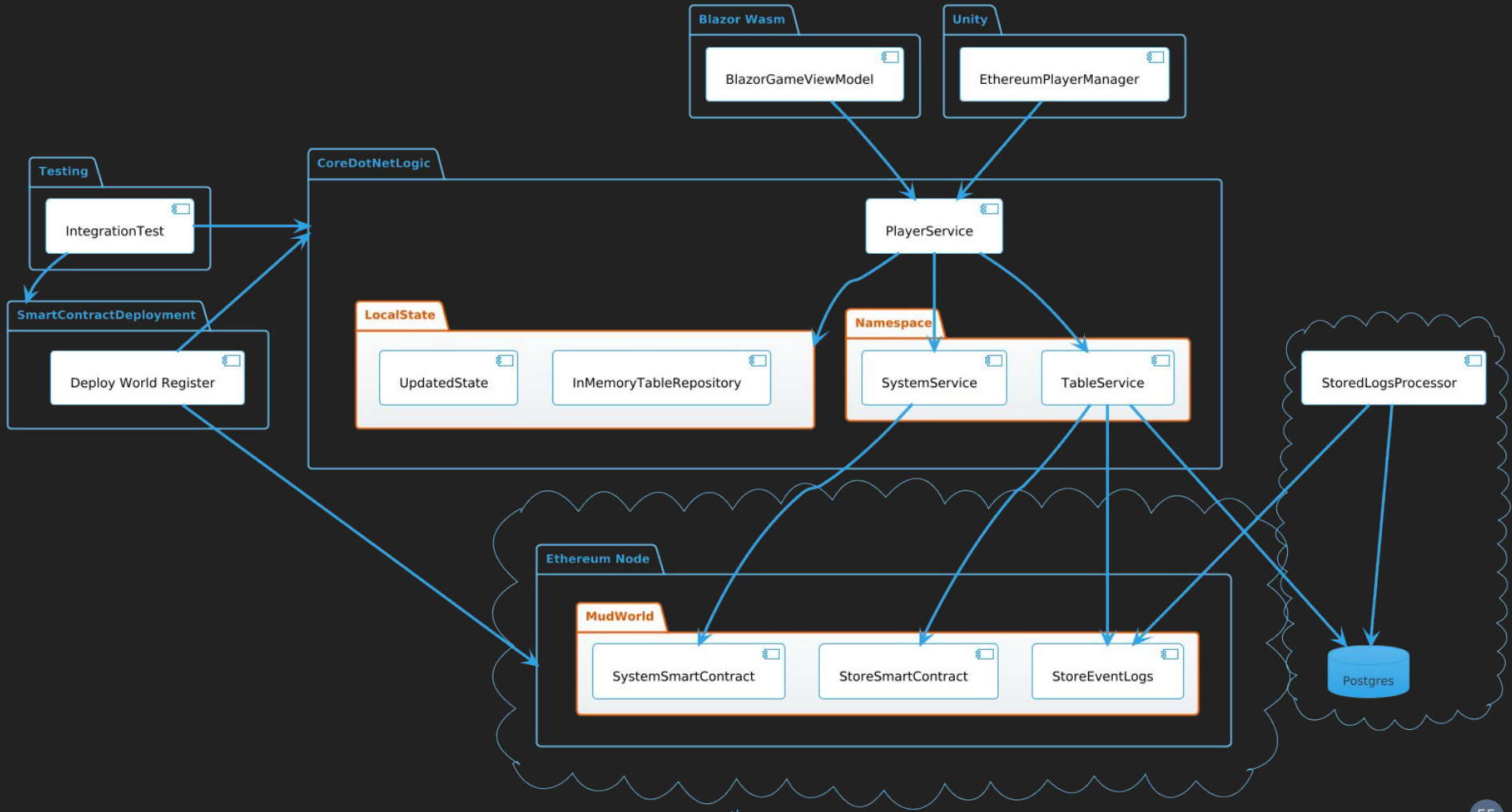
- Nethereum includes a testing library in all of its templates and examples, which allow you to launch a geth / hardhat test chain (and clear them) automatically or just connect it to an existing chain like Anvil that is running in the background.
- Extension methods for Hardhat or Anvil are supported using the package Nethereum.RPC.Extensions .
- In this test all the contracts are deployed and specific data scenario is set for easy testing. This test uses the player service directly and local state for a full e2e integration test.

```
1     [Collection(EthereumClientIntegrationFixture.ETHEREUM_CLIENT_COLLECTION_DEFAULT)]
2     public class XP_Tests
3     {
4         private readonly EthereumClientIntegrationFixture _ethereumClientIntegrationFixture;
5         private string PlayerAddress { get; set; } = TestAccounts.Account1Address;
6         private TestContractsDeployerServices _testContractsDeployerServices;
7
8         public XP_Tests(EthereumClientIntegrationFixture ethereumClientIntegrationFixture)
9         {
10            _ethereumClientIntegrationFixture = ethereumClientIntegrationFixture;
11            _testContractsDeployerServices = new
12                 TestContractsDeployerServices(ethereumClientIntegrationFixture);
13        }
14
15        [Fact]
16        public async Task ShouldIncreaseXpWhilstCrafting()
17        {
18            var landTestAndContractAddresses = await _testContractsDeployerServices
19                                                              .DeployContractsAsync(
20
21 PlayerLocalStateTestScenarioFactory.TestScenario.AllFloorWithAllInventoryItems);
22            var contractAddresses = landTestAndContractAddresses.ContractAddresses;
23            var landId = landTestAndContractAddresses.LandId;
24
25            var web3 = TestAccounts.GetAccount1Web3();
26
27            var playerService = new PlayerService(web3, contractAddresses, PlayerAddress);
28            playerService.SelectedLandId = (int)landId;
29
30            var localState = await playerService.GetNewPlayerLocalStateAsync();
31            Assert.Equal(0, localState.PlayerLandInfo.CumulativeXp);
32
33            localState.CraftItem(DefaultCraftingRecipes.Cooking.BLENDER);
34
35            await playerService.SavePlayerStateAndWaitForReceiptAsync(localState);
36            Assert.True(localState.PlayerLandInfo.CumulativeXp > 0);
37        }
38    }
39
```

Delegated authority

- To constantly save we need temporary accounts, we have called them many things over the years, device accounts, session accounts or burner accounts.
- To register a delegatee account, use the namespace world systems.
- Afterwards you will need to set the delegator in your namespace, this configures every system / service to use a new **MudCallFromContractHandler** that wraps every transaction with **CallFromFunction**

```
1  public Task<TransactionReceipt> SetDelegateeRequestAndWaitForReceiptAsync(string delegatee)
2  {
3      return LandNamespace
4                  .World
5                  .Systems
6                  .RegistrationSystem
7                  .RegisterDelegationRequestAndWaitForReceiptAsync(delegatee,
                               ResourceEncoder.EncodeUnlimitedAccess(), new byte[] { });
8  }
9
10 public void SetDelegator(string delegator)
11 {
12       LandNamespace
13         .Systems
14         .SetSystemsCallFromDelegatorContractHandler(delegator);
15 }
```


Blazor Wasm
BlazorGameViewModel
Unity
EthereumPlayerManager
Testing
IntegrationTest
CoreDotNetLogic
PlayerService
SmartContractDeployment
Deploy World Register
LocalState
UpdatedState
InMemoryTableRepository
Namespace
SystemService
TableService
StoredLogsProcessor
Ethereum Node
MudWorld
SystemSmartContract
StoreSmartContract
StoreEventLogs
Postgres
nethereum

Q&A and Links

- **Mud :** https://mud.dev/ (MUD !)
- **Nethereum Github:** https://github.com/Nethereum (The code)
- **Nethereum Nugets:** https://www.nuget.org/profiles/nethereum (Nuget packages)
- **Nethereum Discord:** https://discord.gg/u3Ej2BReNn (Come and and ask questions and chat)
- **Netherum.Unity**: https://github.com/Nethereum/Nethereum.Unity (Unity package)
- **Unity starter template:** https://github.com/Nethereum/Unity3dSampleTemplate
- **Nethereum playground:** http://playground.nethereum.com
- **Vscode Solidity:** https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity
- **Cafe Cosmos:** https://www.cafecosmos.io/ (Have a play)