B2B - Blockchain Competition Report

Progressive Jackpot Raffle - Smart Contract DApp

A decentralized blockchain application implementing a progressive jackpot raffle system using Solidity smart contracts.

Link to repository - https://github.com/KalsoomTariq/B2B-Block-Chain-Competition.git

Overview

This project provides a complete decentralized raffle solution where users can purchase tickets, contribute to a growing prize pool, and participate in a provably fair winner selection process when the raffle concludes.

Features

Role-Based Access Control

Role	Description
Organizer	Deploys contract and manages raffle setup
Buyer	Purchases tickets and claims jackpot

Configurable Parameters

When deploying the smart contract, you can customize:

- `ticketPrice`: Price per ticket (in wei)
- `maxTicketsPerTx`: Maximum tickets a buyer can purchase per transaction
- `jackpotPercentage`: Percentage of contract balance awarded to winner (0-100)
- `raffleDuration`: Time period before raffle ends (in seconds)

Installation

1. Clone the repository:

```
git clone https://github.com/KalsoomTariq/B2B-Block-Chain-Competition.git
```

2.
3. Ensure Ganache is running and connected to MetaMask
4. Compile the contract:

```
truffle compile
```

5.
6. Install DApp dependencies:

```
cd raffle-dapp
npm install
```

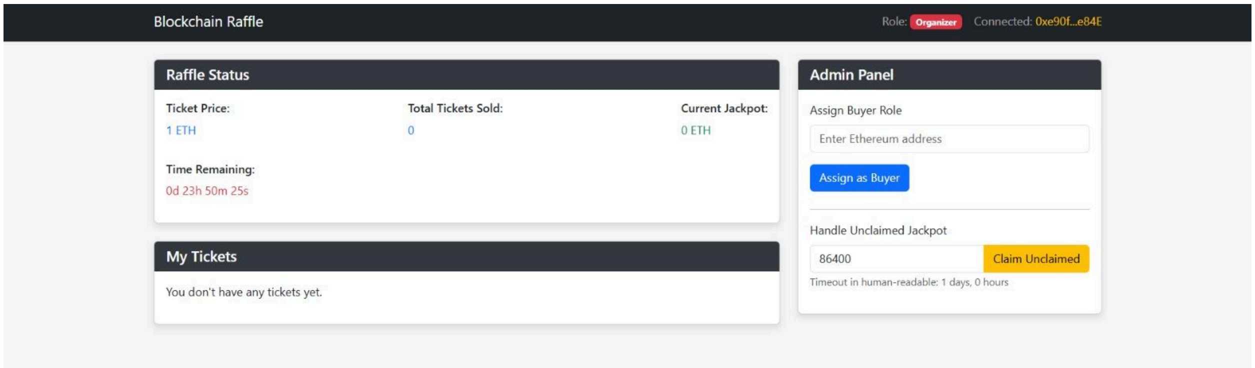
7.
8. Launch the application:

```
npm start
```

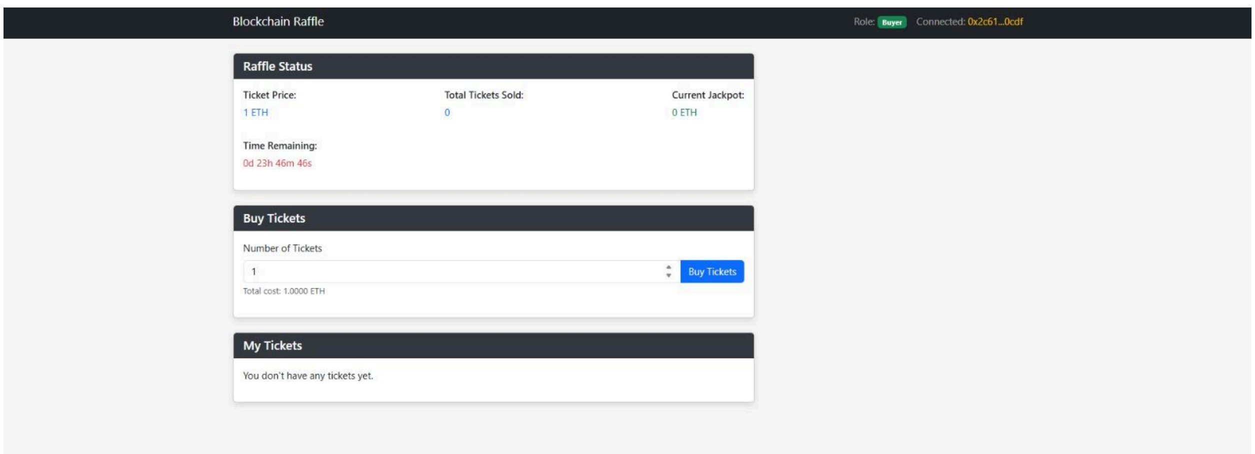
9.

