# Project Overall File Structure

spurnet-data-management/

├── pom.xml

└── src

├── main

│ ├── java

│ │ └── com

│ │ └── spurnet

│ │ ├── Main.java

│ │ ├── cache

│ │ │ └── BlockCache.java

│ │ ├── model

│ │ │ ├── Block.java

│ │ │ ├── BlockHeader.java

│ │ │ ├── State.java

│ │ │ └── Transaction.java

│ │ ├── state

│ │ │ └── MerkleTrie.java

│ │ ├── storage

│ │ │ ├── BlockStorage.java

│ │ │ ├── StateStorage.java

│ │ │ ├── TransactionStorage.java

│ │ │ └── rocksdb

│ │ │ ├── BlockStorageImpl.java

│ │ │ ├── RocksDBManager.java

│ │ │ ├── StateStorageImpl.java

│ │ │ └── TransactionStorageImpl.java

│ │ └── sync

│ │ └── DataSynchronizer.java

│ └── resources

│ └── logback.xml

└── test

└── java

└── com

└── spurnet

├── cache

│ └── BlockCacheTest.java

├── model

│ ├── BlockTest.java

│ ├── BlockHeaderTest.java

│ ├── StateTest.java

│ └── TransactionTest.java

├── storage

│ ├── BlockStorageImplTest.java

│ ├── StateStorageImplTest.java

│ └── TransactionStorageImplTest.java

├── state

│ └── MerkleTrieTest.java

└── sync

└── DataSynchronizerTest.java

```

Detailed File Descriptions and Locations

## 1. Project Setup

- File: `pom.xml`

- Location: Project root directory `spurnet-data-management/pom.xml`

- Function: Maven project's configuration file, containing all necessary dependencies and plugins.

## 2. Data Model Design

Data Model Package Path: `src/main/java/com/spurnet/model/`

#### 2.1. BlockHeader.java

- Path: `src/main/java/com/spurnet/model/BlockHeader.java`

- Function: Defines the data structure for a block header.

```java

package com.spurnet.model;

import java.util.Objects;

public class BlockHeader {

private String previousHash;

private long timestamp;

private int nonce;

private int merkleRoot; // Placeholder for Merkle Root representation

public BlockHeader() {}

public BlockHeader(String previousHash, long timestamp, int nonce, int merkleRoot) {

this.previousHash = previousHash;

this.timestamp = timestamp;

this.nonce = nonce;

this.merkleRoot = merkleRoot;

}

// Getters and Setters

public String getPreviousHash() {

return previousHash;

}

public void setPreviousHash(String previousHash) {

this.previousHash = previousHash;

}

public long getTimestamp() {

return timestamp;

}

public void setTimestamp(long timestamp) {

this.timestamp = timestamp;

}

public int getNonce() {

return nonce;

}

public void setNonce(int nonce) {

this.nonce = nonce;

}

public int getMerkleRoot() {

return merkleRoot;

}

public void setMerkleRoot(int merkleRoot) {

this.merkleRoot = merkleRoot;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

BlockHeader that = (BlockHeader) o;

return timestamp == that.timestamp &&

nonce == that.nonce &&

merkleRoot == that.merkleRoot &&

Objects.equals(previousHash, that.previousHash);

}

@Override

public int hashCode() {

return Objects.hash(previousHash, timestamp, nonce, merkleRoot);

}

@Override

public String toString() {

return "BlockHeader{" +

"previousHash='" + previousHash + '\'' +

", timestamp=" + timestamp +

", nonce=" + nonce +

", merkleRoot=" + merkleRoot +

'}';

}

}

```

#### 2.2. Transaction.java

- Path: `src/main/java/com/spurnet/model/Transaction.java`

- Function: Defines the data structure for a transaction.

