Domain Entities: // Share.java

```java
public class Share {
    private String symbol;
    // Other share attributes and methods
}
// Price.java
public class Price {
    private double value;
    private LocalDate date;
    // Other price attributes and methods
}
```

Interactors: import java.time.LocalDate;

```java
import java.util.Collections;
import java.util.List;
import java.util.logging.Logger;
// FetchSharePriceUseCase.java
public class FetchSharePriceUseCase {
    private static final Logger logger =
Logger.getLogger(FetchSharePriceUseCase.class.getName());
    public List<Price> fetchSharePrice(String symbol, LocalDate startDate, LocalDate endDate) {
        try {
            // Placeholder for actual logic to fetch share price data from Yahoo Finance API
            // Return a list of Price objects representing the share prices within the specified date range
            return Collections.emptyList(); // Placeholder
        } catch (Exception e) {
            logger.severe("Error fetching share prices: " + e.getMessage());
            return Collections.emptyList();
        }
    }
}
// StoreSharePriceUseCase.java
public class StoreSharePriceUseCase {
    private static final Logger logger =
Logger.getLogger(StoreSharePriceUseCase.class.getName());
    public void storeSharePrice(List<Price> prices) {
        try {
            // Placeholder for actual logic to store share price data in the database
        } catch (Exception e) {
            logger.severe("Error storing share prices: " + e.getMessage());
        }
    }
}
```

Data Access Objects (DAOs): import java.sql.*;

```
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;
// SharePriceDAO.java
public class SharePriceDAO {
   private static final Logger logger = Logger.getLogger(SharePriceDAO.class.getName());
   private static final String DATABASE_URL = "jdbc:sqlite:share_prices.db";
   public void save(Price price) {
      try (Connection connection = DriverManager.getConnection(DATABASE_URL);
          PreparedStatement statement = connection.prepareStatement("INSERT INTO prices
(value, date) VALUES (?, ?)")) {
         statement.setDouble(1, price.getValue());
         statement.setDate(2, java.sql.Date.valueOf(price.getDate()));
         statement.executeUpdate();
      } catch (SQLException e) {
         logger.severe("Error saving share price: " + e.getMessage());
      }
   }   public List<Price> getBySymbolAndDateRange(String symbol, LocalDate startDate,
LocalDate endDate) {
      List<Price> prices = new ArrayList<>();
      try (Connection connection = DriverManager.getConnection(DATABASE_URL);
          PreparedStatement statement = connection.prepareStatement("SELECT value, date FROM
prices WHERE date BETWEEN ? AND ?")) {
         statement.setDate(1, java.sql.Date.valueOf(startDate));
         statement.setDate(2, java.sql.Date.valueOf(endDate));
         ResultSet resultSet = statement.executeQuery();
         while (resultSet.next()) {
            double value = resultSet.getDouble("value");
            LocalDate date = resultSet.getDate("date").toLocalDate();
            prices.add(new Price(value, date));
         }
      } catch (SQLException e) {
         logger.severe("Error fetching share prices: " + e.getMessage());
      }
      return prices;
   }
}
```

Main Application Entry Point: import java.time.LocalDate;

```
import java.util.List;
import java.util.logging.Logger;
// MainApplication.java
public class MainApplication {
  private static final Logger logger = Logger.getLogger(MainApplication.class.getName());
  public static void main(String[] args) {
    FetchSharePriceUseCase fetchSharePriceUseCase = new FetchSharePriceUseCase();
    StoreSharePriceUseCase storeSharePriceUseCase = new StoreSharePriceUseCase();
    SharePriceDAO sharePriceDAO = new SharePriceDAO();
    // Example usage: Fetch share price data for symbol "AAPL" between two dates and store it
    LocalDate startDate = LocalDate.of(2022, 1, 1);
    LocalDate endDate = LocalDate.of(2022, 12, 31);
    List<Price> prices = fetchSharePriceUseCase.fetchSharePrice("AAPL", startDate, endDate);
    storeSharePriceUseCase.storeSharePrice(prices);
    // Example usage: Retrieve and display share prices for symbol "AAPL" between two dates
    List<Price> retrievedPrices = sharePriceDAO.getBySymbolAndDateRange("AAPL", startDate,
endDate);
    for (Price price : retrievedPrices) {
      System.out.println("Date: " + price.getDate() + ", Value: " + price.getValue());
    }
  }
}
```

Overview: The Share Price Comparison system helps users track share prices over time. It fetches data from sources like Yahoo Finance and stores it for offline use. The system uses different design concepts to work efficiently.

Issues dealt with;

- Error Handling:

I've noticed that when errors occur in the system, it doesn't provide clear messages on what's happening. This lack of clarity makes it harder for me to understand and fix issues when they arise.

- Database Connections:

I've observed that the way the system handles database connections may not be the most efficient. It opens and closes connections for every operation, which might not be the best approach, especially during busy periods.

Where We Could Improve:

- Better Error Handling:

I think improving how errors are handled in the system is crucial. Instead of vague messages, specific information about what went wrong would make troubleshooting much easier for me as a user.

Test Cases:

- Fetch Share Price Test:

Input: Valid share symbol, valid date range.

Expected Result: Application successfully fetches daily price information for the specified share symbol within the given date range.

- Invalid Share Symbol Test:

Input: Invalid share symbol.

Expected Result: Application displays an error message indicating that the share symbol is invalid.

- Invalid Date Range Test:

Input: Date range exceeding the maximum limit.

Expected Result: Application displays an error message indicating that the date range is invalid.

Conclusion: The Share Price Comparison system has potential but needs some improvements to work better. By addressing the identified issues and implementing the suggested improvements, the system can become more reliable and secure. Testing the system thoroughly will ensure it performs as expected and meets user needs effectively.