Application Screenshots

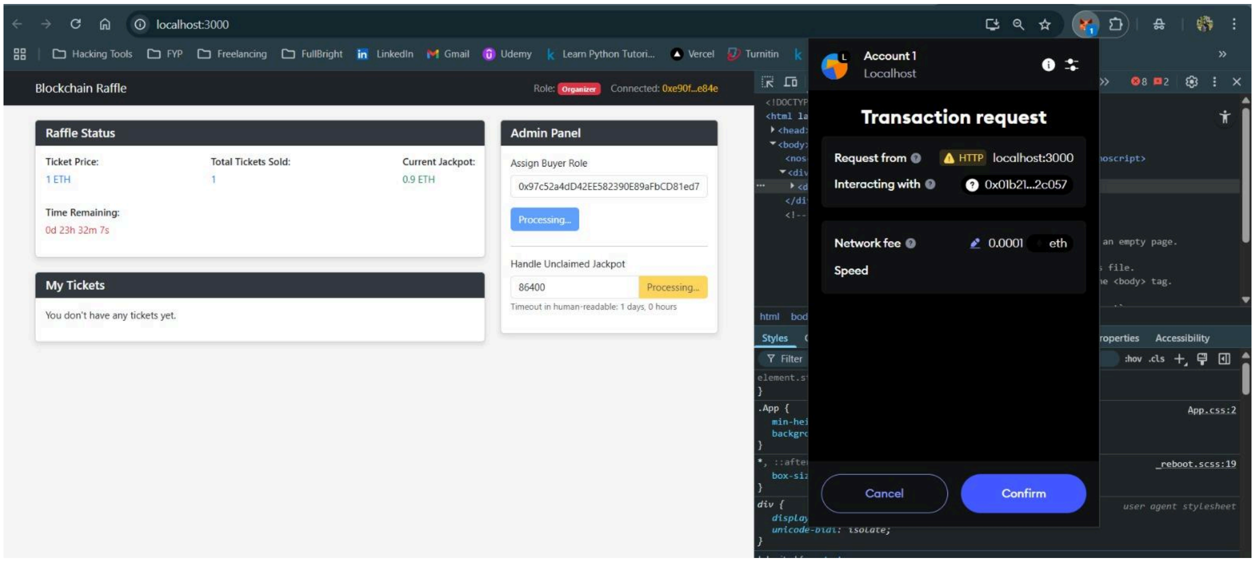
Organizer Dashboard



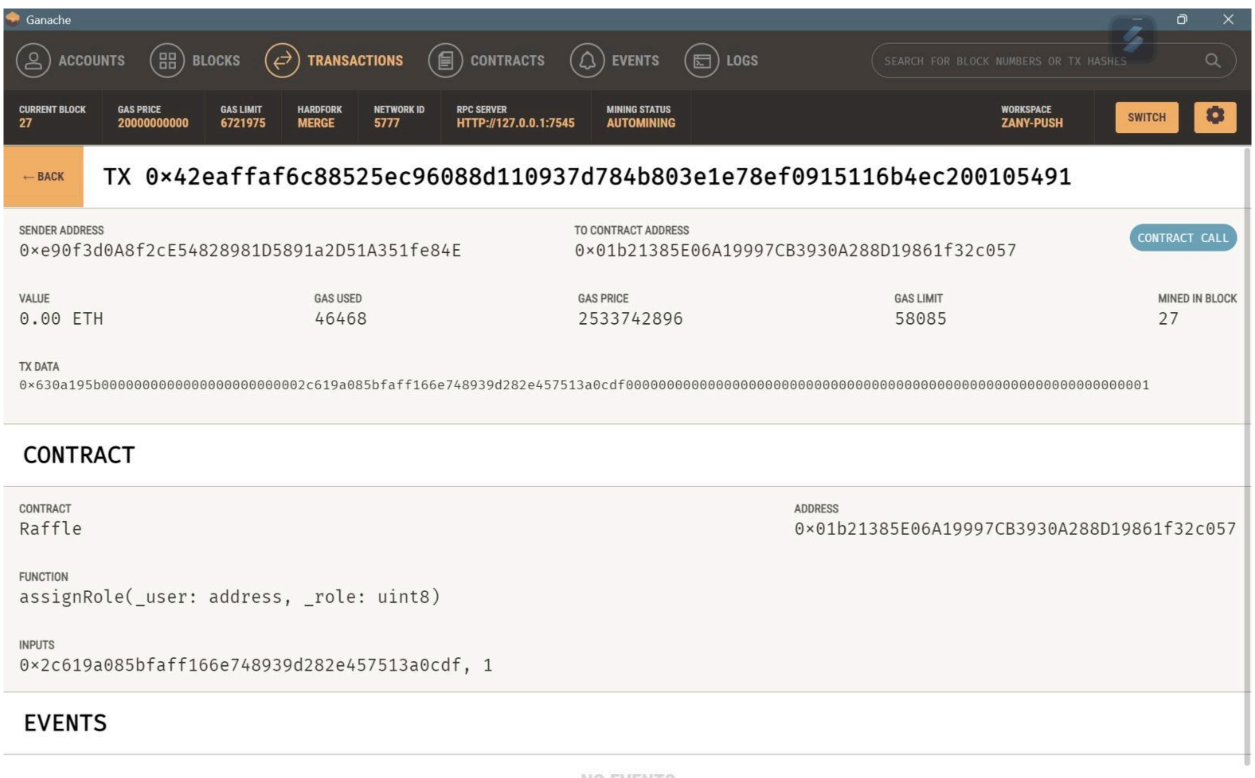