```java

package com.spurnet.model;

import java.util.Objects;

public class Transaction {

private String txId;

private String from;

private String to;

private double amount;

private long timestamp;

public Transaction() {}

public Transaction(String txId, String from, String to, double amount, long timestamp) {

this.txId = txId;

this.from = from;

this.to = to;

this.amount = amount;

this.timestamp = timestamp;

}

// Getters and Setters

public String getTxId() {

return txId;

}

public void setTxId(String txId) {

this.txId = txId;

}

public String getFrom() {

return from;

}

public void setFrom(String from) {

this.from = from;

}

public String getTo() {

return to;

}

public void setTo(String to) {

this.to = to;

}

public double getAmount() {

return amount;

}

public void setAmount(double amount) {

this.amount = amount;

}

public long getTimestamp() {

return timestamp;

}

public void setTimestamp(long timestamp) {

this.timestamp = timestamp;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Transaction that = (Transaction) o;

return Double.compare(that.amount, amount) == 0 &&

timestamp == that.timestamp &&

Objects.equals(txId, that.txId) &&

Objects.equals(from, that.from) &&

Objects.equals(to, that.to);

}

@Override

public int hashCode() {

return Objects.hash(txId, from, to, amount, timestamp);

}

@Override

public String toString() {

return "Transaction{" +

"txId='" + txId + '\'' +

", from='" + from + '\'' +

", to='" + to + '\'' +

", amount=" + amount +

", timestamp=" + timestamp +

'}';

}

}

```

#### 2.3. Block.java

- Path: `src/main/java/com/spurnet/model/Block.java`

- Function: Defines the data structure for a block.

```java

package com.spurnet.model;

import java.util.List;

import java.util.Objects;

public class Block {

private BlockHeader header;

private List<Transaction> transactions;

public Block() {}

public Block(BlockHeader header, List<Transaction> transactions) {

this.header = header;

this.transactions = transactions;

}

// Getters and Setters

public BlockHeader getHeader() {

return header;

}

public void setHeader(BlockHeader header) {

this.header = header;

}

public List<Transaction> getTransactions() {

return transactions;

}

public void setTransactions(List<Transaction> transactions) {

this.transactions = transactions;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Block block = (Block) o;

return Objects.equals(header, block.header) &&

Objects.equals(transactions, block.transactions);

}

@Override

public int hashCode() {

return Objects.hash(header, transactions);

}

@Override

public String toString() {

return "Block{" +

"header=" + header +

", transactions=" + transactions +

'}';

}

}

```

#### 2.4. State.java

- Path: `src/main/java/com/spurnet/model/State.java`

- Function: Manages account balances and the state of smart contracts.

```java

package com.spurnet.model;

import java.util.Map;

import java.util.concurrent.ConcurrentHashMap;

public class State {

private Map<String, Double> accountBalances;

public State() {

this.accountBalances = new ConcurrentHashMap<>();

}

public State(Map<String, Double> accountBalances) {

this.accountBalances = new ConcurrentHashMap<>(accountBalances);

}

// Getters and Setters

public Map<String, Double> getAccountBalances() {

return accountBalances;

}

public void setAccountBalances(Map<String, Double> accountBalances) {

this.accountBalances = new ConcurrentHashMap<>(accountBalances);

}

public void updateBalance(String account, double amount) {

this.accountBalances.put(account, amount);

}

@Override

public String toString() {

return "State{" +

"accountBalances=" + accountBalances +

'}';

}

}

```

## 3. Storage Interface Definitions

Storage Interface Package Path: `src/main/java/com/spurnet/storage/`

#### 3.1. BlockStorage.java

- Path\*\*: `src/main/java/com/spurnet/storage/BlockStorage.java`

- Function: Defines the interface for block storage.

```java

package com.spurnet.storage;

import com.spurnet.model.Block;

import java.util.Optional;

public interface BlockStorage {

void saveBlock(Block block) throws Exception;

Optional<Block> getBlock(String blockHash) throws Exception;

String getLatestBlockHash() throws Exception;

}

```

#### 3.2. TransactionStorage.java

- Path: `src/main/java/com/spurnet/storage/TransactionStorage.java`

- Function: Defines the interface for transaction storage.

```java

package com.spurnet.storage;

import com.spurnet.model.Transaction;

import java.util.List;

import java.util.Optional;

public interface TransactionStorage {

void addTransaction(Transaction tx) throws Exception;

Optional<Transaction> getTransaction(String txId) throws Exception;

List<Transaction> getPendingTransactions() throws Exception;

void removeTransaction(String txId) throws Exception;

}

```

#### 3.3. StateStorage.java

- Path: `src/main/java/com/spurnet/storage/StateStorage.java`

- Function: Defines the interface for state storage.

