

In the name of Allah

Computer Stuff (NOSQL & Blockchain)

They didn't teach you

Contents

- NoSQL
- Polyglot persistence architecture
- Microservices
- Motivations of NoSQL
- CAP Theorem
- Document Based NoSQL
- ACID vs BASE
- Mongo CRUD operations
- .Net Based Services for Mongo
- Blockchain
- Genesis block
- Bdaya.Net.Blockchain the UnitOfWork design pattern

NoSQL



NoSQL

A NoSQL (originally referring to "non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

- Such databases have existed since the late 1960s, but the name "NoSQL" was only coined in the early 21st century, triggered by the needs of Web 2.0 companies.
- NoSQL databases are increasingly used in big data and real-time web applications.

- NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages or sit alongside SQL databases in polyglot-persistent architectures.

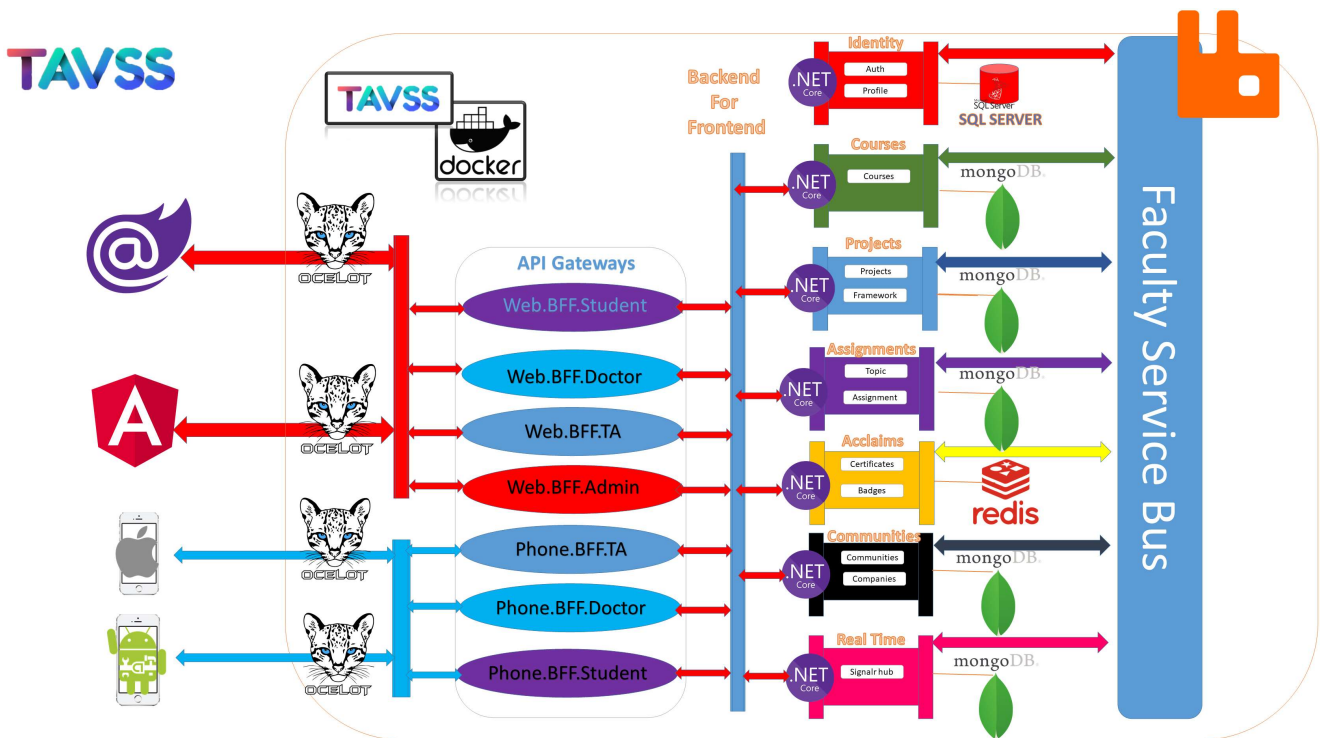
[Wikipedia \(https://en.wikipedia.org/wiki/NoSQL\)](https://en.wikipedia.org/wiki/NoSQL)

Polyglot Persistence Architecture

is a term that refers to using multiple data storage technologies for varying data storage needs across an application or within smaller components of an application. Such varying data storage needs could arise in both the cases, i.e. an enterprise with multiple applications or singular components of an application needing to store data differently.

Microservices and Polyglot Persistence Architecture

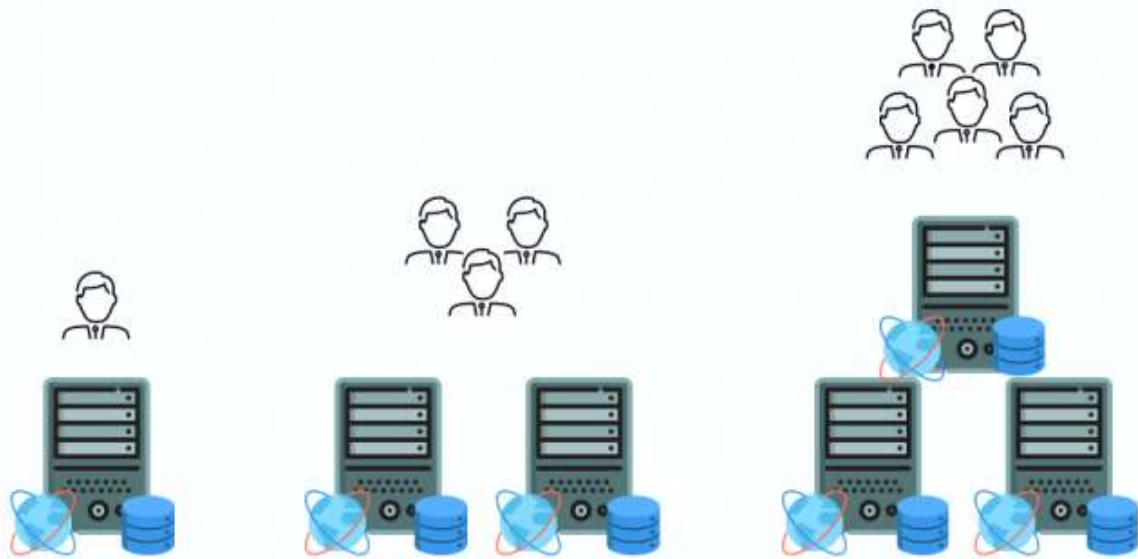
- Like my Graduation project for Microservices Architecture using heterogeneous databases as Polyglot Persistence Architecture with single database per service (SDPS) Design pattern