Buyer Dashboard



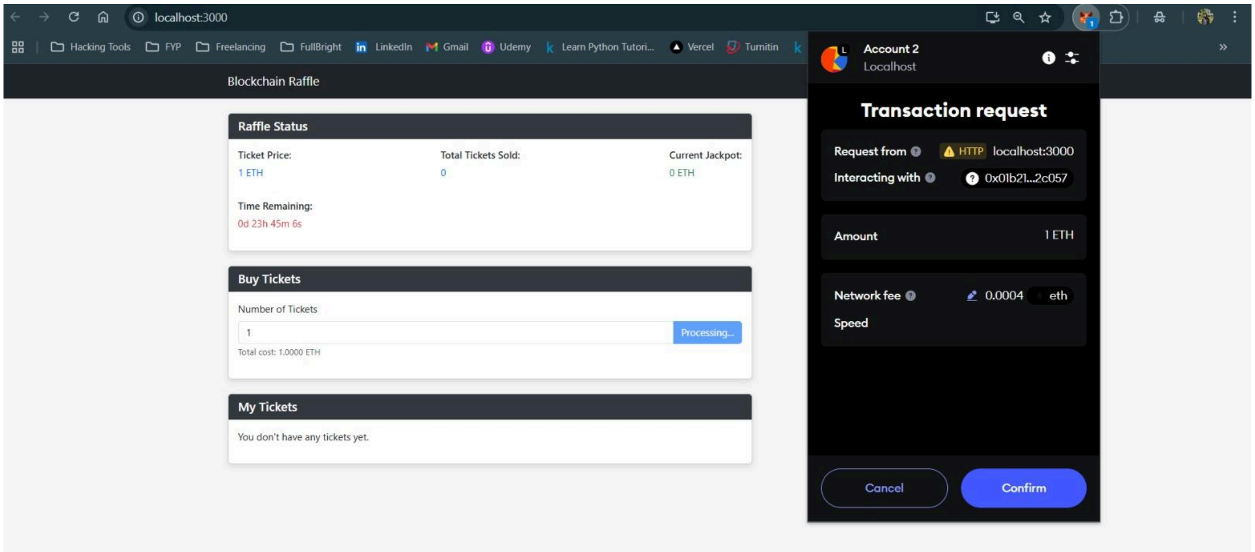
Role Assignment Interface



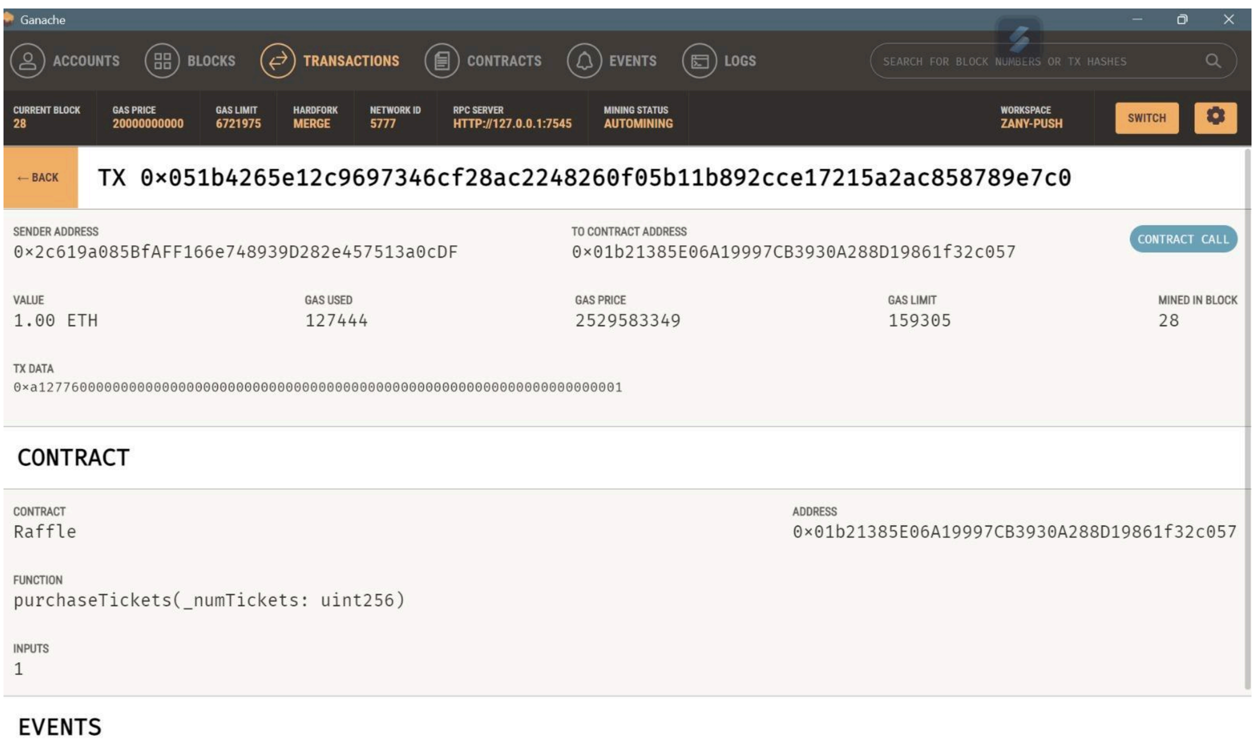
