Rymbaev Eskender
Se-2421

Module 1:

#1.

Centralized ledger - This is a database managed by a single trusted operator. All users access one center that stores, updates and monitors record

Examples:

1)Bank accounts in banks(Kaspi, Freedom, Halyk, …)
2)Cloud storages(Google, Yandex, …)
3)CRM systems (AmoCRM, Salesforce, …)

Distributed ledger - This is a database that is copied and synchronized on multiple independent nodes. No one has absolute control

Examples:

1)Bitcoin blockchain
2)Ethereum
3)IBM food trust

Comparison by main parameters:

Parameters	Centralized	Distributed
Trust	Users must trust the operator	Trust is transferred from human to mathematical consensus (Pow/Pos)
Confidentiality	Hight confidentiality: the data is closed from outside	Data is visible for everyone, Except for private DLT
Fault tolerance	Crush in one center -> system will not work	Even if 30-40% of nodes are disconnected, the network will continue to work
Attack surface	Server hacking, corruption, data falsification and insiders	Network attacks, but falsifying records is extremely difficult

#2.1

Immutability is a property of the blockchain, in which data already added to a block cannot be changed without changing all subsequent blocks and without the control of most network validators. *(if Bi block have a hash of the previous Hi-1 block, then* *changing any byte in Bi-1 -> changing Hi-1 -> disrupting the chain of Bi, Bi+1 , ... Bn)*

2.2

Role of cryptographic hash-functions:
1)irreversible(the input cannot be restored)
2)the slightest data change -> completely new hash
3)used in Merkle tree, black headers, addresses

Contribution of hashes:
1)They prove the integrity of transactions
2)Create a block connection via *prevHash (prevHash: hash of the previous block, which is included in the header of the next block in the blockchain)*
3)They make changing blocks computationally impossible

2.3

When the Immutability its not working:
1)***51% attack:***
If an attacker controls > 50% of the computing power/steak, they can *spend a double-spend and roll back the chain by several blocks*
2)Hard fork:
Immutability breaks then community decides to change history
For example: Ethereum DAO Hard Fork - the attackers funds were returned by creating a new chain.
3)Private corporate blockchains:
In Hyperledger Fabric, the administrator can delete or change records, so immutability depends on the policy.

#3 Blockchain transparency vs privacy

3.1

Bitcoin Transparency:

Bitcoins is as transparent as possible:
- UTXO are public,
- Everyone can see:
  - Inputs/outputs
  - Amounts
  - Addresses
- No smart contracts -> less data, but completely public

3.2
Ethereum Transparency:
Ethereum Reveals more information than Bitcoin:

The balances of all accounts are public,
Smart contracts are public,
Storage and code can be read
Transaction data(input data) is available to everyone

For example:

You can view the USDT smart contract, its entire storage and operations.

3.3
How technologies are changing transparency:

1)Mixers (Bitcoin and Ethereum) :

For example: Tornado Cash, Wasabi. They break the link between the sender and the recipient, mixing transactions.

2)Stealth addresses:

They are used in Monero, but they are also available in Ethereum. Allow: generate one-time hidden addresses, hide the recipient.

#4 DApp Architecture

4.1
Main components:
A DApp is an application where business logic partially runs on the blockchain.

Components:

1)Smart contract layer(L1/L2)(Stores data and contains functions performed in the EVM)

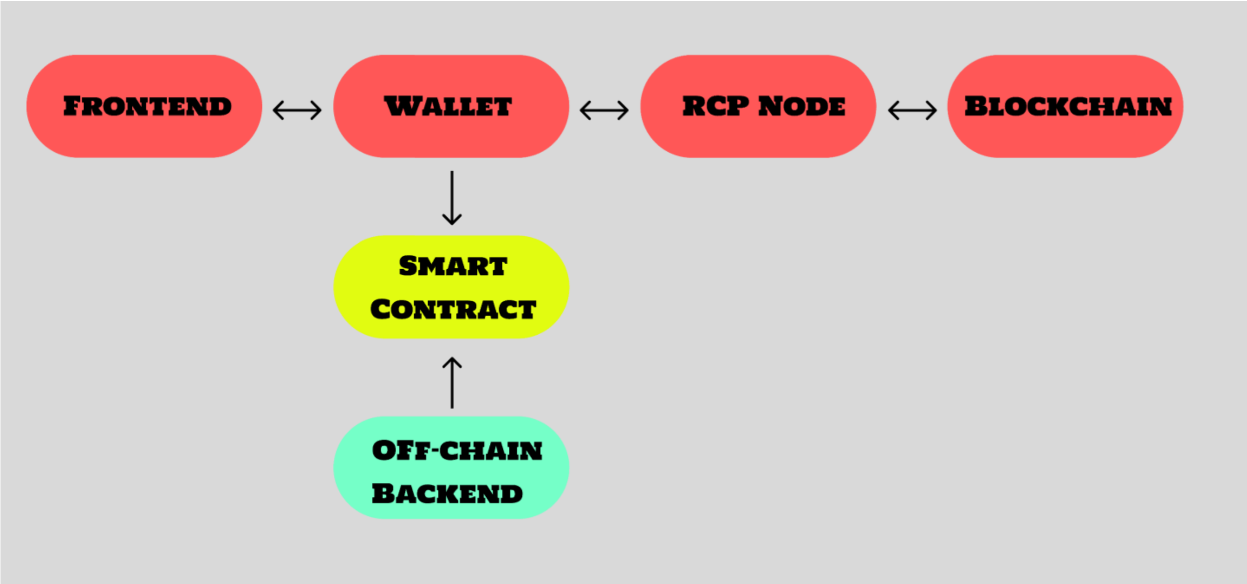
2)Off - chain backend (indexing of data, API, event handling, off-chain calcutaions)