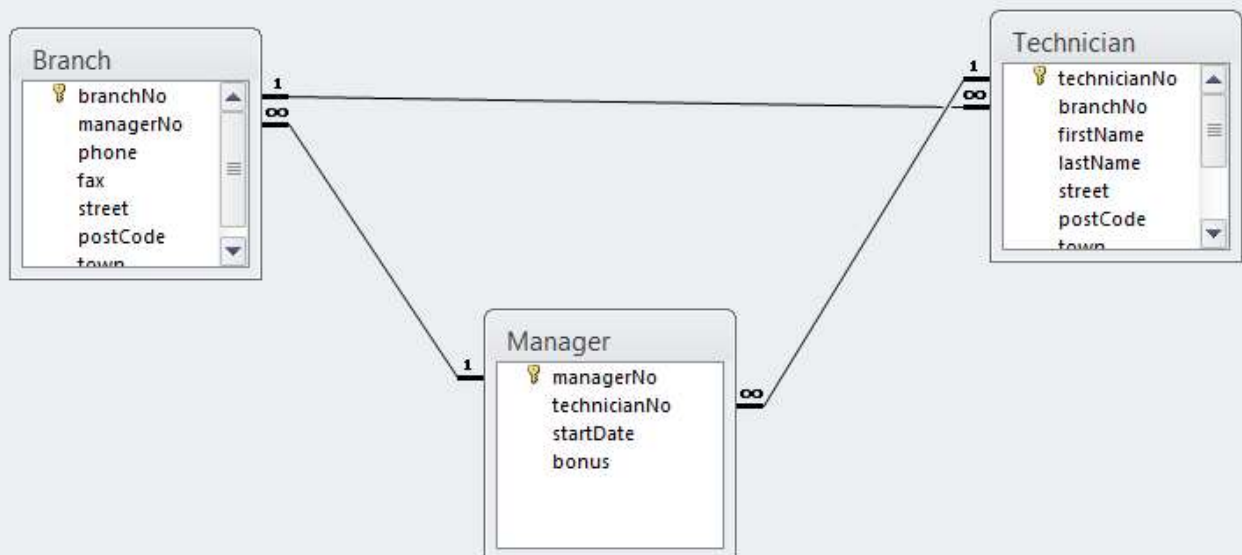
Motivations

- **Motivations** for this approach include:
 - simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases)

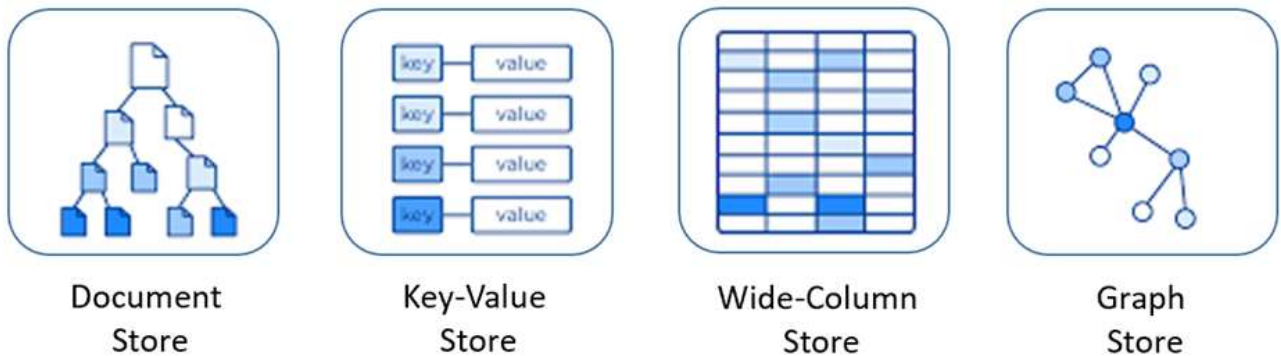
Horizontal Scaling



- finer control over availability and limiting the object-relational impedance mismatch.



- The data structures used by NoSQL databases (e.g. key–value pair, wide column, graph, or document) are different from those used by default in relational databases making some operations faster in NoSQL.



- The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.

Document Store NoSQL databases

Document stores make use of unique documents, usually a type of JSON object with fields and values. {MongoDb}

- These documents support many of the primitive data types we're used to from our programming languages, making it easy to get up and running with your language of choice.
- As opposed to normalizing data and building relations, documents allow you to store all the information related to an entity in that document.

Case Study 1

- Create a database for users every user can have many hobbies and a hobby owned by many users

SQL implementation

Users Table

id	first_name	last_name	email	hashed_password
1	Alex	Shelley	s.alex@mail.com	7Fy5vQhehSBAuJaKBJLC
2	Manuela	Avery	mavery@cool.org	GFUtPYL3vyKZAQqJeVmA

Hobbies Table

id	hobby
1	Swimming
2	Music

users_hobbies

id	user_id	hobby_id
1	1	1
2	1	2
3	2	1

Document store Implementation

```
{
  "_id": 1,
  "name": {
    "first": "Alex",
    "last": "Shelley"
  },
  "email": "s.alex@mail.com",
  "hashed_password": "7Fy5vQhehSBAuJaKBJLC",
  "hobbies": [{ "hobby": "Swimming" }, { "hobby": "Music" }]
}
```

```
{
  "_id": 2,
  "name": {
    "first": "Manuela",
    "last": "Avery"
  },
  "email": "mavery@cool.org",
  "hashed_password": "GFUtPYL3vyKZAQqJeVmA",
  "hobbies": [{ "hobby": "Swimming" }]
}
```

CAP Therom

In theoretical computer science, the CAP theorem, also named Brewer's theorem after computer scientist Eric Brewer, states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees

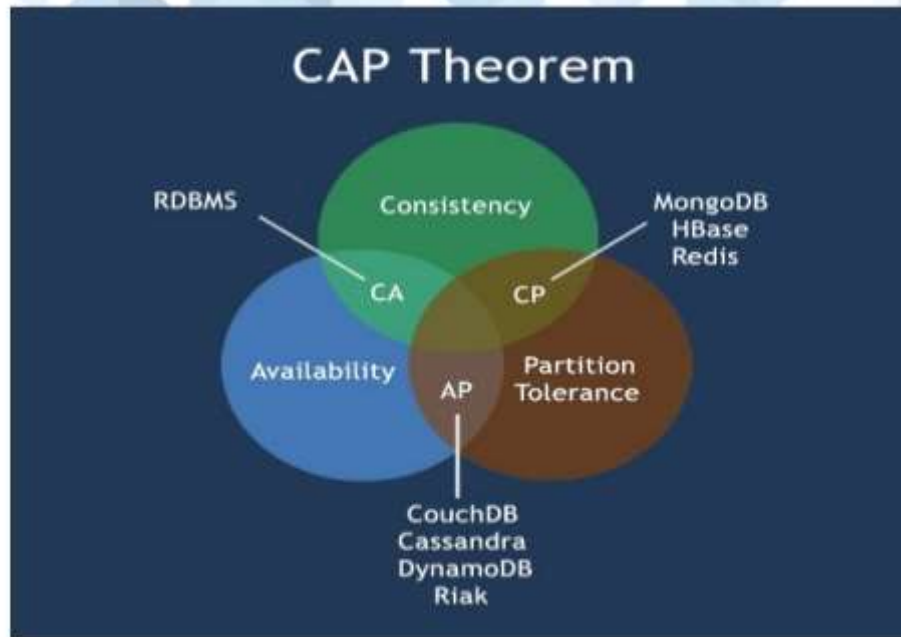
- Consistency: Every read receives the most recent write or an error
- Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
- Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes



Pick

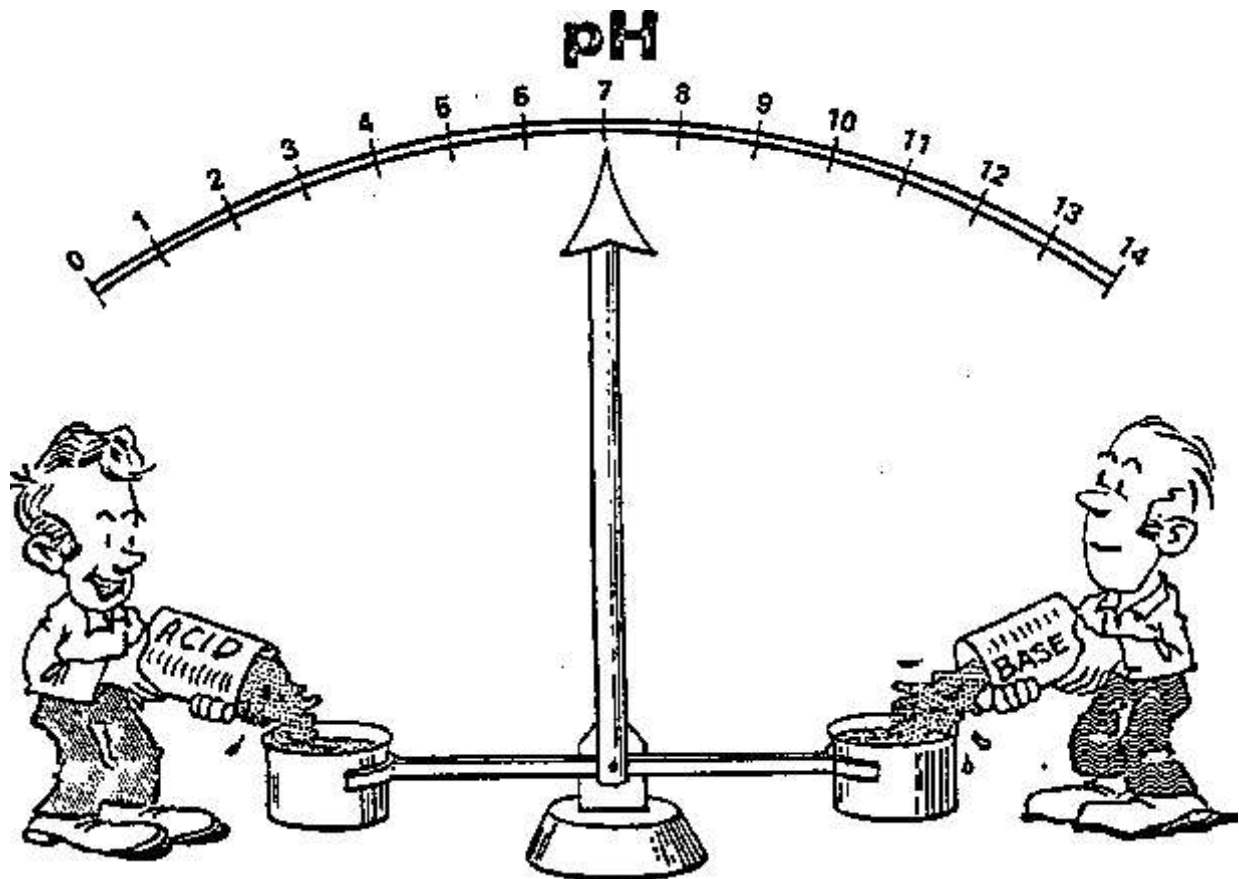
Only

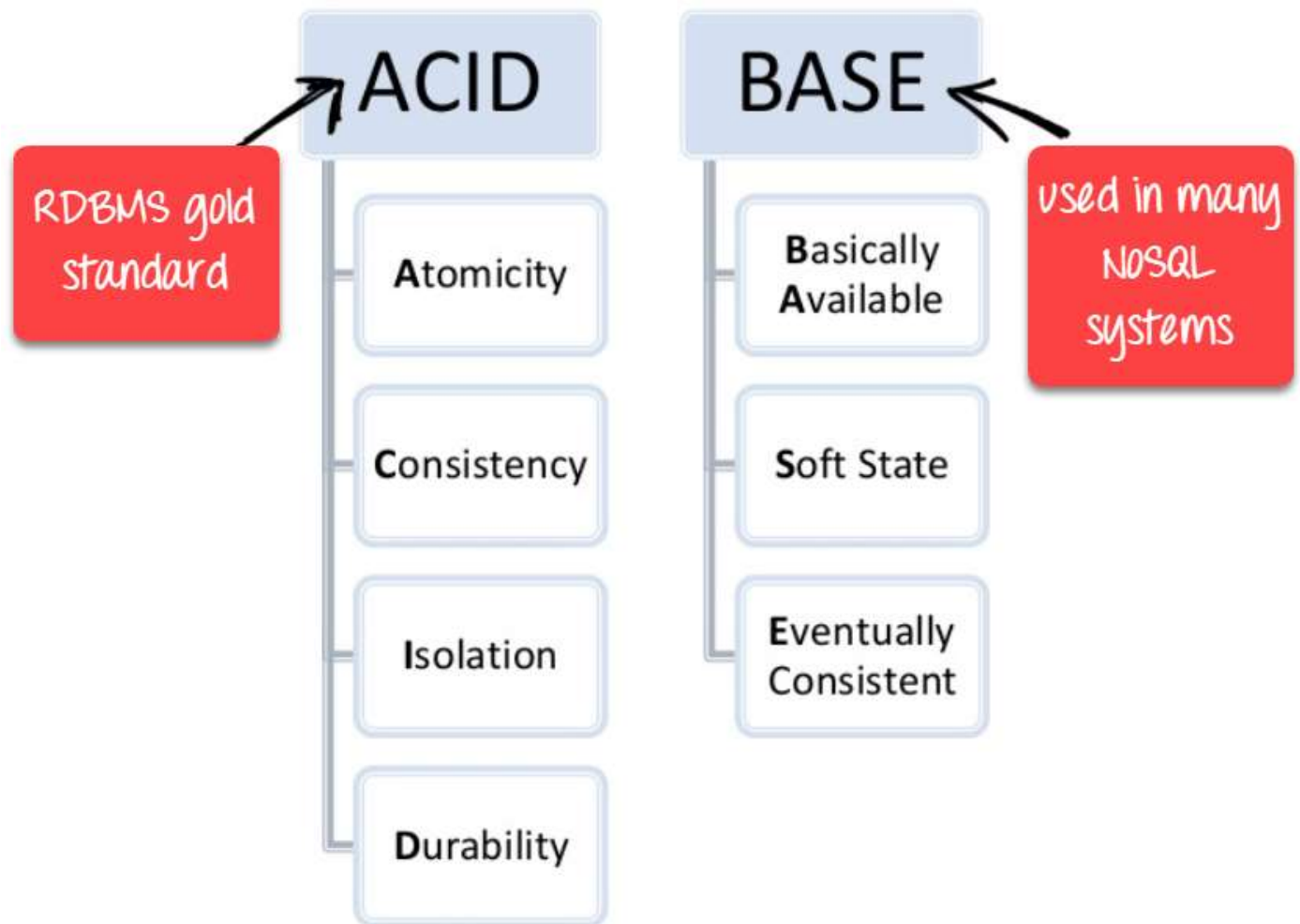
"TWO"