Role Assignment (Ganache View)



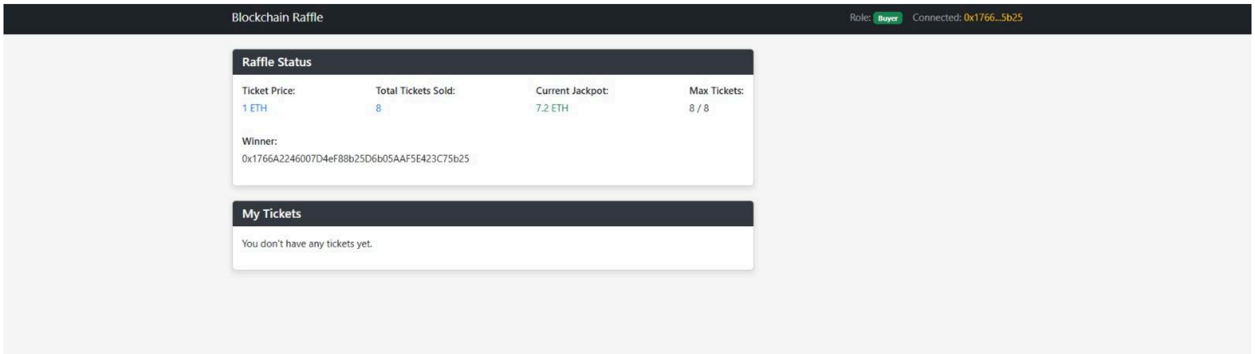
Ticket Binding Interface



Ticket Binding (Ganache View)