3)Frontend(An interface that interacts with contracts via WEB3(React,Next.js))

4)Wallets(Metamask provides: signature of transactions , key management, connection to an RPC)

5)Nodes(RPC, full, light)(RPC node – interface for queries, Full node – stores the entire history, Light node – checks only the block headers)

4.2
Interaction of components diagram:



The flow of work:

DApp frontend -> calls the contract method

Wallet -> signs the transaction

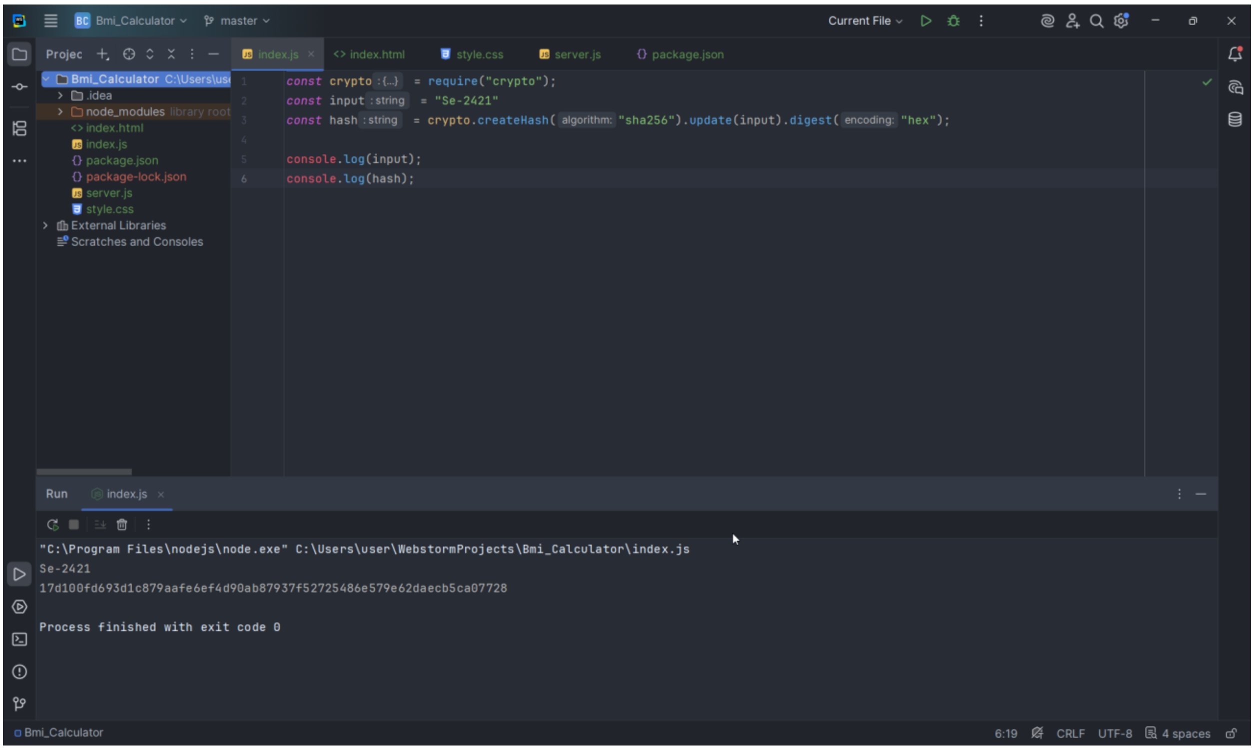
Node -> sends a transaction to the network