source: <http://www.abramson.com/>

ACID vs BASE



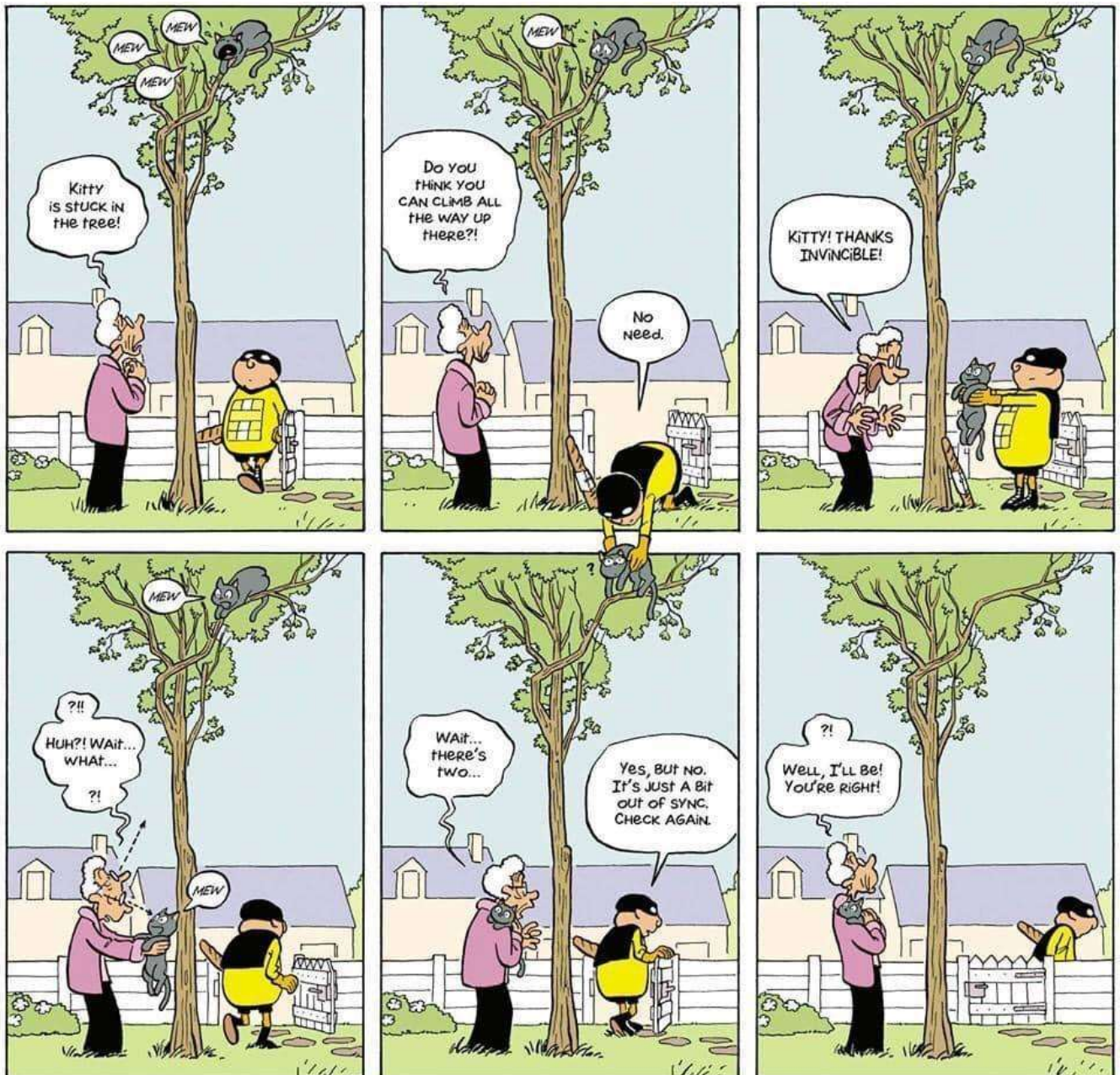


ACID

- Atomic
 - All operations in a transaction succeed or every operation is rolled back.
- Consistent
 - On the completion of a transaction, the database is structurally sound.
- Isolated
 - Transactions do not contend with one another. Contentious access to data is moderated by the database so that transactions appear to run sequentially.
- Durable
 - The results of applying a transaction are permanent, even in the presence of failures.

BASE

- Basic Availability
 - The database appears to work most of the time.
- Soft-state
 - The state of the system could change over time, so even during times without input there may be changes going on due to 'eventual consistency,' thus the state of the system is always 'soft.'
- Eventual consistency
 - Stores exhibit consistency at some later point (e.g., lazily at read time).