Ticket Binding (Ganache View)



Q2: Dynamic Adaptive State Sharding with Recursive Merkle Trie Rebalancing and Time-Sliced Execution

This repository contains our Go-based implementation for Question 2 of the B2B Blockchain Competition. The problem is divided into two distinct parts:

- Part A: Dynamic Adaptive State Sharding with Recursive Merkle Tries and Time-Sliced Execution
- Part B: In-Memory DAG-based Block Propagation with Conflict Resolution and Fork Pruning

---

Part A: Dynamic Adaptive State Sharding

Key Features

- Recursive Merkle Tries: Custom implementation for efficient state verification
- Adaptive Sharding: Dynamic redistribution based on state access patterns
- Time-Sliced Execution: Different shards active at different consensus ticks
- Cross-Shard Proofs: Compact proofs for cross-shard transactions
- Global State Consistency: Maintained through hierarchical root hash calculation

Output

```
=== Tick 0 ===
Shard 0:
  [root]:  | Hash:
3d0edd2945f5110b5fca65eda51880db899c5e85dc4bba923112180fed8c1951
    [account]:  | Hash:
54c45d3e6df9bebc483ef4854f1eaea3f8c2e366739c9e8964f81b78e45f6be3
      [bob]:  | Hash:
5cac4de5c6a7ef492e35a8244739c7960b188b2efcd0c60be2314ee059ef3f36
        [balance]: 200 | Hash:
dfb377f937ab0268fdb97244385cf33ee6de38afb2ac1774256bdee51cfcd12c
Shard 2:
  [root]:  | Hash:
56c7897ea2a838bf6366ceac4b4d6a9e4a84aab3a52d6ac2ad1c785b859850bd
    [account]:  | Hash:
a26f0b5b09e505478db12fdfa49cf6e38e912b2ee4ec6f0035b7e9338e886cda
      [carol]:  | Hash:
189ceb8661562872fe63cfc5b63032089616eff0d17e031abf6368d1071e2304
        [balance]: 300 | Hash:
9d0016c6742cf1d0acb9d91d6555812d886bc5b150e645d8e0caefabafbe3c28
Rebalancing shards...
🌐 Global Root Hash:
d26b8a7086e8cae4b7232fa3a3dcbce92b98ab386f91809a7f4f28ff8be13200
✔️  Compressed Proof for 'account.alice.balance':
[4b9066f26c2655eded09f0ded404561f4bc81fcb8a0cc9a18c4a3dfcd0f28b37]
✔️  Compressed Proof for 'account.bob.balance': []

===== TICK 1 =====
=== Tick 1 ===
Shard 1:
  [root]:  | Hash:
4813494d137e1631bba301d5acab6e7bb7aa74ce1185d456565ef51d737677b2
Shard 3:
  [root]:  | Hash:
fac900b75fc64f65847f43e620af9095856a5a39ce91d318da47e15f4e4674e3
    [account]:  | Hash:
46c2a2cfeb4aa8f295e6d75d9a20f1ad09ed98930ff57d8f79784aa6c9aaa0a8
      [alice]:  | Hash:
96c27fed2a64e71c4e911c7eb2fcc334a9e638b7617fa03cd663f39974af8aea
        [balance]: 100 | Hash:
606489318fbc182ecd7ff6cce7f3abb2de84b9931b94a9a071afd9414fc6888b
      [dave]:  | Hash:
4b9066f26c2655eded09f0ded404561f4bc81fcb8a0cc9a18c4a3dfcd0f28b37
        [nonce]: 1 | Hash:
0a78009591722cc84825ca95ee7ffa52428047ed12c9076044ebfe8665f9657f
Rebalancing shards...
🌐 Global Root Hash:
d26b8a7086e8cae4b7232fa3a3dcbce92b98ab386f91809a7f4f28ff8be13200
✔️  Compressed Proof for 'account.alice.balance':
[4b9066f26c2655eded09f0ded404561f4bc81fcb8a0cc9a18c4a3dfcd0f28b37]
```

```
✔️  Compressed Proof for 'account.bob.balance': []

===== TICK 2 =====
=== Tick 2 ===
Shard 0:
  [root]:  | Hash:
3d0edd2945f5110b5fca65eda51880db899c5e85dc4bba923112180fed8c1951