Smart Contract -> executed in EVM

Backend -> listens to events, updates UI

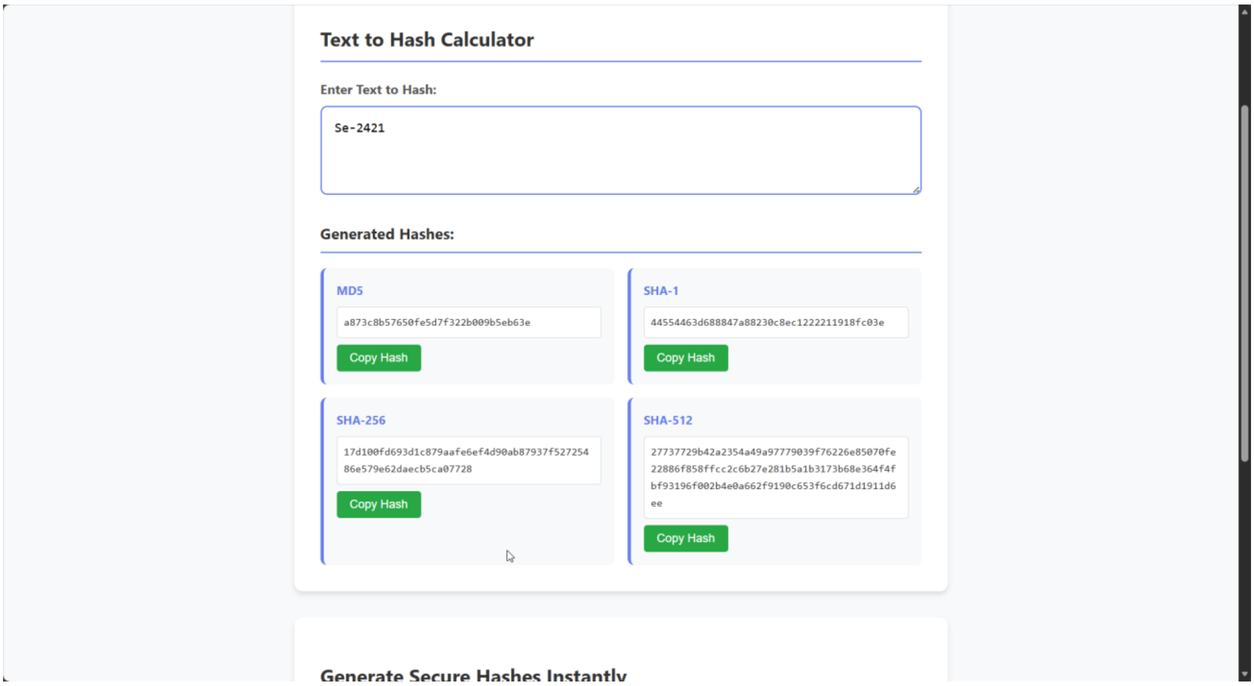
<u>Module 2:</u>

<u>#1</u>
Hash by node.js:



CreateHash("sha256") - creates hashing object
update(input) - passes the input data
digest("hex") - output hash in hexadecimal format

Online hashing tool:



#2

Comparison of the outputs:
Result : In all 2 cases, the same SHA-256 hash was obtained:
17d100fd693d1c879aafe6ef4d90ab87937f52725486e579e62daecb5ca07728

Why hashes are same?

That's because SHA-256 this is deterministic function , the same input -> always the same output and it does not depend on the language, OS or tools.

#3

Changing one bit of the input data: From Se-2421 -> se-2421

```
"C:\Program Files\nodejs\node.exe" C:\Users\user\WebstormProjects\Bmi_Calculator\index.js
se-2421
46e6b3e6830d0eeda7077d5738f88c56cf1d872447cffbbd328132cadc007ab0

Process finished with exit code 0
```

The hashes are completely different, despite the minimal change in the input.
This is due to the "avalanche effect"

Avalanche effect – property of cryptographic hashes that means that changing one bit of the input data results in a change of ~50% of the output hash bits

For SHA-256 changing 1 bit -> 128 bits will change (SHA-256 = 256 bits)

3.3
Why collisions are almost impossible?

A collision is a situation where : $SHA256(x1) = SHA256(x2)$, *where* $x1 \neq x2$

For SHA-256 possible hashes: 2^256 (2 to the power of 256)

2^256 is a HUGE number. For example all computers in earth wont able to sort through all the options.

3.4

Estimation of the probability of a collision (Birthday Paradox)

The formula for the approximate probability of a collision:

$$P \approx \frac{n^2}{2 \cdot 2^{256}}$$

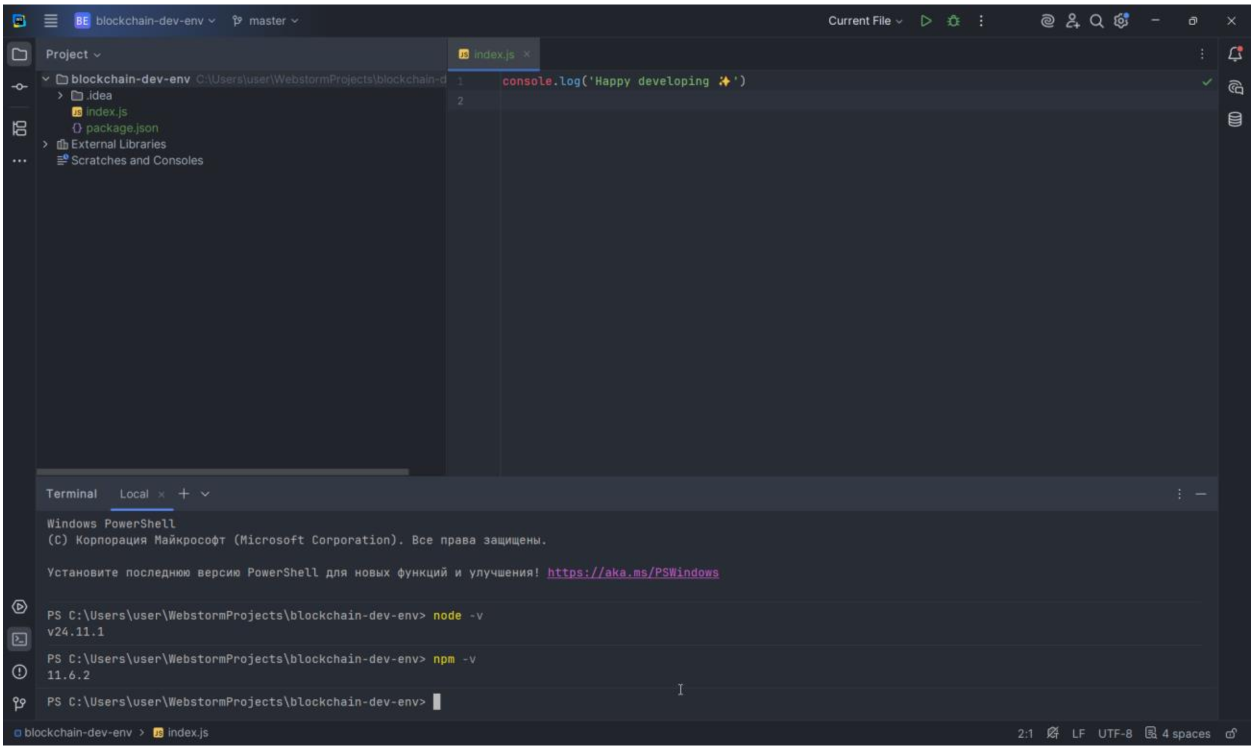
Where:

n = the number of hashed messages.

