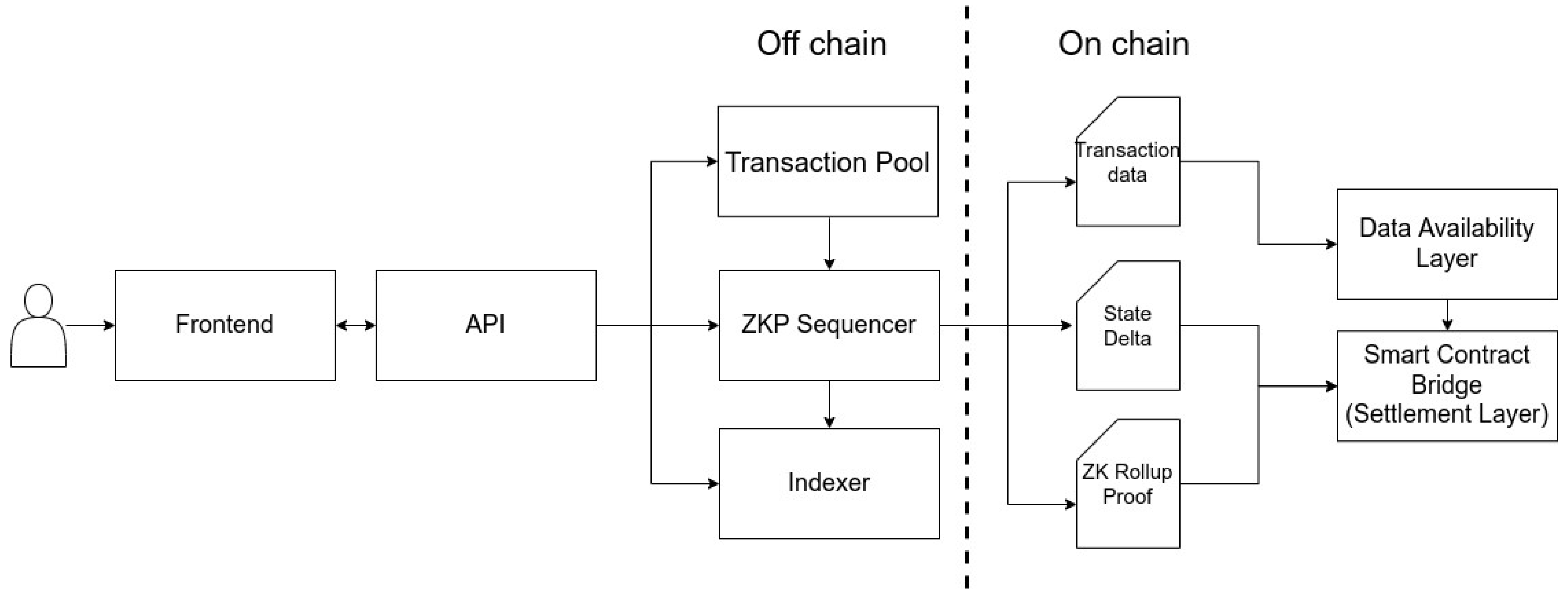
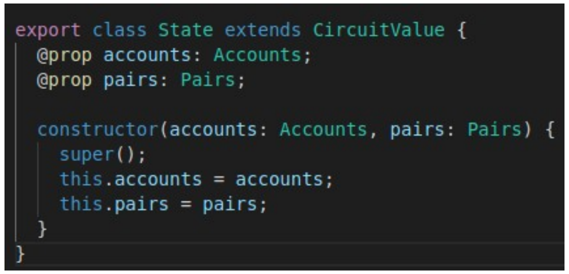
MINA SWAP