```java

package com.spurnet.storage;

import com.spurnet.model.State;

public interface StateStorage {

void saveState(String stateKey, State state) throws Exception;

State getState(String stateKey) throws Exception;

}

```

## 4. Storage Implementations

Storage Implementation Package Path: `src/main/java/com/spurnet/storage/rocksdb/`

#### 4.1. RocksDBManager.java

- Path: `src/main/java/com/spurnet/storage/rocksdb/RocksDBManager.java`

- Function: Manages RocksDB instances, including hot data, cold data, transaction data, and state data.

```java

package com.spurnet.storage.rocksdb;

import org.rocksdb.CompressionType;

import org.rocksdb.Options;

import org.rocksdb.RocksDB;

import org.rocksdb.RocksDBException;

public class RocksDBManager {

private static RocksDBManager instance;

private RocksDB blockDBHot;

private RocksDB blockDBCold;

private RocksDB transactionDB;

private RocksDB stateDB;

private RocksDBManager(String blockDBHotPath, String blockDBColdPath, String txDBPath, String stateDBPath) throws RocksDBException {

RocksDB.loadLibrary();

Options options = new Options()

.setCreateIfMissing(true)

.setCompressionType(CompressionType.SNAPPY\_COMPRESSION);

this.blockDBHot = RocksDB.open(options, blockDBHotPath);

this.blockDBCold = RocksDB.open(options, blockDBColdPath);

this.transactionDB = RocksDB.open(options, txDBPath);

this.stateDB = RocksDB.open(options, stateDBPath);

}

public static synchronized RocksDBManager getInstance(String blockDBHotPath, String blockDBColdPath, String txDBPath, String stateDBPath) throws RocksDBException {

if (instance == null) {

instance = new RocksDBManager(blockDBHotPath, blockDBColdPath, txDBPath, stateDBPath);

}

return instance;

}

public RocksDB getBlockDBHot() {

return blockDBHot;

}

public RocksDB getBlockDBCold() {

return blockDBCold;

}

public RocksDB getTransactionDB() {

return transactionDB;

}

public RocksDB getStateDB() {

return stateDB;

}

public void close() {

if (blockDBHot != null) blockDBHot.close();

if (blockDBCold != null) blockDBCold.close();

if (transactionDB != null) transactionDB.close();

if (stateDB != null) stateDB.close();

}

}

```

#### 4.2. BlockStorageImpl.java

- Path: `src/main/java/com/spurnet/storage/rocksdb/BlockStorageImpl.java`

- Function: Implements the `BlockStorage` interface, managing hot and cold data storage, and integrates caching.

```java

package com.spurnet.storage.rocksdb;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.spurnet.model.Block;

import com.spurnet.storage.BlockStorage;

import com.spurnet.cache.BlockCache;

import org.rocksdb.RocksDB;

import java.util.Optional;

public class BlockStorageImpl implements BlockStorage {

private final RocksDB blockDBHot;

private final RocksDB blockDBCold;

private final ObjectMapper objectMapper;

private final BlockCache blockCache;

private final int HOT\_BLOCK\_THRESHOLD = 1000; // Example threshold

public BlockStorageImpl(RocksDBManager dbManager, BlockCache cache) {

this.blockDBHot = dbManager.getBlockDBHot();

this.blockDBCold = dbManager.getBlockDBCold();

this.objectMapper = new ObjectMapper();

this.blockCache = cache;

}

@Override

public void saveBlock(Block block) throws Exception {

String blockHash = calculateBlockHash(block);

byte[] blockData = objectMapper.writeValueAsBytes(block);

// Determine if the block is hot or cold based on some criteria, e.g., block number

// This is a simplified example using blockHash length

if (blockHash.length() < HOT\_BLOCK\_THRESHOLD) {

blockDBHot.put(blockHash.getBytes(), blockData);

} else {

blockDBCold.put(blockHash.getBytes(), blockData);

}

blockCache.put(blockHash, block);

// Update the latest block hash in hot DB

blockDBHot.put("latest".getBytes(), blockHash.getBytes());

}

@Override

public Optional<Block> getBlock(String blockHash) throws Exception {

// First check the cache

Block cachedBlock = blockCache.get(blockHash);

if (cachedBlock != null) {

return Optional.of(cachedBlock);

