Development and Testing Setup

Overview

This page provides a detailed guide to setting up the local development environment, configuring essential services, and running integration and unit tests. It outlines how developers can emulate a production-like setup locally using Docker Compose and test the service comprehensively.

---

1. Local Development Setup

1.1 Prerequisites

Ensure the following tools are installed on your system:

- **Operating System:** Ubuntu 22.04 or higher.
- **Docker:** Version 20.10 or later.
- **Docker Compose:** Version 1.29 or later.
- **Node.js:** Version 18.x or higher.
- **PostgreSQL:** Version 14.x.

1.2 Clone the Repository

```
bash
```

Copy code

`git clone https://github.com/example/autosync.git cd autosync`

1.3 Configure the Environment Variables

Create a `.env` file in the project root directory with the following details:

```
env
```

Copy code

`NODE_ENV=development DATABASE_URL=postgresql://user:password@localhost:5432/autosync REDIS_URL=redis://localhost:6379 KAFKA_BROKER=kafka://localhost:9092 JWT_SECRET=your_jwt_secret CLOUD_STORAGE_BUCKET=test-bucket`

1.4 Run the Docker Compose Setup

Run all services locally:

```
bash
```

Copy code

`docker-compose up --build`

1.5 Verify Running Services

- API: http://localhost:3000
- PostgreSQL: `localhost:5432`
- Redis: `localhost:6379`
- Kafka: `localhost:9092`

---

2. Local Service Components

2.1 File Sync Service

- Handles file operations and synchronization workflows.
- Endpoints:
  - `POST /api/sync`: Start a sync operation.
  - `GET /api/sync/status`: Check sync status.

2.2 Authentication Service

- Handles user authentication and token management.
- Supported methods:
  - OAuth2.0 (Google, Microsoft).
  - SSO (Keycloak, OpenID Connect).
  - Username/Password.

2.3 Storage Adapter

- Interfaces with local storage (MinIO) or mocks for cloud storage APIs.

---

3. Testing Framework

3.1 Unit Testing

- **Framework:** Jest
- Run unit tests:

  ```
  bash
  ```

  Copy code

  `npm run test:unit`

3.2 Integration Testing

- **Framework:** Supertest and Postman
- Test API endpoints:

  ```
  bash
  ```

  Copy code

  `npm run test:integration`

3.3 End-to-End Testing

- **Framework:** Cypress
- Launch E2E tests:

  ```
  bash
  ```

  Copy code

  `npm run test:e2e`

3.4 Test Coverage Report

Generate and view test coverage:

```
bash
```

Copy code

`npm run test:coverage`

---

4. Debugging and Troubleshooting

4.1 Common Issues

Issue	Solution
Docker container not starting	Check Docker logs: `docker-compose logs <container_name>`
Database connection failed	Verify `.env` configuration and ensure PostgreSQL is running locally.
Service not reachable	Check network bindings and port availability in `docker-compose.yml`.

4.2 Debugging Tips

- Use **VSCode Debugger** with `launch.json` configured for Node.js.
- Attach debugger to the running process using the `--inspect` flag:

  ```
  bash
  ```

  Copy code

  `npm run start:debug`

---

5. Development Workflow

5.1 Git Workflow

- Follow the **GitFlow** branching model:
  - `main`: Production-ready code.
  - `develop`: Latest development changes.
  - `feature/*`: Features under development.
  - `hotfix/*`: Critical fixes for production.

5.2 Code Style

- Linting:

  ```
  bash
  ```

  Copy code

  `npm run lint`

- Prettier formatting:

  ```
  bash
  ```

  Copy code

  `npm run format`

5.3 CI/CD Integration

- GitHub Actions automates:
  - Code quality checks.
  - Build and test processes.

---

6. Local Development Flowchart

(Below is a visual representation of the local setup and testing flow.)