For example : n = 64

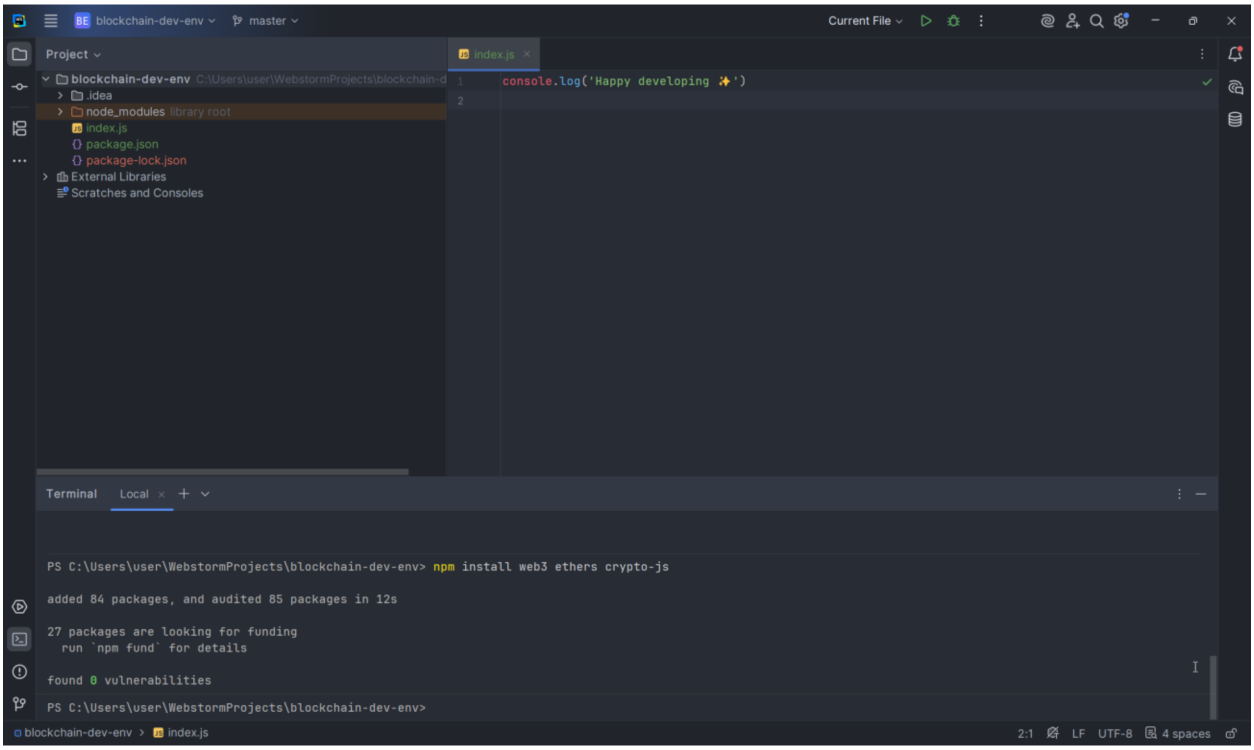
$$P \approx \frac{2^{128}}{2^{257}} = 2^{-129}$$

MODULE 3:



blockchain-dev-env
master
Current File
Project
blockchain-dev-env C:\Users\user\WebstormProjects\blockchain-d
.idea
index.js
package.json
External Libraries
Scratches and Consoles
index.js
console.log('Happy developing ✨')
Terminal
Local
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
Установите последнюю версию PowerShell для новых функций и улучшения! https://aka.ms/PSWindows
PS C:\Users\user\WebstormProjects\blockchain-dev-env> node -v
v24.11.1
PS C:\Users\user\WebstormProjects\blockchain-dev-env> npm -v
11.6.2
PS C:\Users\user\WebstormProjects\blockchain-dev-env>
blockchain-dev-env > index.js
2:1 LF UTF-8 4 spaces

Terminal
Local
PS C:\Users\user\WebstormProjects\blockchain-dev-env> npm init -y
Wrote to C:\Users\user\WebstormProjects\blockchain-dev-env\package.json:
{
"name": "blockchain-dev-env",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1"
},
"private": true,
"keywords": [],
blockchain-dev-env > index.js
2:1 LF UTF-8 4 spaces



blockchain-dev-env
master
Current File
Project
blockchain-dev-env C:\Users\user\WebstormProjects\blockchain-d
.idea
node_modules library root
index.js
package.json
package-lock.json
External Libraries
Scratches and Consoles
index.js
console.log('Happy developing ✨')
Terminal
Local
PS C:\Users\user\WebstormProjects\blockchain-dev-env> npm install web3 ethers crypto-js
added 84 packages, and audited 85 packages in 12s
27 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\user\WebstormProjects\blockchain-dev-env>
blockchain-dev-env > index.js
2:1 LF UTF-8 4 spaces

Web3.js library for :interactions with Ethereum nodes via RPC , sending transactions, reading data from the blockchain, working with smart contracts .

ethers.js – modern and more lighter alternative to Web3. Used for: working with wallets , signing transactions, interactions with smart contracts.

crypto-js is a cryptography library for js. Used for: SHA-256 calculations, hashing data, symmetric encryption , demonstrations of cryptographic principles.