}

// Then check the hot DB

byte[] data = blockDBHot.get(blockHash.getBytes());

if (data == null) {

// If not found, check the cold DB

data = blockDBCold.get(blockHash.getBytes());

}

if (data == null) return Optional.empty();

Block block = objectMapper.readValue(data, Block.class);

blockCache.put(blockHash, block);

return Optional.of(block);

}

@Override

public String getLatestBlockHash() throws Exception {

byte[] data = blockDBHot.get("latest".getBytes());

if (data == null) return null;

return new String(data);

}

private String calculateBlockHash(Block block) {

// Placeholder for actual hash calculation (e.g., SHA-256)

return String.valueOf(block.hashCode());

}

}

```

#### 4.3. TransactionStorageImpl.java

- Path: `src/main/java/com/spurnet/storage/rocksdb/TransactionStorageImpl.java`

- Function\*\*: Implements the `TransactionStorage` interface, managing transaction storage.

```java

package com.spurnet.storage.rocksdb;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.spurnet.model.Transaction;

import com.spurnet.storage.TransactionStorage;

import org.rocksdb.RocksDB;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

public class TransactionStorageImpl implements TransactionStorage {

private final RocksDB transactionDB;

private final ObjectMapper objectMapper;

private static final String PENDING\_KEY = "pending";

public TransactionStorageImpl(RocksDBManager dbManager) {

this.transactionDB = dbManager.getTransactionDB();

this.objectMapper = new ObjectMapper();

}

@Override

public void addTransaction(Transaction tx) throws Exception {

byte[] txData = objectMapper.writeValueAsBytes(tx);

transactionDB.put(tx.getTxId().getBytes(), txData);

// Optionally, maintain a list of pending transactions

// This is a simplified example

}

@Override

public Optional<Transaction> getTransaction(String txId) throws Exception {

byte[] data = transactionDB.get(txId.getBytes());

if (data == null) return Optional.empty();

Transaction tx = objectMapper.readValue(data, Transaction.class);

return Optional.of(tx);

}

@Override

public List<Transaction> getPendingTransactions() throws Exception {

// Simplified: Iterate through all keys and collect transactions

List<Transaction> pendingTxs = new ArrayList<>();

try (var iter = transactionDB.newIterator()) {

iter.seekToFirst();

while (iter.isValid()) {

String key = new String(iter.key());

if (!key.equals(PENDING\_KEY)) { // Exclude special keys

Transaction tx = objectMapper.readValue(iter.value(), Transaction.class);

pendingTxs.add(tx);

}

iter.next();

}

}

return pendingTxs;

}

@Override

public void removeTransaction(String txId) throws Exception {

transactionDB.delete(txId.getBytes());

}

}

```

#### 4.4. StateStorageImpl.java

- Path: `src/main/java/com/spurnet/storage/rocksdb/StateStorageImpl.java`

- Function: Implements the `StateStorage` interface, managing state storage.

```java

package com.spurnet.storage.rocksdb;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.spurnet.model.State;

import com.spurnet.storage.StateStorage;

import org.rocksdb.RocksDB;

public class StateStorageImpl implements StateStorage {

private final RocksDB stateDB;

private final ObjectMapper objectMapper;

public StateStorageImpl(RocksDBManager dbManager) {

this.stateDB = dbManager.getStateDB();

this.objectMapper = new ObjectMapper();

}

@Override

public void saveState(String stateKey, State state) throws Exception {

byte[] stateData = objectMapper.writeValueAsBytes(state);

stateDB.put(stateKey.getBytes(), stateData);

}

@Override

public State getState(String stateKey) throws Exception {

byte[] data = stateDB.get(stateKey.getBytes());

if (data == null) return null;

return objectMapper.readValue(data, State.class);

}

}

```

## 5. Merkle Trie Implementation

Merkle Trie Package Path: `src/main/java/com/spurnet/state/`

#### 5.1. MerkleTrie.java

- Path: `src/main/java/com/spurnet/state/MerkleTrie.java`

- Function: Implements a Merkle Trie for managing the state of accounts and smart contracts.