    [account]:  | Hash:
54c45d3e6df9bebc483ef4854f1eaea3f8c2e366739c9e8964f81b78e45f6be3
      [bob]:  | Hash:
5cac4de5c6a7ef492e35a8244739c7960b188b2efcd0c60be2314ee059ef3f36
        [balance]: 200 | Hash:
dfb377f937ab0268fdb97244385cf33ee6de38afb2ac1774256bdee51cfcd12c
Shard 2:
  [root]:  | Hash:
56c7897ea2a838bf6366ceac4b4d6a9e4a84aab3a52d6ac2ad1c785b859850bd
    [account]:  | Hash:
a26f0b5b09e505478db12fdfa49cf6e38e912b2ee4ec6f0035b7e9338e886cda
      [carol]:  | Hash:
189ceb8661562872fe63cfc5b63032089616eff0d17e031abf6368d1071e2304
        [balance]: 300 | Hash:
9d0016c6742cf1d0acb9d91d6555812d886bc5b150e645d8e0caefabafbe3c28
Rebalancing shards...
🌐 Global Root Hash:
d26b8a7086e8cae4b7232fa3a3dcbce92b98ab386f91809a7f4f28ff8be13200
✔️  Compressed Proof for 'account.alice.balance':
[4b9066f26c2655eded09f0ded404561f4bc81fcb8a0cc9a18c4a3dfcd0f28b37]
✔️  Compressed Proof for 'account.bob.balance': []

===== TICK 3 =====
=== Tick 3 ===
Shard 1:
  [root]:  | Hash:
4813494d137e1631bba301d5acab6e7bb7aa74ce1185d456565ef51d737677b2
Shard 3:
  [root]:  | Hash:
fac900b75fc64f65847f43e620af9095856a5a39ce91d318da47e15f4e4674e3
    [account]:  | Hash:
46c2a2cfeb4aa8f295e6d75d9a20f1ad09ed98930ff57d8f79784aa6c9aaa0a8
      [dave]:  | Hash:
4b9066f26c2655eded09f0ded404561f4bc81fcb8a0cc9a18c4a3dfcd0f28b37
        [nonce]: 1 | Hash:
0a78009591722cc84825ca95ee7ffa52428047ed12c9076044ebfe8665f9657f
      [alice]:  | Hash:
96c27fed2a64e71c4e911c7eb2fcc334a9e638b7617fa03cd663f39974af8aea
        [balance]: 100 | Hash:
606489318fbc182ecd7ff6cce7f3abb2de84b9931b94a9a071afd9414fc6888b
Rebalancing shards...
```

```
🌐 Global Root Hash:
d26b8a7086e8cae4b7232fa3a3dcbce92b98ab386f91809a7f4f28ff8be13200
✔️  Compressed Proof for 'account.alice.balance':
[4b9066f26c2655eded09f0ded404561f4bc81fcb8a0cc9a18c4a3dfcd0f28b37]
✔️  Compressed Proof for 'account.bob.balance': []
```

This confirms:

- How accounts are distributed across multiple shards.
- Use of Merkle Patricia Trie to maintain cryptographic consistency.
- How global state root hashes reflect system-wide state
- Generation of compressed proofs to efficiently verify individual account values.

Part B: DAG-based Block Propagation

Key Features

- Multi-Parent Blocks: Blocks can reference multiple parent blocks
- Parallel Block Proposals: Multiple validators can propose blocks simultaneously
- Conflict Resolution: Automatic detection and resolution of transaction conflicts
- Fork Pruning: Algorithm to determine the "heaviest" path in the DAG
- Visual Representation: ASCII visualization of the growing DAG structure

Output

```
Block 2 added with transactions [tx3]
Block 3 added with transactions [tx4 tx5 tx6]
Block 1 hash: a4f4...
Block 2 hash: 3bc9...
Block 3 hash: f00d...
Block 3 Merkle Root: 1a2b...
```

This confirms:

- Transactions are added correctly.
- DAG blocks are linking properly.
- Merkle roots are generated based on transactions in each block.

Contributors

- Kalsoom Tariq
- Kissa Zahra
- Aliza Ibrahim