MODULE 4:

#1

Bitcoin's UTXO Model:

What is UTXO:

UTXO (Unspent Transaction Output) – is an unspent transaction output that can be used as an input in a new transaction. *Bitcoin does not store balances. It stores a set of UTXOs.*

UTXO motion diagram:

Each UTXO used only once , either spent or exists

1.1 Script validation

Each UTXO contains a *scriptPubKey*
The transaction contains a *scriptSig*

The validation process:

scriptSig + *scriptPubKey* are combined -> executed by the stack VM -> If the stack ended with the value *TRUE* -> the transaction is valid

1.2 Stateless validation and concurrency:

Each UTXO is independent, transactions that do not use the same UTXOs can be verified in parallel, nodes do not need a global state

This makes Bitcoin easier, safer, and more scalable to validate.

#2

Ethereum — Account Model

Accounts types

Type	Description
EOA(Externally Owned Accaunt)	Controlled by a private key
Contract Account	Controlled by the smart contract code

2.1
Status of the Ethereum account

Each account is stored in the <u>*state trie*</u> and contains:

Field	Appointment
nonce	Replay protection
balance	ETH Balance
sstorageRoot	Storage Hash
codeHash	Hash of the bytecode

2.2

JSON – example:

```
{
  "nonce": "0x09",
  "balance": "0xde0b6b3a7640000",
  "storageRoot": "0xabc123...",
  "codeHash": "0xdef456..."
}
```

2.3
Difference from UTXO:
Ethereum stores the global state, any transaction can change the state, transaction verification is consistent

#3
Security Implications: UTXO vs Account Model

Reply attacks.
In the UTXO model, repetition within the same network is almost impossible, because each output can only be spent once. But with a fork, a transaction can be accepted in two chains.
In the account model, nonce protects against repeats within the network. However, during a fork, the same transaction can also be performed on both networks if there is no protection by chain ID.

3.2 Transaction malleability:
Bitcoin used to be vulnerable (fixed SegWit)
Ethereum is less vulnerable due to the tx hash structure

3.3 Double spend
You cant spend one UTXO twice
Nonce + block order(Account)

3.4 Smart contract attack surface
Bitcoin: No contracts -> Minimal attack surface
Ethereum: reentrancy, overflow, logic bugs.

3.5 State bloat and scalability
The Ethereum state is consistanly growing, each node stores accounts and storage contracts
Bitcoin storage only UTXO set

#4 EVM architecture

4.1
EVM bytecode execution

Solidity -> compiled into EVM bytecode
Bytecode is executed the same way on all nodes

4.2
Stack-based computation model

EVM – stack VM

PUSH 2
PUSH 3
ADD

Stack:
[2]
[2,3]
[5]

Maximum 1024 elements
Has no registers

4.3
Gas metering
Each instruction has a value: ADD -> 3 gas , SSTORE -> up to 20000 gas
Gas protects the network from endless cycles

4.4
Error handling

Type	Behaviour
Revert	Rollback of state, gas is partially returned
Invalid opcode	Full gas consumption
Out-of-gas	Full gas consumption

#5
Smart Contracts

5.1
Gas cost model

Gas = calculations + storage + calldata

Formula: *Total Fee = gasUsed * gasPrice*

5.2
Contract storage:

Storage = persistent state
Writing to storage is extremely expensive
Reading is cheaper, but not free.

5.3
Why storage writes are expensive?

Storage is expensive because Storage is stored forever, Must be replicated on all nodes, and Increases global state

5.4
Example: inefficient vs optimized code

inefficient code:

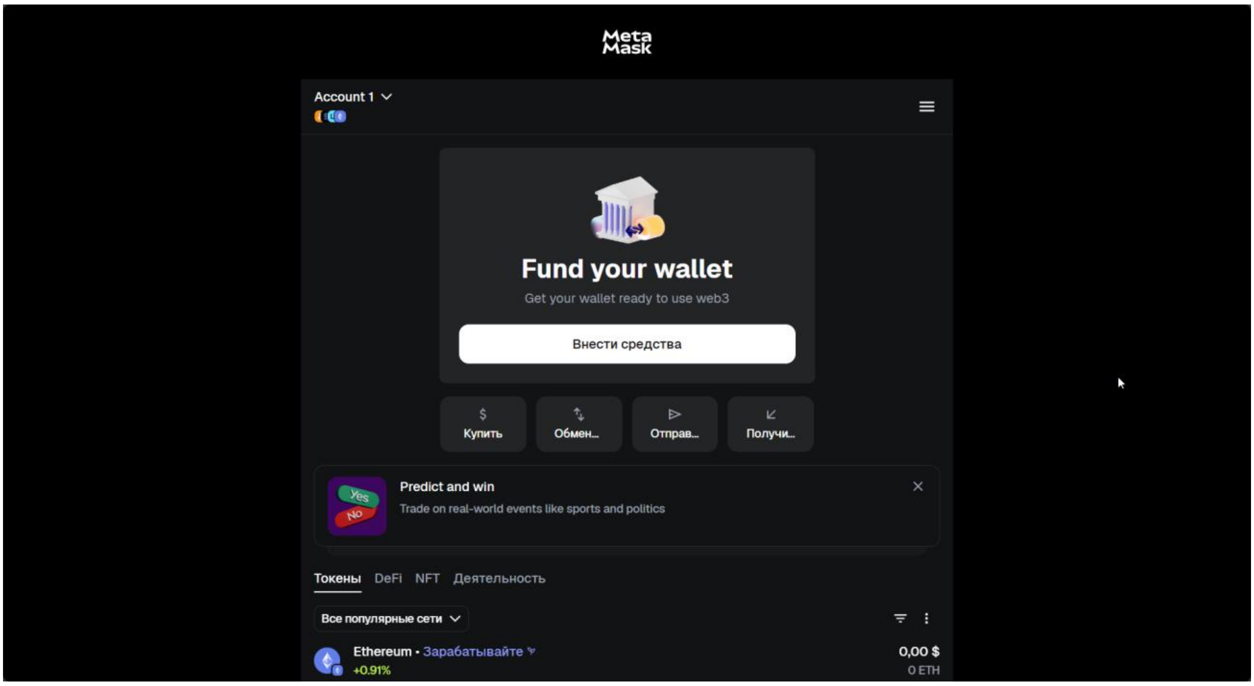
```
function set(uint x) public {
    value = x;
}
```

optimized code:

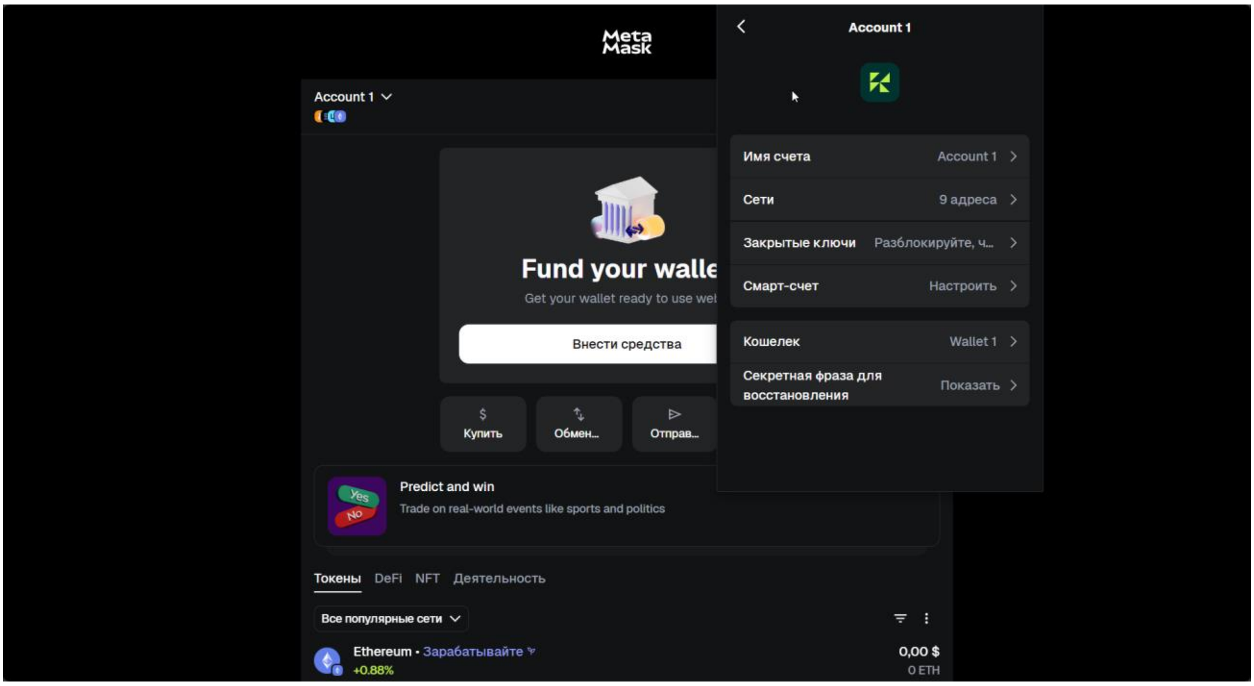
```
function set(uint x) external {
    if (value != x) value = x;
}
```