```java

package com.spurnet.state;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.\*;

public class MerkleTrie {

private final Node root;

public MerkleTrie() {

this.root = new Node("");

}

public void insert(String key, String value) {

Node current = root;

for (char c : key.toCharArray()) {

current = current.getChildren().computeIfAbsent(c, k -> new Node(String.valueOf(k)));

}

current.setValue(value);

}

public String get(String key) {

Node current = root;

for (char c : key.toCharArray()) {

current = current.getChildren().get(c);

if (current == null) return null;

}

return current.getValue();

}

public String getMerkleRoot() throws NoSuchAlgorithmException {

return computeHash(root);

}

private String computeHash(Node node) throws NoSuchAlgorithmException {

MessageDigest digest = MessageDigest.getInstance("SHA-256");

for (Node child : node.getChildren().values()) {

digest.update(computeHash(child).getBytes());

}

if (node.getValue() != null) {

digest.update(node.getValue().getBytes());

}

byte[] hashBytes = digest.digest();

return bytesToHex(hashBytes);

}

private String bytesToHex(byte[] bytes) {

StringBuilder sb = new StringBuilder();

for(byte b : bytes){

sb.append(String.format("%02x", b));

}

return sb.toString();

}

private static class Node {

private final String key;

private String value;

private final Map<Character, Node> children;

public Node(String key) {

this.key = key;

this.children = new HashMap<>();

}

public String getKey() {

return key;

}

public String getValue() {

return value;

}

public void setValue(String value) {

this.value = value;

}

public Map<Character, Node> getChildren() {

return children;

}

}

}

```

## 6. Data Synchronization and Validation

Data Synchronization Package Path: `src/main/java/com/spurnet/sync/`

#### 6.1. DataSynchronizer.java

- Path: `src/main/java/com/spurnet/sync/DataSynchronizer.java`

- Function: Implements data synchronization and validation mechanisms to ensure data consistency across different nodes.

```java

package com.spurnet.sync;

import com.spurnet.model.Block;

import com.spurnet.storage.BlockStorage;

public class DataSynchronizer {

private final BlockStorage blockStorage;

public DataSynchronizer(BlockStorage blockStorage) {

this.blockStorage = blockStorage;

}

public void synchronize(Block newBlock) throws Exception {

String latestHash = blockStorage.getLatestBlockHash();

if (latestHash == null || isValidNewBlock(newBlock, latestHash)) {

blockStorage.saveBlock(newBlock);

// Additional synchronization logic (e.g., notifying other nodes)

} else {

throw new Exception("Invalid block received");

}

}

private boolean isValidNewBlock(Block newBlock, String previousHash) {

// Simplified validation: Check if previousHash matches

return newBlock.getHeader().getPreviousHash().equals(previousHash);

}

}

```

## 7. Storage Optimization

#### 7.1. Data Compression

Configure Compression Options: Already configured Snappy compression in `RocksDBManager.java`.

```java

import org.rocksdb.CompressionType;

import org.rocksdb.Options;

// Modify Options configuration in RocksDBManager's constructor

private RocksDBManager(String blockDBHotPath, String blockDBColdPath, String txDBPath, String stateDBPath) throws RocksDBException {

RocksDB.loadLibrary();

Options options = new Options()

.setCreateIfMissing(true)

.setCompressionType(CompressionType.SNAPPY\_COMPRESSION);

this.blockDBHot = RocksDB.open(options, blockDBHotPath);

this.blockDBCold = RocksDB.open(options, blockDBColdPath);

this.transactionDB = RocksDB.open(options, txDBPath);

this.stateDB = RocksDB.open(options, stateDBPath);

}

```

#### 7.2. Caching Mechanism

Cache Package Path: `src/main/java/com/spurnet/cache/`

##### 7.2.1. BlockCache.java

- Path: `src/main/java/com/spurnet/cache/BlockCache.java`

- Function: Implements block caching using Caffeine to enhance data access speed.