MINA L2 AMM DEX

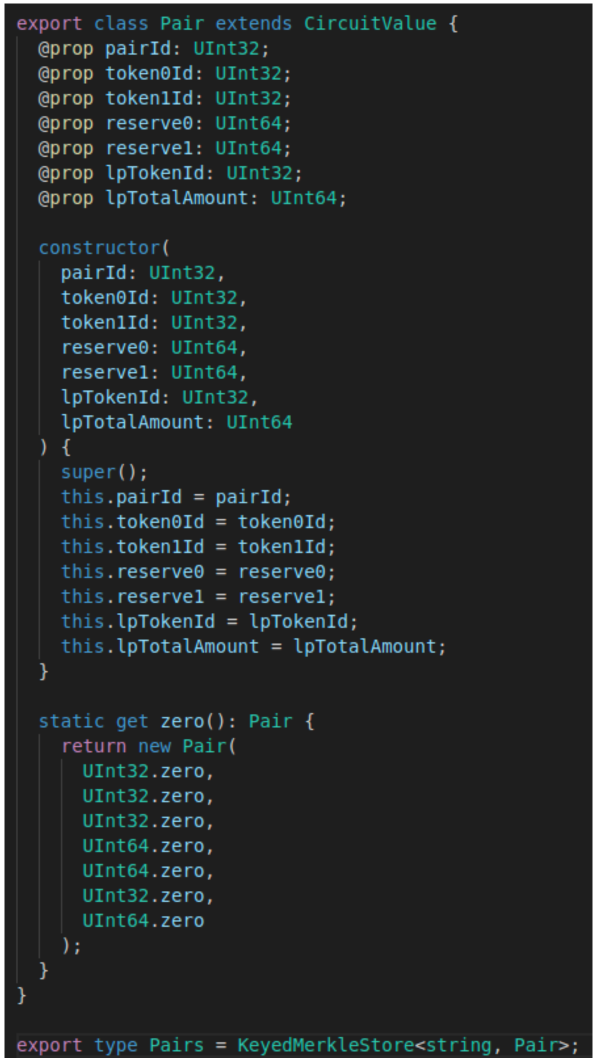
ARCHITECTURE



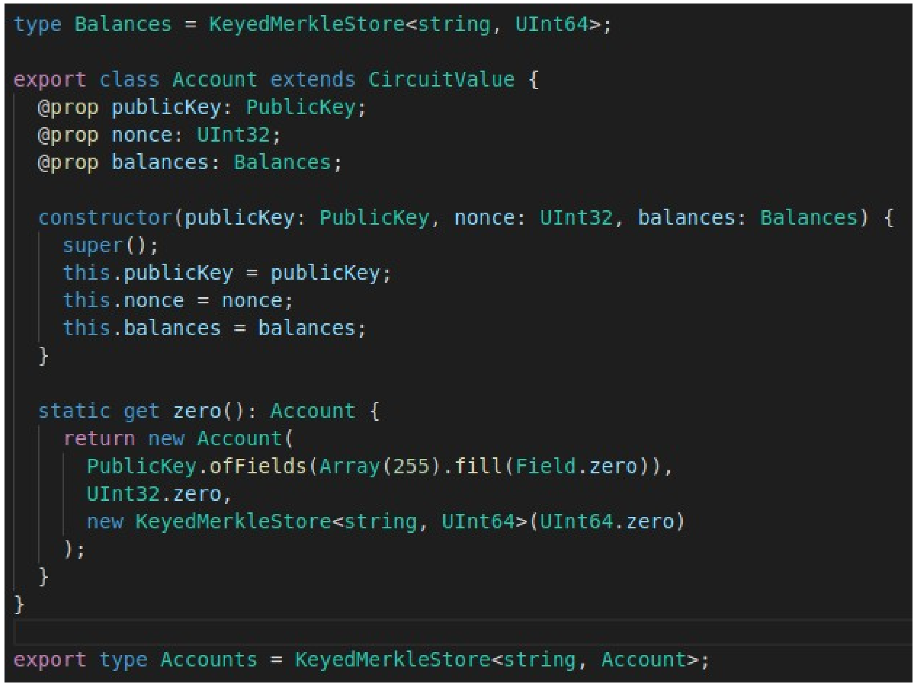
DATA MODEL



```
export class State extends CircuitValue {
  @prop accounts: Accounts;
  @prop pairs: Pairs;

  constructor(accounts: Accounts, pairs: Pairs) {
    super();
    this.accounts = accounts;
    this.pairs = pairs;
  }
}
```