Let's Talk To Mongo and Hear its sound

MongoDb CRUD Operations from Base Web Interface

- First we need to create a collection, a collection is like the entity on ERD or Table => SQL Data

```
MongoDB Enterprise atlas-446t8b-shard-0:PRIMARY> db.createCollection("Messages")
{
  "ok" : 1,
  "$clusterTime" : {
    "clusterTime" : Timestamp(1610721781, 1),
    "signature" : {
      "hash" : BinData(0,"UORYq7GaKqTtXfyBLm4jkChuv1g="),
      "keyId" : NumberLong("6916057254017892355")
    }
  },
  "operationTime" : Timestamp(1610721781, 1)
}
```

```
db.createCollection("Messages")
```

- The first CRUD is to create an observation


```
MongoDB Enterprise atlas-446t8b-shard-0:PRIMARY> db.Messages.insert({Issuer: "Ahmed Khalil", Audience: ["Dr Ahmed", "Friend 1", "Friend 2"],Body : "Hello There",Type:"TextMessage"})
WriteResult({ "nInserted" : 1 })
```

```
db.Messages.insert({Issuer: "Ahmed Khalil", Audience: ["Dr Ahmed", "Friend 1", "Friend 2"],Body : "Hello There",Type:"TextMessage"})
```

- When Read we can read a list of a collection by calling `find` function , the `pretty` make the json in good structure

```
MongoDB Enterprise atlas-446t8b-shard-0:PRIMARY> db.Messages.find().pretty()
{
  "_id" : ObjectId("6001abb16cadd3f6bbd3ed6e"),
  "Issuer" : "Ahmed Khalil",
  "Audience" : [
    "Dr Ahmed",
    "Friend 1",
    "Friend 2"
  ],
  "Body" : "Hello There",
  "Type" : "TextMessage"
}
```

```
db.Messages.find().pretty()
```

- insert another message

```
db.Messages.insert({Issuer: "Dr Ahmed", Audience: ["Ahmed Khalil" , "Friend 1", "Friend 2"],Body : "Hello Students" ,Type:"TextMessage"})
```

```
_id: ObjectId("6001abb16cadd3f6bbd3ed6e")
Issuer: "Ahmed Khalil"
Audience: Array
  0: "Dr Ahmed"
  1: "Friend 1"
  2: "Friend 2"
Body: "Hello There"
Type: "TextMessage"
```

```
_id: ObjectId("6001ae6d6cadd3f6bbd3ed6f")
Issuer: "Dr Ahmed"
Audience: Array
  0: "Ahmed Khalil"
  1: "Friend 1"
  2: "Friend 2"
Body: "Hello Students"
Type: "TextMessage"
```

- we can get only one by calling it with id directly

```
MongoDB Enterprise atlas-446t8b-shard-0:PRIMARY> db.Messages.find({"_id" : ObjectId("6001abb16cadd3f6bbd3ed6e")}).pretty()
{
  "_id" : ObjectId("6001abb16cadd3f6bbd3ed6e"),
  "Issuer" : "Ahmed Khalil",
  "Audience" : [
    "Dr Ahmed",
    "Friend 1",
    "Friend 2"
  ],
  "Body" : "Hello There",
  "Type" : "TextMessage"
}
```

```
db.Messages.find({"_id" : ObjectId("6001abb16cadd3f6bbd3ed6e")}).pretty()
```

- either we can limit the queries and other operations of sorting like SQL operation `'limit'`
- we can update the existing document by

```
MongoDB Enterprise atlas-446t8b-shard-0:PRIMARY> db.operation.updateOne({ Issuer: "Ahmed Khalil" },{$set: {Body: "Making some fun"}})
{ "acknowledged" : true, "matchedCount" : 0, "modifiedCount" : 0 }
```

```
db.operation.updateOne({ Issuer: "Ahmed Khalil" },{$set: {Body: "Making some fun"}})
```

- is it safe ?? no a disaster !!!!!!!!!!!!!!!
- delete

```
db.Messages.deleteOne({ name : "Ahmed Khalil" })
```

.Net Mongo Db

We gonna install the assemblies of the mongo drive and the BSON Types

- Simple Talabat System



In [11]:

```
#r "nuget:MongoDB.Driver.Core,2.11.5"  
#r "nuget:MongoDB.Driver,2.11.5"  
#r "nuget:MongoDB.Bson,2.11.5"
```

In [12]:

```
using MongoDB.Bson;
using MongoDB.Driver;
using MongoDB.Bson.Serialization.Attributes;

public enum OrderStatus
{
    Pending,
    Approved,
    Completed,
    Rejected,
    Canceled
}

public class Order
{
    [BsonId]
    public ObjectId Id { get; set; }
    public string Owner { get; set; }
    public OrderStatus Status { get; set; }
    public List<string> Meals { get; set; }

    public string Address { get; set; }
}
```

In [13]:

```
// Intializing connection

var password = "Password123";
var db = "TalabatDb";
var client = new MongoClient($"mongodb+srv://AhmedKhalil777:{password}@db1.lbuwf.mongodb.n
var database = client.GetDatabase(db);
var ordersDTOCollection = database.GetCollection<Order>("CRUDOrders");
```

In [32]:

```
var id = ObjectId.GenerateNewId();
var order = new Order
{
    Address = "Mahalah",
    Id = id ,
    Meals = new List<string> { "Burger", "Fruiet Salad" },
    Status = OrderStatus.Pending,
    Owner = "Ahmed Khalil"
};

// Creating
await ordersDTOCollection.InsertOneAsync(order);
```

In [33]:

```
// Creating a filter
var filter = Builders<Order>.Filter.Eq(x=> x.Id, id);

// Requesting the order
var myOrder = await ordersDTOCollection.Find(filter).SingleOrDefaultAsync();

// displaying
display(myOrder);
```

Id	Owner	Status	Meals	Address
600495e6560c42b4b96f87ea	Ahmed Khalil	Pending	[Burger, Fruiet Salad]	Mahalah

In [34]:

```
// add update definition

var updateDef = Builders<Order>.Update.Set(x=> x.Status, OrderStatus.Approved);

// Updating
// Watching the Eventual consistent
// How we can avoid this
myOrder= await ordersDTOCollection.FindOneAndUpdateAsync(filter,updateDef );

// displaying
display(myOrder);
```

Id	Owner	Status	Meals	Address
600495e6560c42b4b96f87ea	Ahmed Khalil	Pending	[Burger, Fruiet Salad]	Mahalah

In [35]:

```
display(myOrder);
```

Id	Owner	Status	Meals	Address
600495e6560c42b4b96f87ea	Ahmed Khalil	Pending	[Burger, Fruiet Salad]	Mahalah

In [36]:

```
// add update definition

updateDef = Builders<Order>.Update.Set(x=> x.Status, OrderStatus.Completed);

// Updating
// Watching the Eventual consistent
// How we can avoid this
await ordersDTOCollection.UpdateOneAsync(filter,updateDef );

// getting
myOrder = await ordersDTOCollection.Find(filter).SingleOrDefaultAsync();

// displaying
display(myOrder);
```