```java

package com.spurnet.cache;

import com.spurnet.model.Block;

import com.github.benmanes.caffeine.cache.Cache;

import com.github.benmanes.caffeine.cache.Caffeine;

import java.util.concurrent.TimeUnit;

public class BlockCache {

private final Cache<String, Block> cache;

public BlockCache() {

this.cache = Caffeine.newBuilder()

.maximumSize(1000)

.expireAfterWrite(10, TimeUnit.MINUTES)

.build();

}

public Block get(String blockHash) {

return cache.getIfPresent(blockHash);

}

public void put(String blockHash, Block block) {

cache.put(blockHash, block);

}

public void invalidate(String blockHash) {

cache.invalidate(blockHash);

}

public void invalidateAll() {

cache.invalidateAll();

}

}

```

##### 7.2.2. Integrate Cache into BlockStorageImpl

- Modify Path: `src/main/java/com/spurnet/storage/rocksdb/BlockStorageImpl.java`

- Function: Integrates `BlockCache` into `BlockStorageImpl`.

```java

package com.spurnet.storage.rocksdb;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.spurnet.model.Block;

import com.spurnet.storage.BlockStorage;

import com.spurnet.cache.BlockCache;

import org.rocksdb.RocksDB;

import java.util.Optional;

public class BlockStorageImpl implements BlockStorage {

private final RocksDB blockDBHot;

private final RocksDB blockDBCold;

private final ObjectMapper objectMapper;

private final BlockCache blockCache;

private final int HOT\_BLOCK\_THRESHOLD = 1000; // Example threshold

public BlockStorageImpl(RocksDBManager dbManager, BlockCache cache) {

this.blockDBHot = dbManager.getBlockDBHot();

this.blockDBCold = dbManager.getBlockDBCold();

this.objectMapper = new ObjectMapper();

this.blockCache = cache;

}

@Override

public void saveBlock(Block block) throws Exception {

String blockHash = calculateBlockHash(block);

byte[] blockData = objectMapper.writeValueAsBytes(block);

// Determine if the block is hot or cold based on some criteria, e.g., block number

// This is a simplified example using blockHash length

if (blockHash.length() < HOT\_BLOCK\_THRESHOLD) {

blockDBHot.put(blockHash.getBytes(), blockData);

} else {

blockDBCold.put(blockHash.getBytes(), blockData);

}

blockCache.put(blockHash, block);

// Update the latest block hash in hot DB

blockDBHot.put("latest".getBytes(), blockHash.getBytes());

}

@Override

public Optional<Block> getBlock(String blockHash) throws Exception {

// First check the cache

Block cachedBlock = blockCache.get(blockHash);

if (cachedBlock != null) {

return Optional.of(cachedBlock);

}

// Then check the hot DB

byte[] data = blockDBHot.get(blockHash.getBytes());

if (data == null) {

// If not found, check the cold DB

data = blockDBCold.get(blockHash.getBytes());

}

if (data == null) return Optional.empty();

Block block = objectMapper.readValue(data, Block.class);

blockCache.put(blockHash, block);

return Optional.of(block);

}

@Override

public String getLatestBlockHash() throws Exception {

byte[] data = blockDBHot.get("latest".getBytes());

if (data == null) return null;

return new String(data);

}

private String calculateBlockHash(Block block) {

// Placeholder for actual hash calculation (e.g., SHA-256)

return String.valueOf(block.hashCode());

}

}

```

#### 7.3. Hot and Cold Data Separation

Hot and Cold Data Separation Package Path: Implemented in `RocksDBManager.java` and `BlockStorageImpl.java`.

##### 7.3.1. Modify RocksDBManager.java

- Path: `src/main/java/com/spurnet/storage/rocksdb/RocksDBManager.java`

- Function\*\*: Creates separate RocksDB instances for hot and cold data.

```java

package com.spurnet.storage.rocksdb;

import org.rocksdb.CompressionType;

import org.rocksdb.Options;

import org.rocksdb.RocksDB;

import org.rocksdb.RocksDBException;

public class RocksDBManager {

private static RocksDBManager instance;

private RocksDB blockDBHot;

private RocksDB blockDBCold;

private RocksDB transactionDB;

private RocksDB stateDB;

private RocksDBManager(String blockDBHotPath, String blockDBColdPath, String txDBPath, String stateDBPath) throws RocksDBException {

RocksDB.loadLibrary();

Options options = new Options()

.setCreateIfMissing(true)

.setCompressionType(CompressionType.SNAPPY\_COMPRESSION);

this.blockDBHot = RocksDB.open(options, blockDBHotPath);

this.blockDBCold = RocksDB.open(options, blockDBColdPath);

this.transactionDB = RocksDB.open(options, txDBPath);

this.stateDB = RocksDB.open(options, stateDBPath);