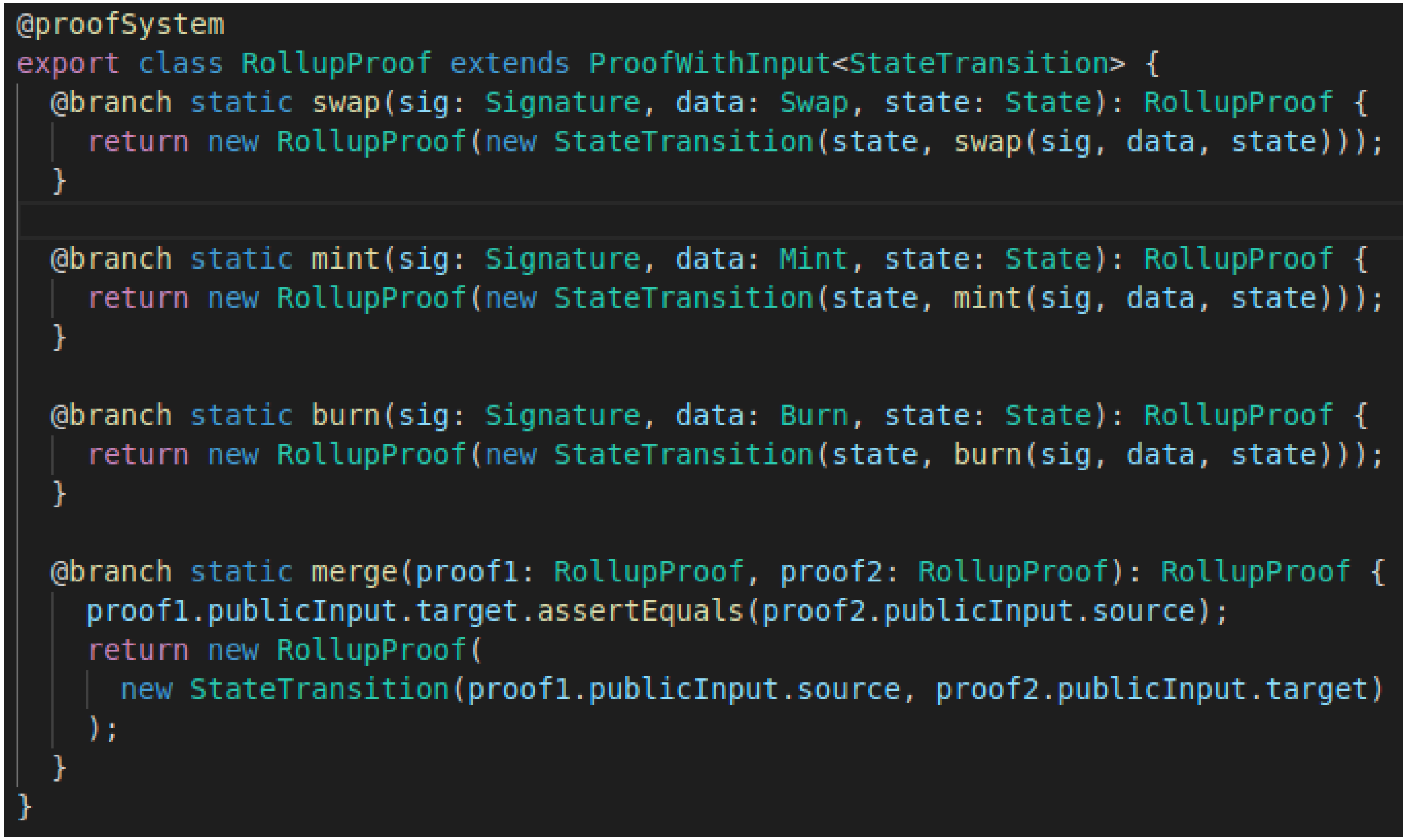


```
export class Pair extends CircuitValue {
  @prop pairId: UInt32;
  @prop token0Id: UInt32;
  @prop token1Id: UInt32;
  @prop reserve0: UInt64;
  @prop reserve1: UInt64;
  @prop lpTokenId: UInt32;
  @prop lpTotalAmount: UInt64;

  constructor(
    pairId: UInt32,
    token0Id: UInt32,
    token1Id: UInt32,
    reserve0: UInt64,
    reserve1: UInt64,
    lpTokenId: UInt32,
    lpTotalAmount: UInt64
  ) {
    super();
    this.pairId = pairId;
    this.token0Id = token0Id;
    this.token1Id = token1Id;
    this.reserve0 = reserve0;
    this.reserve1 = reserve1;
    this.lpTokenId = lpTokenId;
    this.lpTotalAmount = lpTotalAmount;
  }

  static get zero(): Pair {
    return new Pair(
      UInt32.zero,
      UInt32.zero,
      UInt32.zero,
      UInt64.zero,
      UInt64.zero,
      UInt32.zero,
      UInt64.zero
    );
  }
}

export type Pairs = KeyedMerkleStore<string, Pair>;
```



```
type Balances = KeyedMerkleStore<string, UInt64>;

export class Account extends CircuitValue {
  @prop publicKey: PublicKey;
  @prop nonce: UInt32;
  @prop balances: Balances;

  constructor(publicKey: PublicKey, nonce: UInt32, balances: Balances) {
    super();
    this.publicKey = publicKey;
    this.nonce = nonce;
    this.balances = balances;
  }

  static get zero(): Account {
    return new Account(
      PublicKey.ofFields(Array(255).fill(Field.zero)),
      UInt32.zero,
      new KeyedMerkleStore<string, UInt64>(UInt64.zero)
    );
  }
}

export type Accounts = KeyedMerkleStore<string, Account>;
```

PROOF SYSTEM



```
@proofSystem
export class RollupProof extends ProofWithInput<StateTransition> {
  @branch static swap(sig: Signature, data: Swap, state: State): RollupProof {
    return new RollupProof(new StateTransition(state, swap(sig, data, state)));
  }

  @branch static mint(sig: Signature, data: Mint, state: State): RollupProof {
    return new RollupProof(new StateTransition(state, mint(sig, data, state)));
  }

  @branch static burn(sig: Signature, data: Burn, state: State): RollupProof {
    return new RollupProof(new StateTransition(state, burn(sig, data, state)));
  }

  @branch static merge(proof1: RollupProof, proof2: RollupProof): RollupProof {
    proof1.publicInput.target.assertEquals(proof2.publicInput.source);
    return new RollupProof(
      new StateTransition(proof1.publicInput.source, proof2.publicInput.target)
    );
  }
}
```

DEMO...

Actions:

1) **Get tokens from faucet**
2) **Provide liquidity for pair and mint LP token**
3) **Make a swap against pair**
4) **Burn LP token**

FUTURE WORK

- UInt comparison operators not working. Implement new logic once supported.
- Implement proof generation once snarkyjs backend supports it
- Implement simple persistence layer for sequencer state
- Develop on-chain components
- Develop front-end
- Explore DA problem