Less SSTORE -> less gas

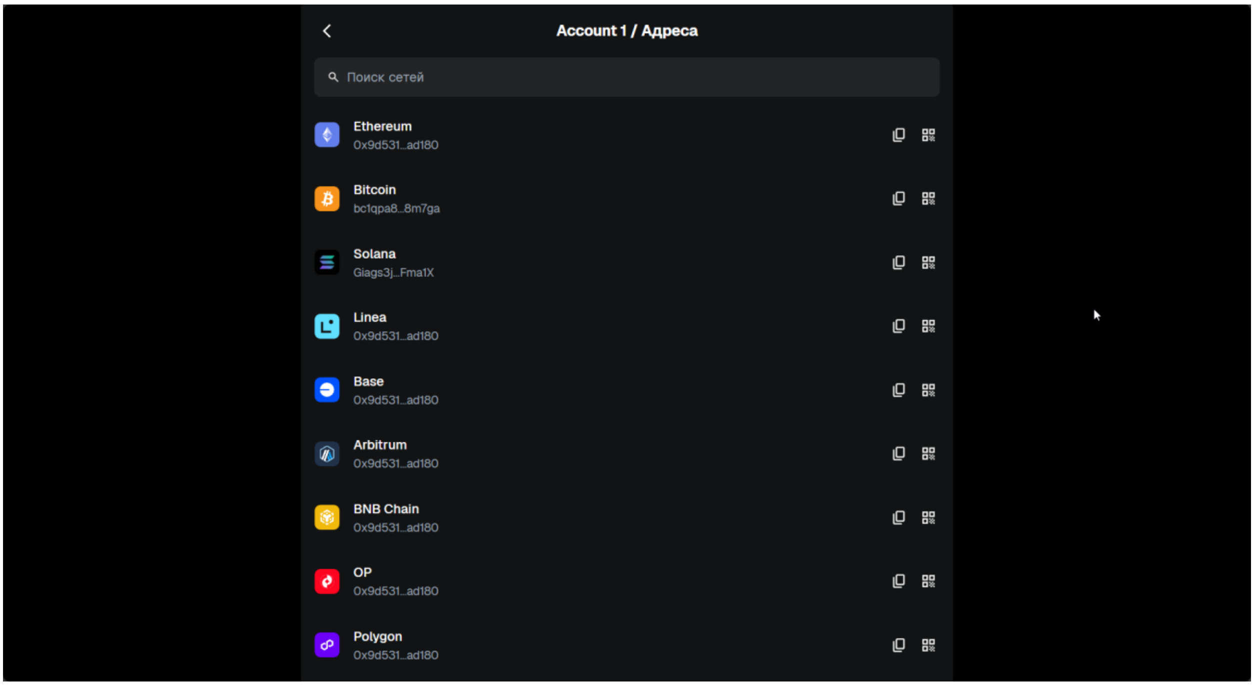
MODULE 5



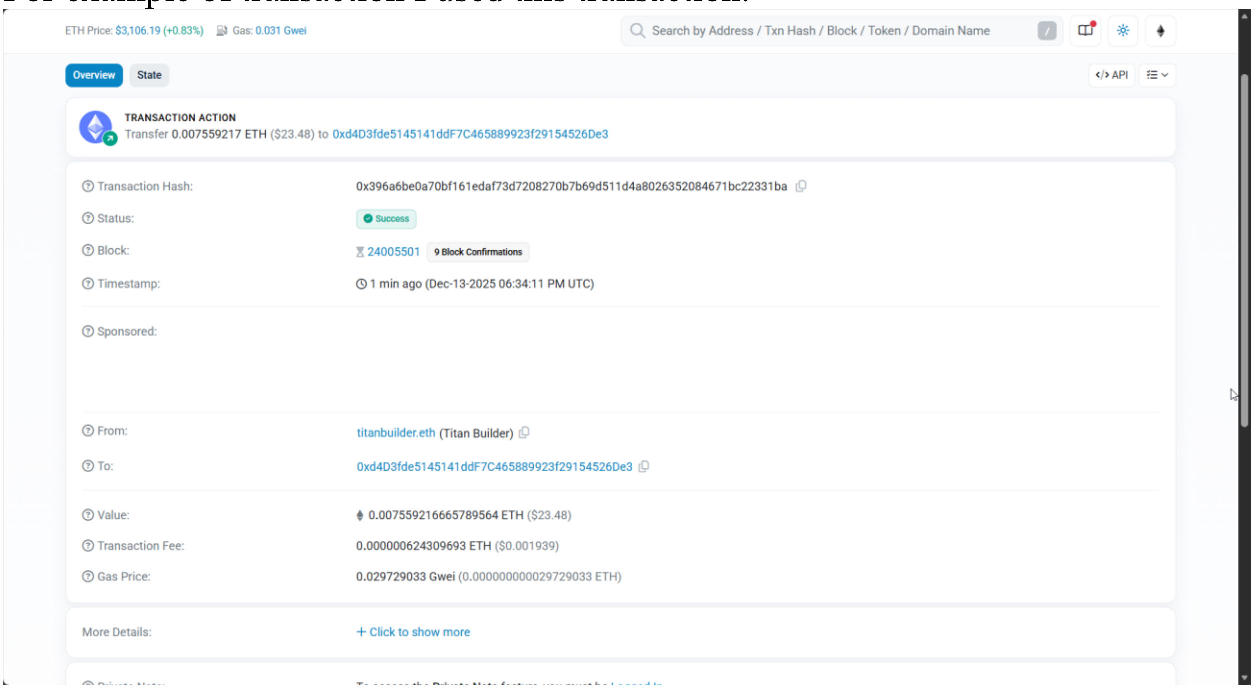
Meta Mask
Account 1
Fund your wallet
Get your wallet ready to use web3
Внести средства
Купить
Обмен...
Отправ...
Получи...
Predict and win
Trade on real-world events like sports and politics
Токены
DeFi
NFT
Деятельность
Все популярные сети
Ethereum • Зарабатывайте
+0.91%
0,00 $
0 ETH