}

public static synchronized RocksDBManager getInstance(String blockDBHotPath, String blockDBColdPath, String txDBPath, String stateDBPath) throws RocksDBException {

if (instance == null) {

instance = new RocksDBManager(blockDBHotPath, blockDBColdPath, txDBPath, stateDBPath);

}

return instance;

}

public RocksDB getBlockDBHot() {

return blockDBHot;

}

public RocksDB getBlockDBCold() {

return blockDBCold;

}

public RocksDB getTransactionDB() {

return transactionDB;

}

public RocksDB getStateDB() {

return stateDB;

}

public void close() {

if (blockDBHot != null) blockDBHot.close();

if (blockDBCold != null) blockDBCold.close();

if (transactionDB != null) transactionDB.close();

if (stateDB != null) stateDB.close();

}

}

```

##### 7.3.2. Use Hot and Cold Databases for Block Storage

- Path: `src/main/java/com/spurnet/storage/rocksdb/BlockStorageImpl.java`

- Function: Stores blocks in different databases based on classification criteria (e.g., hash length in this example).

```java

// Already shown in the above BlockStorageImpl.java implementation

```

## 8. Complete Example

Project Entry Class Path\*\*: `src/main/java/com/spurnet/Main.java`

#### 8.1. Main.java

- Path: `src/main/java/com/spurnet/Main.java`

- Function: Initializes storage modules, adds transactions, saves blocks and states, and outputs results to the console.

```java

package com.spurnet;

import com.spurnet.cache.BlockCache;

import com.spurnet.model.Block;

import com.spurnet.model.BlockHeader;

import com.spurnet.model.Transaction;

import com.spurnet.storage.BlockStorage;

import com.spurnet.storage.StateStorage;

import com.spurnet.storage.TransactionStorage;

import com.spurnet.storage.rocksdb.BlockStorageImpl;

import com.spurnet.storage.rocksdb.RocksDBManager;

import com.spurnet.storage.rocksdb.StateStorageImpl;

import com.spurnet.storage.rocksdb.TransactionStorageImpl;

import java.util.Arrays;

import java.util.Optional;

public class Main {

public static void main(String[] args) {

try {

// Initialize RocksDB Manager with separate paths for hot and cold data

RocksDBManager dbManager = RocksDBManager.getInstance(

"data/block\_hot.db",

"data/block\_cold.db",

"data/transactions.db",

"data/state.db"

);

// Initialize Cache

BlockCache blockCache = new BlockCache();

// Initialize Storages

BlockStorage blockStorage = new BlockStorageImpl(dbManager, blockCache);

TransactionStorage transactionStorage = new TransactionStorageImpl(dbManager);

StateStorage stateStorage = new StateStorageImpl(dbManager);

// Create a sample transaction

Transaction tx = new Transaction("tx123", "Alice", "Bob", 50.0, System.currentTimeMillis());

// Add transaction to storage

transactionStorage.addTransaction(tx);

System.out.println("Transaction added: " + tx);

// Retrieve transaction

Optional<Transaction> retrievedTx = transactionStorage.getTransaction("tx123");

retrievedTx.ifPresent(System.out::println);

// Create a sample block

BlockHeader header = new BlockHeader("prevHash123", System.currentTimeMillis(), 100, 12345);

Block block = new Block(header, Arrays.asList(tx));

// Save block

blockStorage.saveBlock(block);

System.out.println("Block saved: " + block);

// Retrieve block

String latestHash = blockStorage.getLatestBlockHash();

Optional<Block> retrievedBlock = blockStorage.getBlock(latestHash);

retrievedBlock.ifPresent(System.out::println);

// Save and retrieve state

com.spurnet.model.State state = new com.spurnet.model.State();

state.updateBalance("Alice", 1000.0);

state.updateBalance("Bob", 500.0);

stateStorage.saveState("stateKey1", state);

com.spurnet.model.State retrievedState = stateStorage.getState("stateKey1");

System.out.println("Retrieved State: " + retrievedState);

// Close databases

dbManager.close();

} catch (Exception e) {

e.printStackTrace();

}

}

}

```

## 9. Running the Example

Steps:

#### 1. Create Data Directory:

- In the project root directory, create a `data` folder to store RocksDB database files.