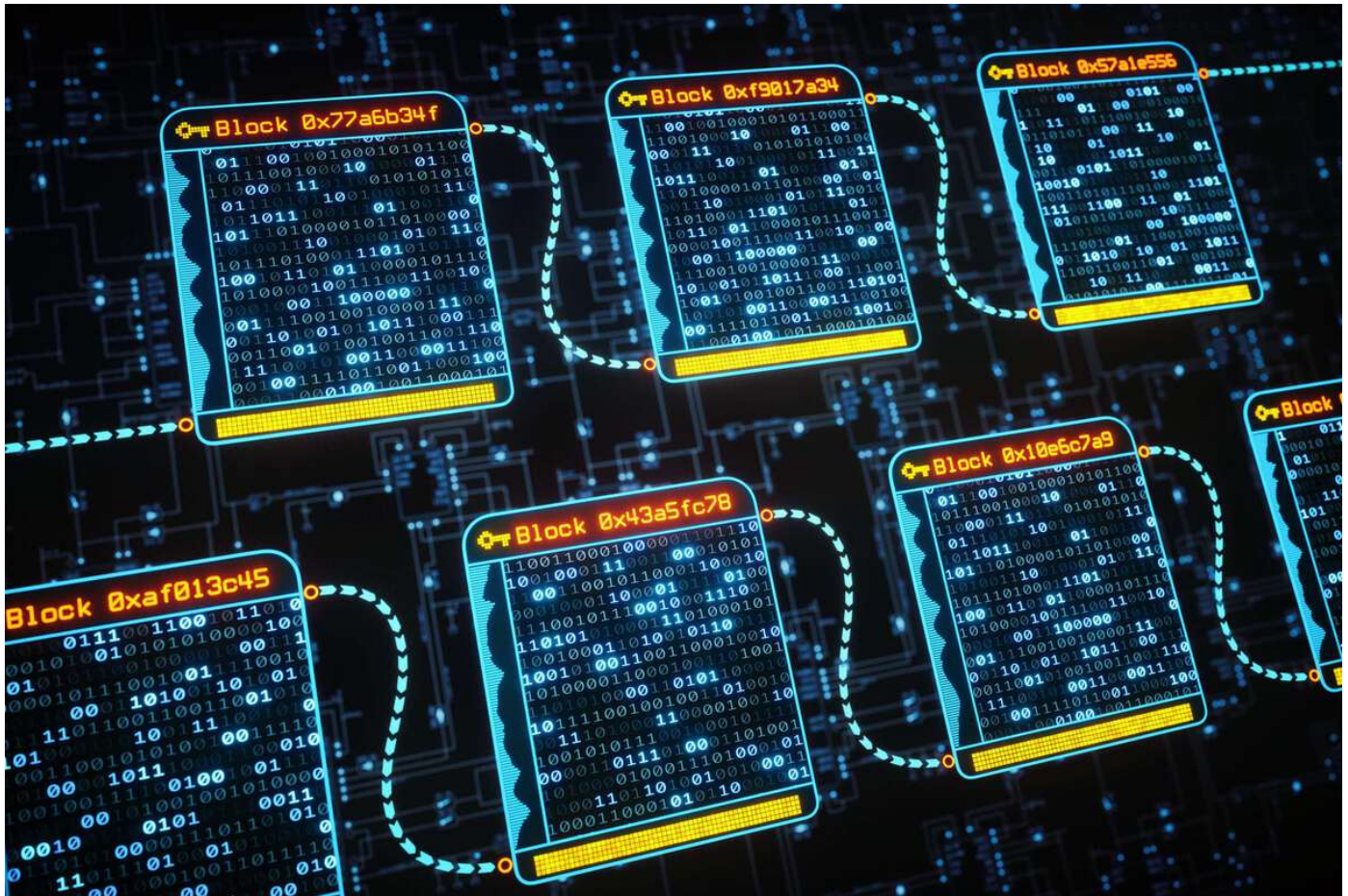
Id	Owner	Status	Meals	Address
600495e6560c42b4b96f87ea	Ahmed Khalil	Completed	[Burger, Fruiet Salad]	Mahalah

In [37]:

```
myOrder = await ordersDTOCollection.FindOneAndDeleteAsync(filter);
```

In []:

Blockchain



Who Is Satoshi Nakamoto?

Satoshi Nakamoto is the name used by the presumed pseudonymous person or persons who developed bitcoin, authored the bitcoin white paper, and created and deployed bitcoin's original reference implementation.

- As part of the implementation, Nakamoto also devised the first blockchain database.
- In the process, Nakamoto was the first to solve the double-spending problem for digital currency using a peer-to-peer network.
- Nakamoto was active in the development of bitcoin up until December 2010. Many people have claimed, or have been claimed, to be Satoshi Nakamoto.
- On or shortly after January 3rd, 2009, the **genesis block** was mined by the elusive Satoshi Nakamoto.
- This would be the beginning of the global technological and economic movement that we now know as blockchain technology.

