Table of Contents

sundaeswap.routes.test.ts

Introduction:

The `sundaeswap.routes.test.ts` file contains automated tests for the Sundaeswap decentralized exchange (DEX) API routes. These tests validate trade execution, price estimation, and error handling using the Express framework and `supertest` for HTTP request simulation.

Setup and Teardown:

The test environment initializes the necessary instances of `Cardano` and `Sundaeswap` before running the tests:

```
beforeAll(async () => {
  app = express();
  app.use(express.json());
  cardano = Cardano.getInstance('preview');
  await cardano.init();
  sundaeswap = Sundaeswap.getInstance('preview');
  await sundaeswap.init();
  app.use('/amm', AmmRoutes.router);
});

afterEach(() => {
  unpatch();
});

afterAll(async () => {
  await cardano.close();
});
```

Mocking Blockchain Data:

To isolate test cases, various patch functions override blockchain-related methods, such as:

- `patchGetWallet`: Mocks wallet retrieval.

- `patchStoredTokenList`: Provides mock token data.
- `patchExecuteTrade`: Simulates trade execution.

Price Route Tests:

The `POST /amm/price` endpoint is tested for different scenarios:

1. **Valid BUY order**
2. **Valid SELL order**
3. **Unrecognized quote token (expecting 500 error)**
4. **Unrecognized base token (expecting 500 error)**

```
it('should return 200 for BUY', async () => {
  patchGetWallet();
  patchInit();
  patchStoredTokenList();
  patchGetTokenBySymbol();
  patchExecuteTrade();

  await request(app)
    .post(`/amm/price`)
    .send({
      chain: 'cardano',
      network: 'preview',
      connector: 'sundaeswap',
      address: address,
      base: 'SBERRY',
      quote: 'ADA',
      amount: '10000',
      side: 'BUY',
    })
    .set('Accept', 'application/json')
    .expect(200);
});
```

Trade Route Tests:

The `POST /amm/trade` endpoint is tested for:

1. **Successful BUY order execution**
2. **Invalid trade parameters (expecting 404 error)**
3. **Trade failure due to execution issues (expecting 500 error)**

```
it('should return 500 when the executeTrade operation fails', async () => {
  patchGetWallet();
  patchInit();
  patchStoredTokenList();
  patchGetTokenBySymbol();
  patch(sundaeswap, 'executeTrade', () => {
    return 'error';
  });

  await request(app)
    .post(`/amm/trade`)
    .send({
      chain: 'cardano',
      network: 'preview',
      connector: 'sundaeswap',
      base: 'SBERRY',
      quote: 'ADA',
      amount: '1000',
```

```
      address: address,
      side: 'SELL',
    })
    .set('Accept', 'application/json')
    .expect(500);
});
```

Error Handling:

The test suite ensures robust error handling by simulating various failure scenarios, such as:

- Invalid token symbols.
- Incorrect request parameters.
- Trade execution failures.

Sundaeswap.lp.routes.ts

Introduction:

The `sundaeswap.lp.routes.ts` file contains automated tests for the Sundaeswap liquidity pool API routes. These tests validate liquidity price estimation, adding liquidity, and removing liquidity using the Express framework and `supertest` for HTTP request simulation.

Setup and Teardown:

The test environment initializes the necessary instances of `Cardano` and `Sundaeswap` before running the tests:

```
beforeAll(async () => {
  app = express();
  app.use(express.json());
  cardano = Cardano.getInstance('preview');
  await cardano.init();
  sundaeswap = Sundaeswap.getInstance('preview');
  await sundaeswap.init();
  app.use('/amm/liquidity', AmmLiquidityRoutes.router);
});

afterEach(() => {
  unpatch();
});

afterAll(async () => {
  await cardano.close();
});
```

Mocking Blockchain Data:

To isolate test cases, various patch functions override blockchain-related methods, such as:

- `patchGetWallet`: Mocks wallet retrieval.
- `patchStoredTokenList`: Provides mock token data.
- `patchPoolPrice`: Simulates price estimation.

Liquidity Price Route Tests:

The `POST /liquidity/price` endpoint is tested for different scenarios:

1. **Valid request with correct parameters**
2. **Invalid fee tier (expecting 404 error)**

```
it('should return 200 when all parameters are OK', async () => {
  patchForBuy();
  await request(app)
    .post(`/amm/liquidity/price`)
    .send({
      chain: 'cardano',
      network: 'preview',
      connector: 'sundaeswap',
      token0: 'SBERRY',
      token1: 'ADA',
      fee: 'LOW',
      period: 120,
      interval: 60,
    })
    .set('Accept', 'application/json')
    .expect(200);
});
```

Add Liquidity Route Tests:

The `POST /liquidity/add` endpoint is tested for:

1. **Successful liquidity addition**
2. **Unrecognized token symbol (expecting 500 error)**
3. **Invalid fee tier (expecting 404 error)**

```
it('should return 200 when all parameters are OK', async () => {
  patchGetWallet();
  patchInit();
  patchStoredTokenList();
  patchGetTokenBySymbol();

  await request(app)
    .post(`/amm/liquidity/add`)
    .send({
      chain: 'cardano',
      network: 'preview',
      connector: 'sundaeswap',
      address: address,
      token0: 'SBERRY',
      token1: 'ADA',
      amount0: '107043',
      amount1: '10',
      fee: 'LOW',
    })
    .set('Accept', 'application/json')
    .expect(200)
    .expect((res) => {
      expect(res.body.txHash).toBeDefined();
    });
});
```

Remove Liquidity Route Tests:

The `POST /liquidity/remove` endpoint is tested for:

1. **Successful liquidity removal**
2. **Invalid tokenId (expecting 404 error)**

```
it('should return 200 when all parameters are OK', async () => {
  patchForBuy();
  await request(app)
    .post(`/amm/liquidity/remove`)
    .send({
      address: address,
      tokenId: 0,
      chain: 'cardano',
      network: 'preview',
      connector: 'sundaeswap',
      decreasePercent: 50,
    })
    .set('Accept', 'application/json')
    .expect(200)
    .expect((res) => {
      expect(res.body.txHash).toBeDefined();
    });
});
```

Error Handling:

The test suite ensures robust error handling by simulating various failure scenarios, such as:

- Invalid token symbols.
- Incorrect request parameters.
- Liquidity operation failures.