```

spurnet-data-management/

├── pom.xml

├── src

└── data

```

#### 2. Build the Project:

- Use Maven to build the project, ensuring all dependencies are correctly downloaded.

```bash

mvn clean install

```

#### 3. Run the Main Class:

- Run the `Main.java` class. This can be done directly through an IDE (such as IntelliJ IDEA or Eclipse) or using Maven command-line tools.

```bash

mvn exec:java -Dexec.mainClass="com.spurnet.Main"

```

Sample Output:

```

Transaction added: Transaction{txId='tx123', from='Alice', to='Bob', amount=50.0, timestamp=1701234567890}

Transaction{txId='tx123', from='Alice', to='Bob', amount=50.0, timestamp=1701234567890}

Block saved: Block{header=BlockHeader{previousHash='prevHash123', timestamp=1701234567891, nonce=100, merkleRoot=12345}, transactions=[Transaction{txId='tx123', from='Alice', to='Bob', amount=50.0, timestamp=1701234567890}]}

Block{header=BlockHeader{previousHash='prevHash123', timestamp=1701234567891, nonce=100, merkleRoot=12345}, transactions=[Transaction{txId='tx123', from='Alice', to='Bob', amount=50.0, timestamp=1701234567890}]}

Retrieved State: State{accountBalances={Alice=1000.0, Bob=500.0}}

```

## 10. Next Steps

After completing the foundational implementation of the data management module, you can proceed with developing **YouNetWork** as follows:

#### 1. Implement Merkle Trie:

- Integrate the state of accounts and smart contracts into the `MerkleTrie` to support efficient state queries and validation.

- Combine `State` with `MerkleTrie` to implement more complex state management logic.

#### 2. Develop Data Synchronization Mechanism:

- Implement synchronization logic for full nodes and light nodes, including communication protocols with other nodes.

- Handle network communication, concurrency control, and other complex issues to ensure data consistency and integrity.

#### 3. Optimize Storage Performance:

- Data Compression: Adjust RocksDB's compression options based on actual needs, choosing more suitable compression algorithms (e.g., ZSTD).

- Caching Mechanism: Further optimize caching strategies, potentially introducing multi-level caches (e.g., L1, L2 caches).

- Hot and Cold Data Separation: Adjust the criteria for classifying hot and cold data based on block usage frequency and dynamically migrate data as needed.

#### 4. Implement Data Validation:

- Add stricter validation logic when saving and retrieving data to ensure data consistency and integrity.

- Implement hash verification and Merkle proofs to prevent data tampering.

#### 5. Expand Error Handling:

- Enhance exception handling mechanisms to ensure system stability under various failure scenarios.

- Implement logging and monitoring to facilitate troubleshooting and issue resolution.

#### 6. Write Unit Tests and Integration Tests:

- Develop comprehensive test cases for each module to ensure functionality correctness and stability.

- Use JUnit or TestNG for Test-Driven Development (TDD).

#### 7. Documentation:

- Write detailed development documentation and user guides to help team members and developers understand and use the module effectively.

#### 8. Continuous Integration and Deployment (CI/CD):

- Set up automated build and testing pipelines to ensure code quality and functional integrity.

- Implement containerized deployments (e.g., Docker, Kubernetes) to enhance deployment flexibility and scalability.

Additional Recommendations

- Version Control:

- Use Git for version control to ensure code security and traceability.

- Initialize a Git repository in the project root directory and create a `.gitignore` file to exclude the `data` directory and compiled files.

```bash

git init

echo "data/" >> .gitignore

echo "target/" >> .gitignore

git add .

git commit -m "Initial commit of SpurNet data management module"

```

- Code Standards:

- Follow Java coding standards to ensure code readability and consistency.

- Use tools like Checkstyle and SpotBugs for static code analysis to detect potential issues early.

- Logging Management:

- Configure `logback.xml` or other logging framework configuration files in the `src/main/resources/` directory to manage log output levels and formats.

- Use SLF4J for logging in various classes to facilitate issue troubleshooting and monitoring.

Sample `logback.xml`:

```xml

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="INFO">

<appender-ref ref="STDOUT" />

</root>

</configuration>

```