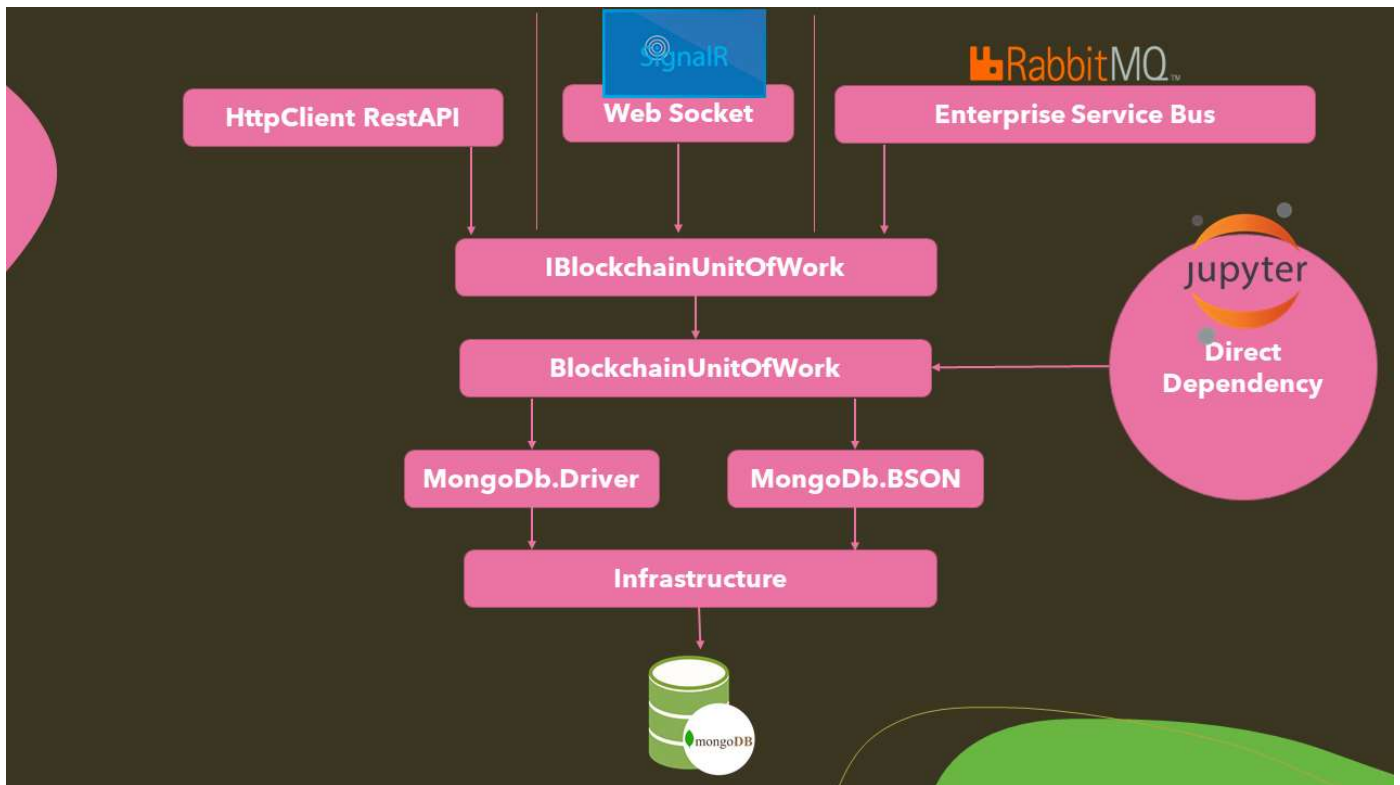


What is a block

A block is like a file filled with transaction data, timestamps, and other data like the hash.

Bdaya.Net.Blockchain Assembly architecture

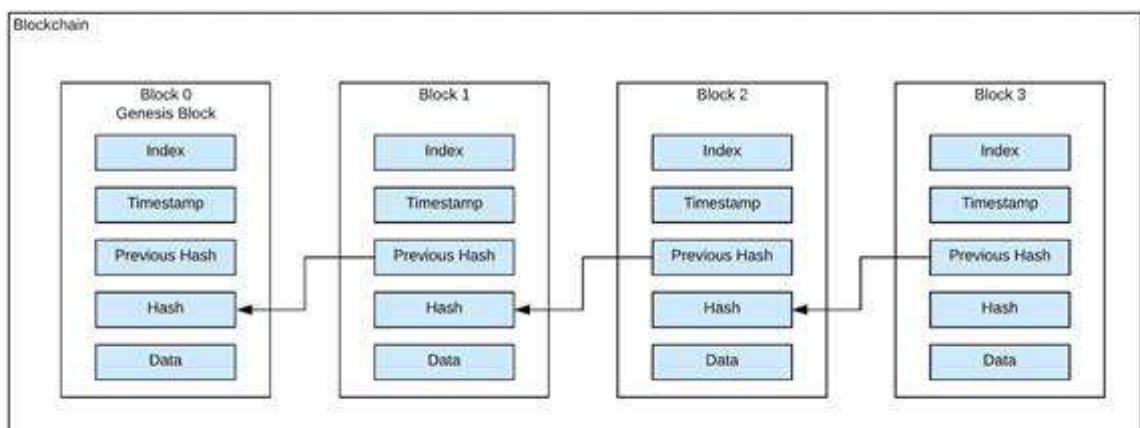
- Abstraction layers and data types that simplify blockchain architecture as a data structure level to focus more in business not implementation
- 3 interfaces messaging
 - Event Driven Based Architecture using rabbitMQ , Publish subscriber Design pattern to serve other architecture { Microservices, SOA, IOT}
 - Websockets for realtime usages {SignalR}
 - Restful api to work within client apps (Mobile Apps, Web Apps , Single page apps)
- also you can use it by calling the direct dependency like what we are going to talk ,, or Dependency Injection Lifecycles {Transient, Scoped, Singleton}

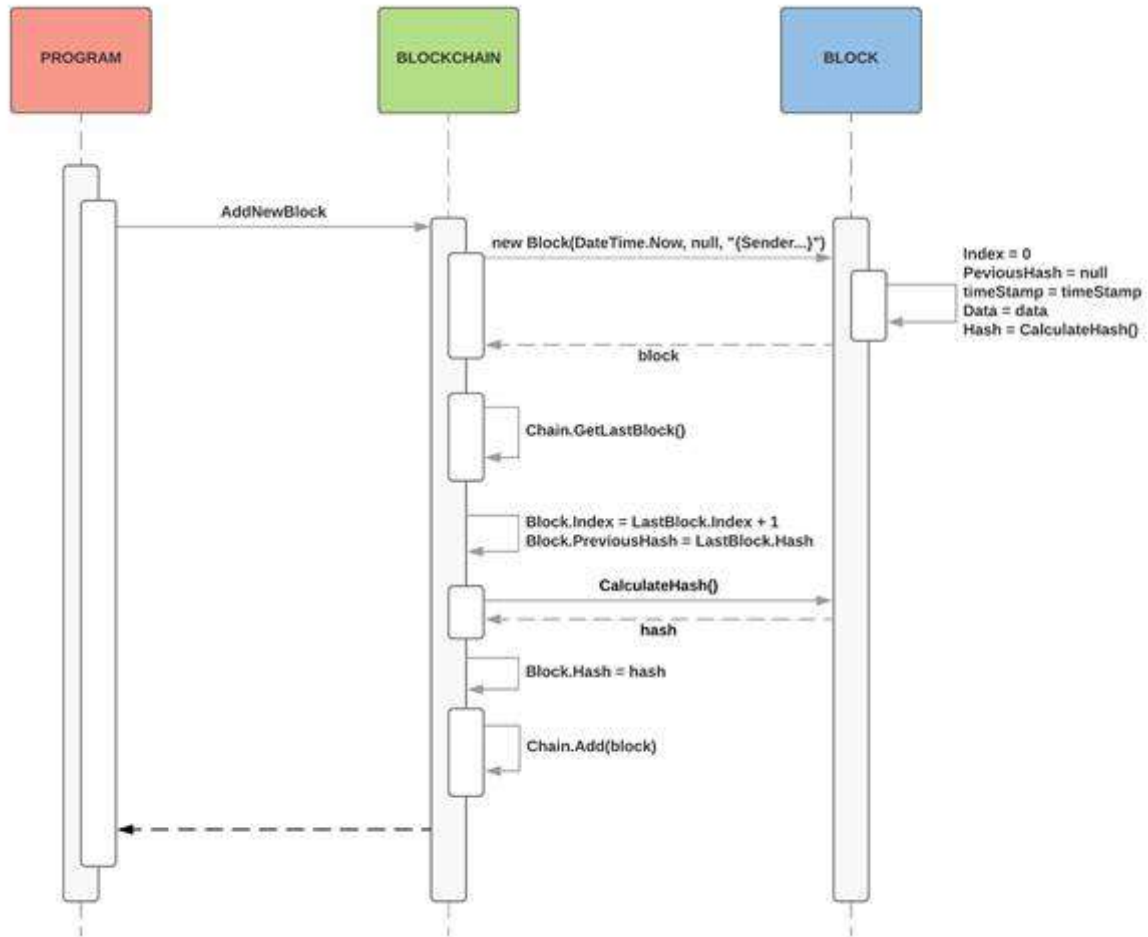


Genesis block

The genesis block is the first block of a blockchain.