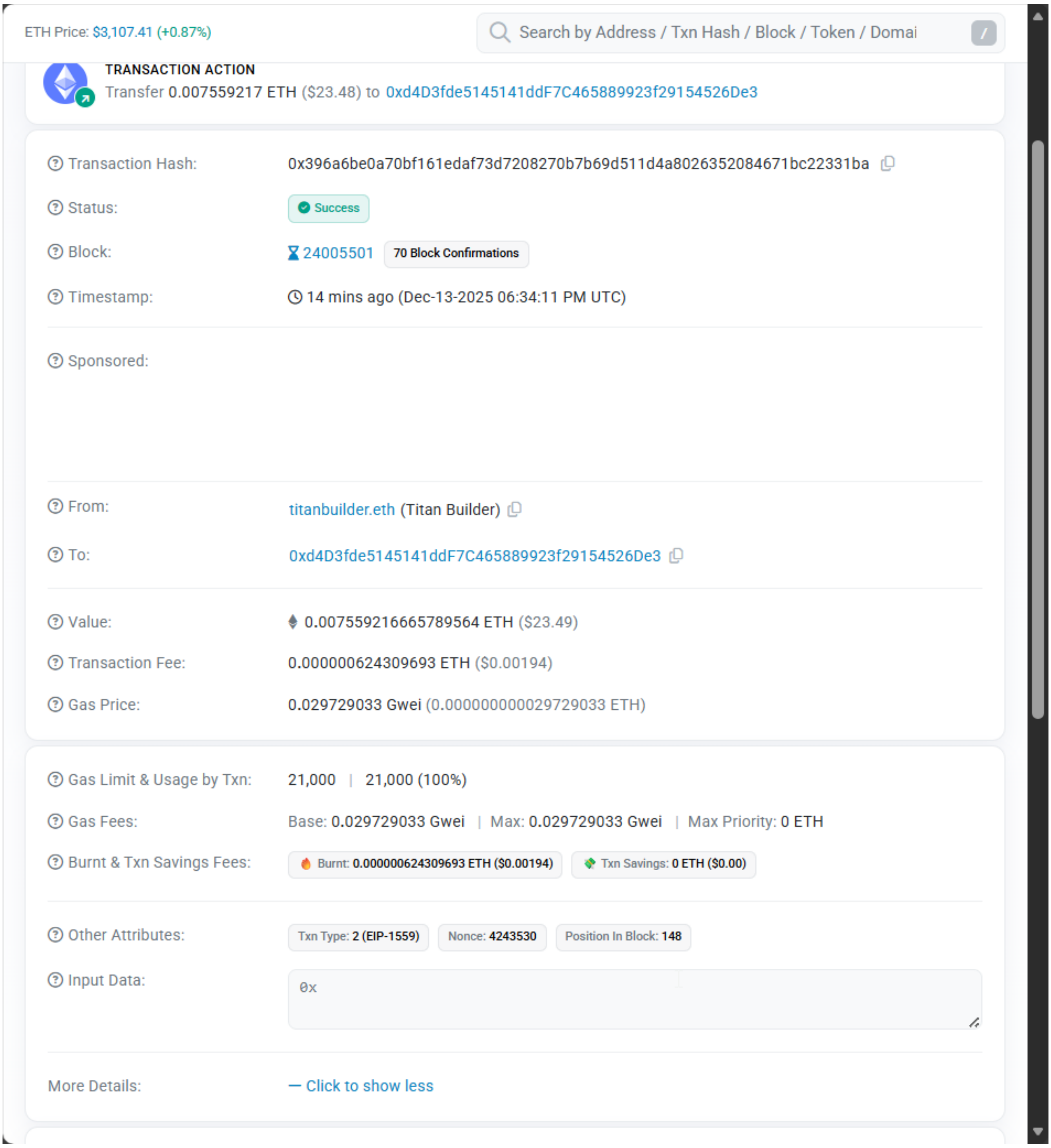


Meta Mask
Account 1
Fund your walle
Get your wallet ready to use wel
Внести средства
Купить
Обмен...
Отправ...
Predict and win
Trade on real-world events like sports and politics
Токены
DeFi
NFT
Деятельность
Все популярные сети
Ethereum • Зарабатывайте
+0.88%
0,00 $
0 ETH
Account 1
Имя счета
Account 1
Сети
9 адреса
Закрытые ключи
Разблокируйте, ч...
Смарт-счет
Настроить
Кошелек
Wallet 1
Секретная фраза для восстановления
Показать



For example of transaction I used this transaction:





Basic information about the transaction

Transaction Hash: 0x396a6be0a70bf161edaf73d7208270b7b69d511d4a8026352084671bc22331ba

Block: 24005501

Status: Success

Confirmations: 70 Block Confirmations

Timestamp: Dec-13-2025 06:34:11 PM UTC

Transaction Participants:

From: titanbuilder.eth

To: 0xd4D3fde5145141ddF7C465889923f29154526De3

Value: 0.007559216665789564 ETH ≈ $23.48

Gas Price: 0.029729033 Gwei = 0.000000029729033 ETH

Transaction type: EIP-1559

Gas Limit: 21000

Gas Used: 21000

Transaction Fee: 0.000000624309693 ETH ≈ $0.001939
Gas Fee Formula: Transaction Fee = Gas Used × Gas Price

Explanation for formula:
The transaction fee compensates validators for processing and including the transaction in a block. Since this is ETH transfer, gas consumption is munumal.

Nonse : 4243530 (Nonce represents the total number of transactions sent from the sender's account prior to this transaction.)

Input data: 0x (Empty data)