- It is unique in comparison to the other blocks in a blockchain because it usually has a special function or method programmed into the code.
- This acts as a means to initiate the chain because, in blockchains, the function that creates the next block requires an input hash from the previous block.
- This, of course, is what actually links the blocks together as each block contains the hash from the previous block thus “chaining” them together. So that is why it’s necessary to hard code the genesis block into the blockchain.





In [39]:

```
#r "nuget:Bdaya.Net.Blockchain,1.2.0"
```

In [40]:

```
using Bdaya.Net.Blockchain.Configurations;
using Bdaya.Net.Blockchain.Infrastructure;
using Bdaya.Net.Blockchain.Repositories;
```

In [41]:

```
// Init connection
var configurations = new MongoConfigurations
{
    ConnectionString = "mongodb+srv://AhmedKhalil777:Password123@db1.lbuwf.mongodb.net/Tala
    Database = "TalabatDb",
    BlockCollection = "BlockchainOrders"
};

var blockchain = new BlockchainUnitOfWork<Order>(configurations);
```

In [42]:

```
// Create Genesis block
var block= await blockchain.CreateGenesisBlockAsync("Ahmed Khalil");

//display Block
display(block);
```

Id	Status	BlockchainId	TimeStamp	PreviousHash
60049610560c42b4b96f87ec	True	60049610560c42b4b96f87ed	2021-01-17 21:54:56Z	<null> J357P

In [44]:

```
id = ObjectId.GenerateNewId();

// Create Transaction
var blocOrder = new Order
{
    Address = "Mahalah",
    Id = id ,
    Meals = new List<string> { "Burger", "Fruiet Salad" },
    Status = OrderStatus.Pending,
    Owner = "Ahmed Khalil"
};

var newBlock = await blockchain.CreateTransactionAsync(
    "J357PtkmC9wshg74bJvVmLY86WM4PUjxb39mz6SQ9DM=",
    "Ibrahim",
    true,
    blocOrder);

// display
display(newBlock);
```

Id	Status	BlockchainId	TimeStamp	PreviousHash
60049623560c42b4b96f87f3	True	60049610560c42b4b96f87ed	2021-01-17 21:55:14Z	J357PtkmC9wshg74bJv\

In [48]:

```
display(newBlock.Data);
```

System.NullReferenceException: Object reference not set to an instance of an object.

```
    at Submission#50.<<Initialize>>d__0.MoveNext()
--- End of stack trace from previous location where exception was thrown ---
    at Microsoft.CodeAnalysis.Scripting.ScriptExecutionState.RunSubmissionsAs
ync[TResult](ImmutableArray`1 precedingExecutors, Func`2 currentExecutor, St
rongBox`1 exceptionHolderOpt, Func`2 catchExceptionOpt, CancellationToken ca
ncellationToken)
    at Submission#50.<<Initialize>>d__0.MoveNext()
--- End of stack trace from previous location where exception was thrown ---
    at Microsoft.CodeAnalysis.Scripting.ScriptExecutionState.RunSubmissionsAs
ync[TResult](ImmutableArray`1 precedingExecutors, Func`2 currentExecutor, St
rongBox`1 exceptionHolderOpt, Func`2 catchExceptionOpt, CancellationToken ca
ncellationToken)
```

In [49]:

```
var blocOrder = new Order
{
    Address = null,
    Id = id ,
    Meals = new List<string> { "Chicken", "Fruiets" },
    Status = OrderStatus.Approved,
    Owner = null
};

var newBlock = await blockchain.CreateTransactionAsync("uYxDh5sxnncojrPBh8Cyz6UXYUsmoZPFpQU

display(newBlock);
```

System.NullReferenceException: Object reference not set to an instance of an object.

```
    at Bdaya.Net.Blockchain.Repositories.BlockchainUnitOfWork`1.CreateTransac
tionAsync(String previousHash, String userClaim, Boolean status, TData data)
    at Submission#51.<<Initialize>>d__0.MoveNext()
--- End of stack trace from previous location where exception was thrown ---
    at Microsoft.CodeAnalysis.Scripting.ScriptExecutionState.RunSubmissionsAs
ync[TResult](ImmutableArray`1 precedingExecutors, Func`2 currentExecutor, St
rongBox`1 exceptionHolderOpt, Func`2 catchExceptionOpt, CancellationToken ca
ncellationToken)
    at Bdaya.Net.Blockchain.Repositories.BlockchainUnitOfWork`1.CreateTransac
tionAsync(String previousHash, String userClaim, Boolean status, TData data)
    at Submission#51.<<Initialize>>d__0.MoveNext()
--- End of stack trace from previous location where exception was thrown ---
    at Microsoft.CodeAnalysis.Scripting.ScriptExecutionState.RunSubmissionsAs
ync[TResult](ImmutableArray`1 precedingExecutors, Func`2 currentExecutor, St
rongBox`1 exceptionHolderOpt, Func`2 catchExceptionOpt, CancellationToken ca
ncellationToken)
```

In [50]:

```
var chain = (await blockchain.GetChangesOfChain("60049610560c42b4b96f87ed")).ToJson();  
display(chain);
```

```
{ "Changes" : { "Id" : { "User" : "Ibrahim", "IssueDate" : ISODate("2021-01-17T19:55:14.84Z") }, "Owner" : { "User" : "Ibrahim", "IssueDate" : ISODate("2021-01-17T19:55:14.84Z") }, "Status" : { "User" : "Ibrahim", "IssueDate" : ISODate("2021-01-17T19:55:14.84Z") }, "Meals" : { "User" : "Ibrahim", "IssueDate" : ISODate("2021-01-17T19:55:14.84Z") }, "Address" : { "User" : "Ibrahim", "IssueDate" : ISODate("2021-01-17T19:55:14.84Z") } }, "Data" : { "_id" : ObjectId("60049622560c42b4b96f87f1"), "Owner" : "Ahmed Khalil", "Status" : 0, "Meals" : ["Burger", "Fruiet Salad"], "Address" : "Mahalah" } }
```

In [51]:

```
var data = await blockchain.GetDataOfChain("60049610560c42b4b96f87ed");  
display(data);
```

Id	Owner	Status	Meals	Address
60049622560c42b4b96f87f1	Ahmed Khalil	Pending	[Burger, Fruiet Salad]	Mahalah

In []: