Task 3 - Refactoring, Code Metrics, LLM

FeedService

Split the FeedService into multiple classes for separation of concerns - ArticleManager, FaviconManager handle operations for articles and Favicons respectively. Also created 2 classes for separating operations of the synchronize method - FeedSynchronizationManager and FeedSyncProcessor.

Metrics Comparison

Metric Name	Original	Refactored
CBO	12	3
RFC	20	5
WMC	13	6
LOC	268	20
LCAM	0.417	0

Package Metrics

Metric Name	Original	Refactored
WMC	33	38
LOC	360	381

RssReader

- Created an interface FeedParser for parsing input streams
- Created an abstract class BaseParser implementing the interface and inheriting from the parent class of the original RssReaderClass. This serves as base class for different feed formats.
- Created classes AtomFeedParser, RssFeedParser and RdfFeedParser inheriting from BaseParser. The methods for different feeds are now implemented separately, instead of using conditional statements. Also split the Element enum and added its elements as attributes to all classes.

- Created a FeedParsorFactory method for implementing the factory pattern for creating the feeds.

Metrics Comparison

Metric Name	Original	Refactored
CBO	5	4
RFC	13	4
SRFC	2	0
WMC	16	6
LOC	362	52
LCAM	0.656	0.5

Package Metrics

Metric Name	Original	Refactored
WMC	23	19
LOC	479	430

SubscriptionResource

- Created a SubscriptionService class for handling all the data operations. SubscriptionResource now only contains the request-response handling code
- Created FaviconService and ImportExportService classes for separation of concerns

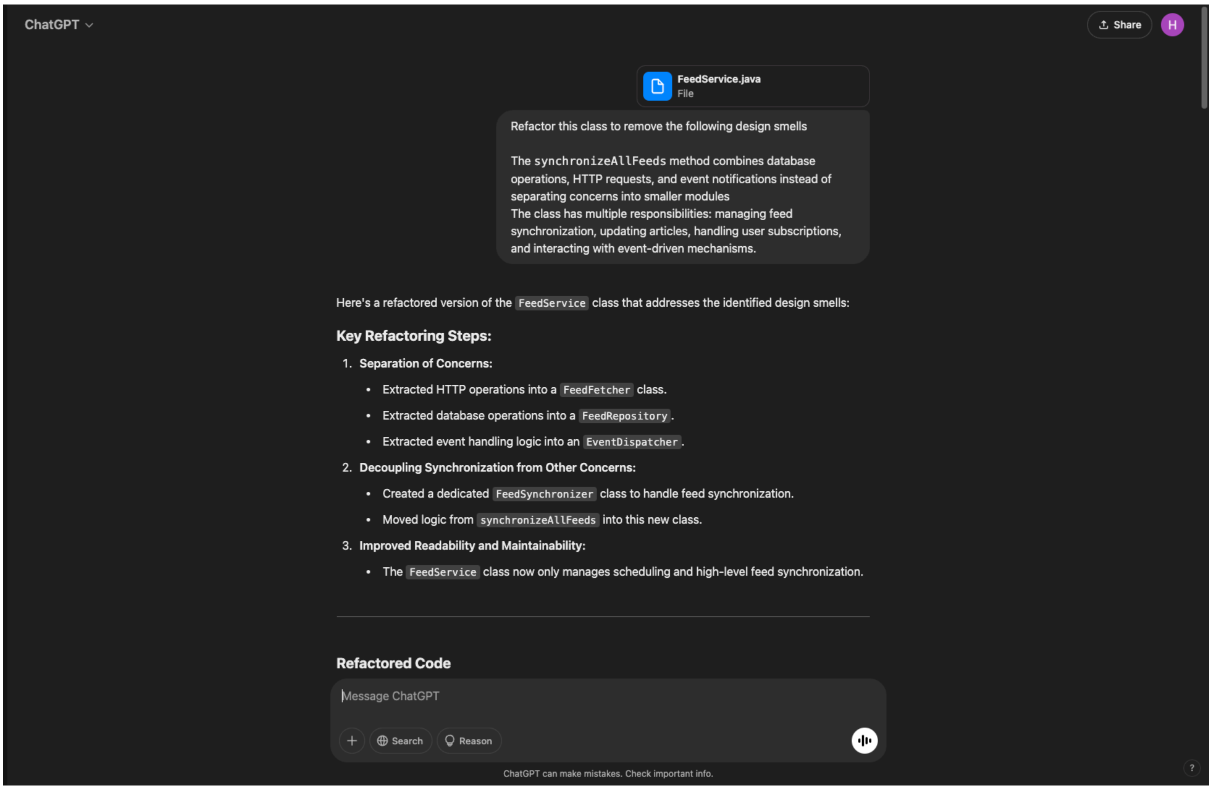
Metrics Comparison

Metric Name	Original	Refactored
CBO	14	7
RFC	17	16
WMC	23	6
LOC	365	108
LCAM	0.444	0.533

Metric Name	Original	Refactored

WMC	73	63
LOC	1333	1368

Using LLM for Refactoring



Refactor this class to remove the following design smells

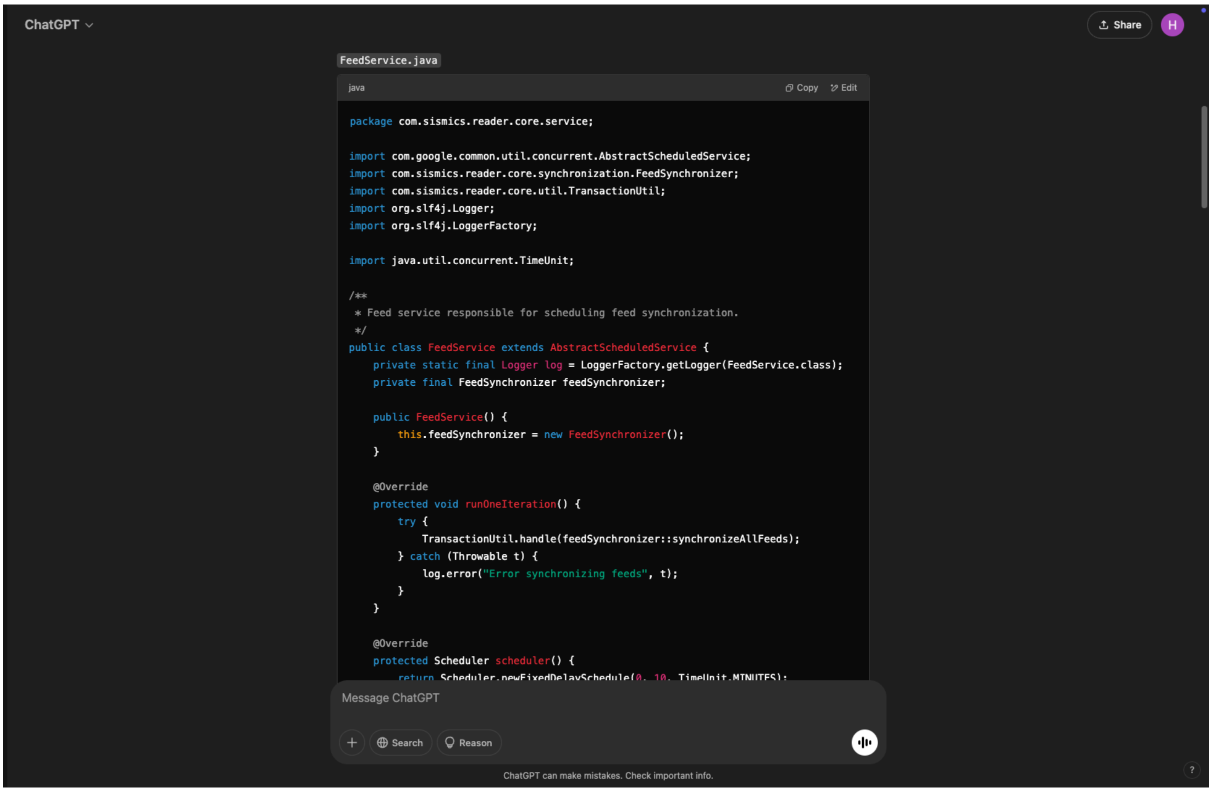
The `synchronizeAllFeeds` method combines database operations, HTTP requests, and event notifications instead of separating concerns into smaller modules
The class has multiple responsibilities: managing feed synchronization, updating articles, handling user subscriptions, and interacting with event-driven mechanisms.

Here's a refactored version of the `FeedService` class that addresses the identified design smells:

Key Refactoring Steps:

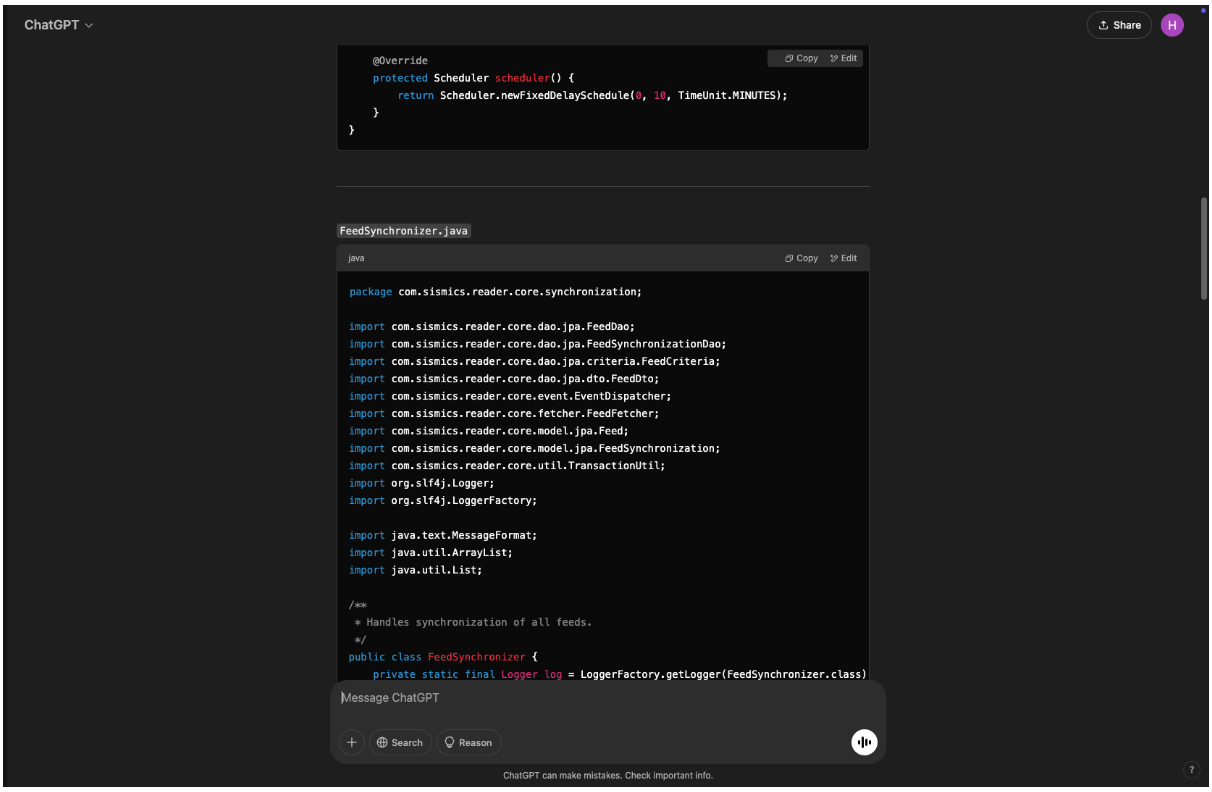
1. **Separation of Concerns:**
   - Extracted HTTP operations into a `FeedFetcher` class.
   - Extracted database operations into a `FeedRepository`.
   - Extracted event handling logic into an `EventDispatcher`.
2. **Decoupling Synchronization from Other Concerns:**
   - Created a dedicated `FeedSynchronizer` class to handle feed synchronization.
   - Moved logic from `synchronizeAllFeeds` into this new class.
3. **Improved Readability and Maintainability:**
   - The `FeedService` class now only manages scheduling and high-level feed synchronization.

Refactored Code



`FeedService.java`

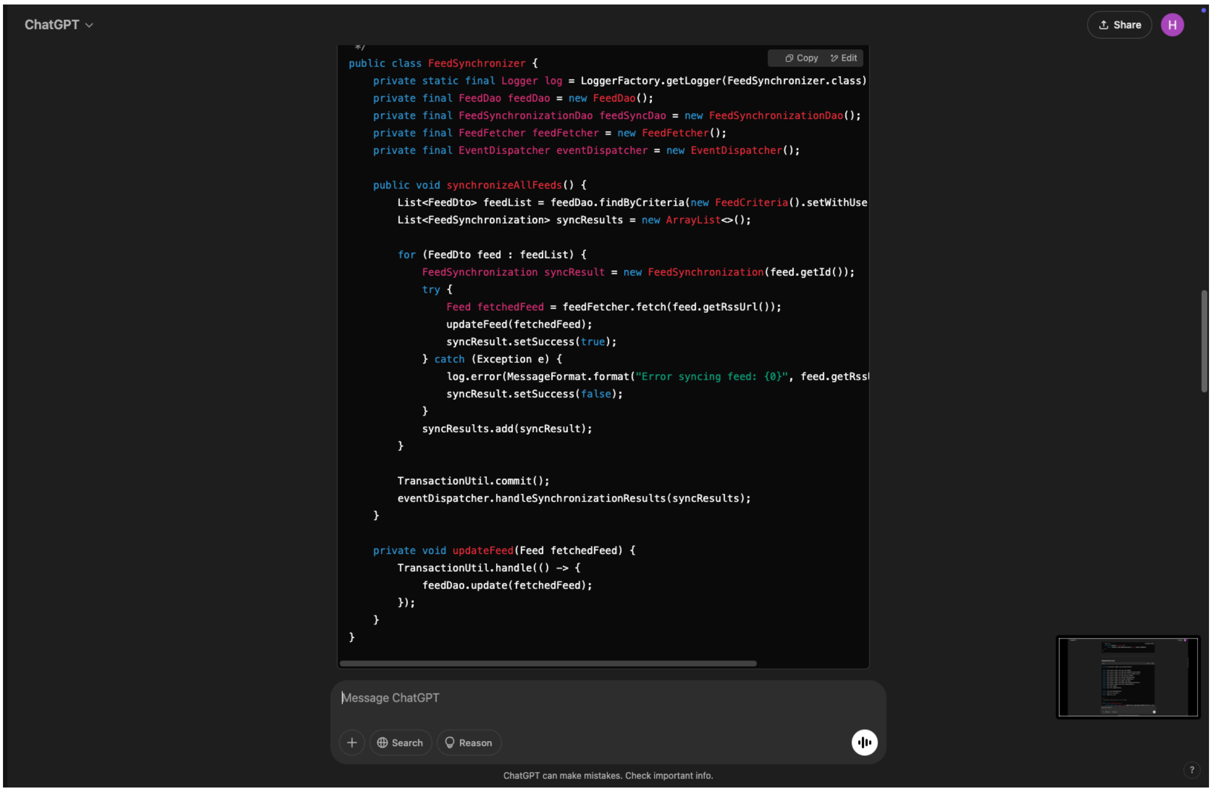
```java
package com.sismics.reader.core.service;

import com.google.common.util.concurrent.AbstractScheduledService;
import com.sismics.reader.core.synchronization.FeedSynchronizer;
import com.sismics.reader.core.util.TransactionUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.concurrent.TimeUnit;

/**
 * Feed service responsible for scheduling feed synchronization.
 */
public class FeedService extends AbstractScheduledService {
    private static final Logger log = LoggerFactory.getLogger(FeedService.class);
    private final FeedSynchronizer feedSynchronizer;

    public FeedService() {
        this.feedSynchronizer = new FeedSynchronizer();
    }

    @Override
    protected void runOneIteration() {
        try {
            TransactionUtil.handle(feedSynchronizer::synchronizeAllFeeds);
        } catch (Throwable t) {
            log.error("Error synchronizing feeds", t);
        }
    }

    @Override
    protected Scheduler scheduler() {
        return Scheduler.newFixedDelaySchedule(0, 10, TimeUnit.MINUTES);
```



```java
    @Override
    protected Scheduler scheduler() {
        return Scheduler.newFixedDelaySchedule(0, 10, TimeUnit.MINUTES);
    }
}
```

`FeedSynchronizer.java`

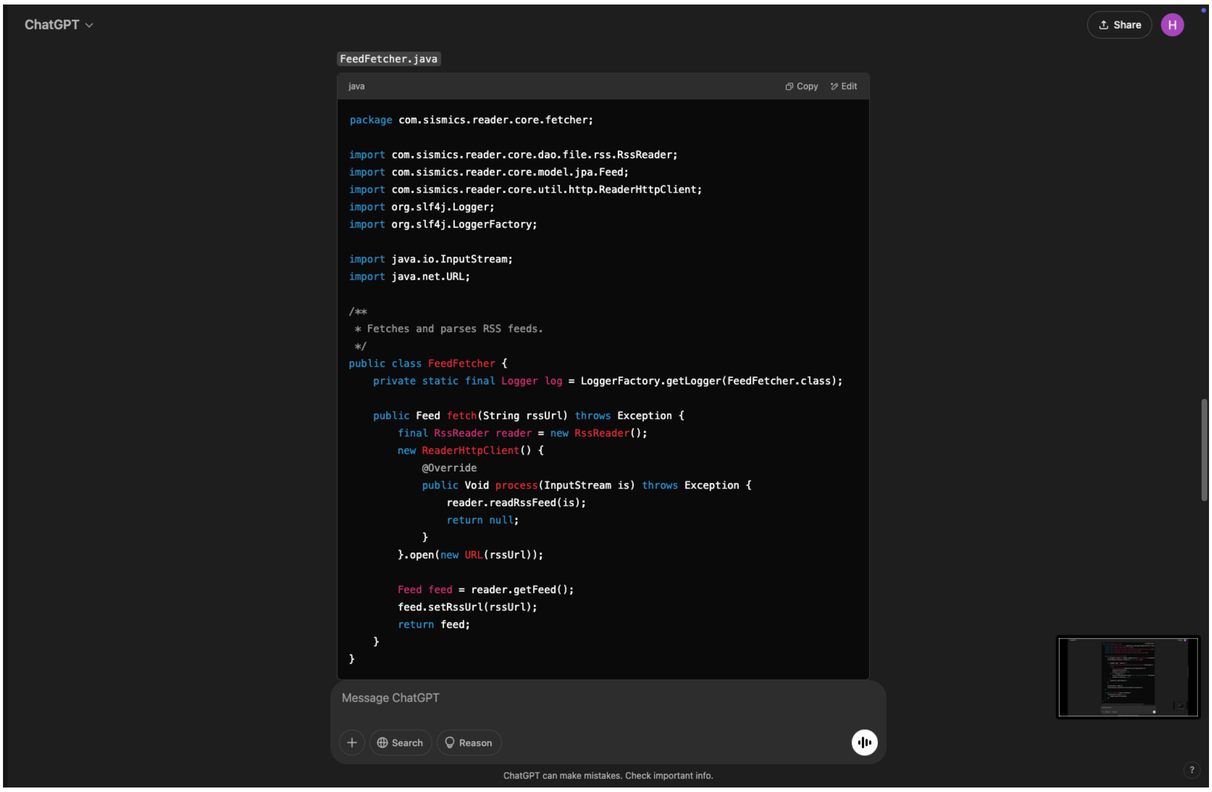
```java
package com.sismics.reader.core.synchronization;

import com.sismics.reader.core.dao.jpa.FeedDao;
import com.sismics.reader.core.dao.jpa.FeedSynchronizationDao;
import com.sismics.reader.core.dao.jpa.criteria.FeedCriteria;
import com.sismics.reader.core.dao.jpa.dto.FeedDto;
import com.sismics.reader.core.event.EventDispatcher;
import com.sismics.reader.core.fetcher.FeedFetcher;
import com.sismics.reader.core.model.jpa.Feed;
import com.sismics.reader.core.model.jpa.FeedSynchronization;
import com.sismics.reader.core.util.TransactionUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.List;

/**
 * Handles synchronization of all feeds.
 */
public class FeedSynchronizer {
    private static final Logger log = LoggerFactory.getLogger(FeedSynchronizer.class)
```



```java
 */
public class FeedSynchronizer {
    private static final Logger log = LoggerFactory.getLogger(FeedSynchronizer.class)
    private final FeedDao feedDao = new FeedDao();
    private final FeedSynchronizationDao feedSyncDao = new FeedSynchronizationDao();
    private final FeedFetcher feedFetcher = new FeedFetcher();
    private final EventDispatcher eventDispatcher = new EventDispatcher();

    public void synchronizeAllFeeds() {
        List<FeedDto> feedList = feedDao.findByCriteria(new FeedCriteria().setWithUse
        List<FeedSynchronization> syncResults = new ArrayList<>();

        for (FeedDto feed : feedList) {
            FeedSynchronization syncResult = new FeedSynchronization(feed.getId());
            try {
                Feed fetchedFeed = feedFetcher.fetch(feed.getRssUrl());
                updateFeed(fetchedFeed);
                syncResult.setSuccess(true);
            } catch (Exception e) {
                log.error(MessageFormat.format("Error syncing feed: {0}", feed.getRss
                syncResult.setSuccess(false);
            }
            syncResults.add(syncResult);
        }

        TransactionUtil.commit();
        eventDispatcher.handleSynchronizationResults(syncResults);
    }

    private void updateFeed(Feed fetchedFeed) {
        TransactionUtil.handle(() -> {
            feedDao.update(fetchedFeed);
        });
    }
}
```

Message ChatGPT

ChatGPT can make mistakes. Check important info.

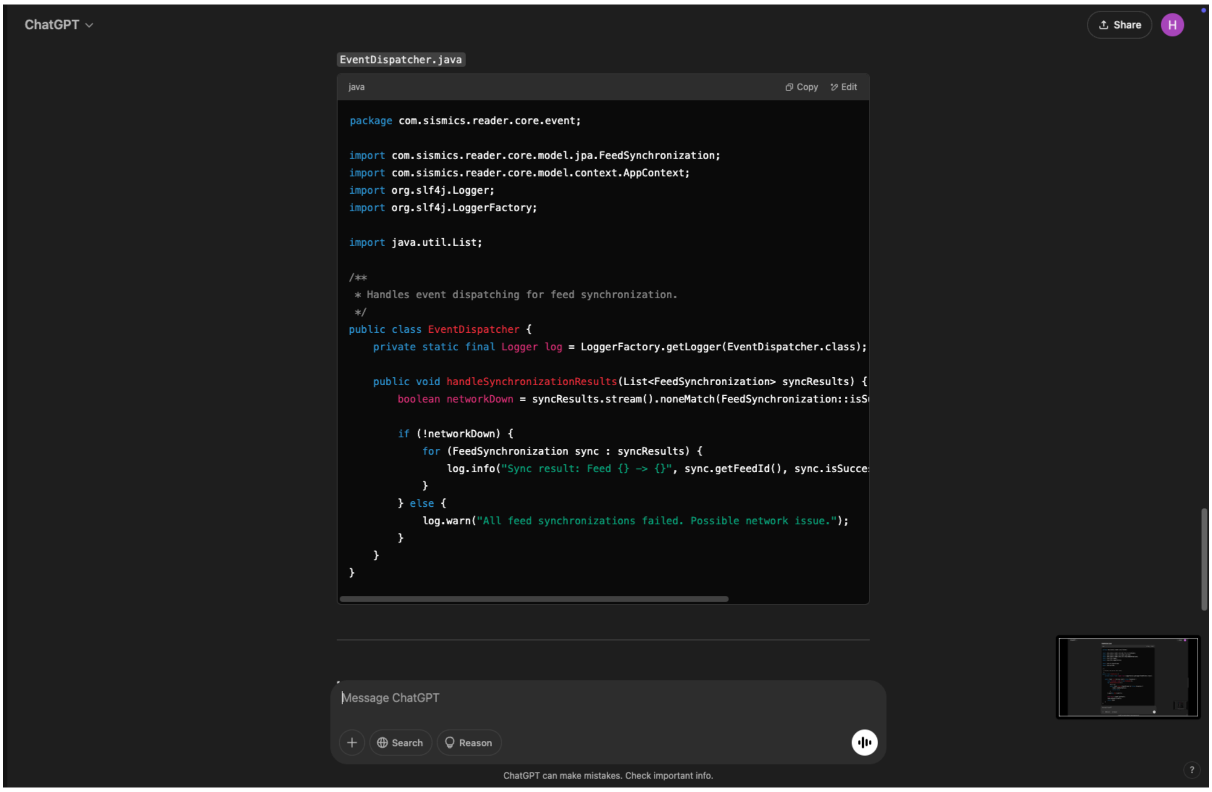


`FeedFetcher.java`

```java
package com.sismics.reader.core.fetcher;

import com.sismics.reader.core.dao.file.rss.RssReader;
import com.sismics.reader.core.model.jpa.Feed;
import com.sismics.reader.core.util.http.ReaderHttpClient;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.InputStream;
import java.net.URL;

/**
 * Fetches and parses RSS feeds.
 */
public class FeedFetcher {
    private static final Logger log = LoggerFactory.getLogger(FeedFetcher.class);

    public Feed fetch(String rssUrl) throws Exception {
        final RssReader reader = new RssReader();
        new ReaderHttpClient() {
            @Override
            public Void process(InputStream is) throws Exception {
                reader.readRssFeed(is);
                return null;
            }
        }.open(new URL(rssUrl));

        Feed feed = reader.getFeed();
        feed.setRssUrl(rssUrl);
        return feed;
    }
}
```

Message ChatGPT

ChatGPT can make mistakes. Check important info.



```java
package com.sismics.reader.core.event;

import com.sismics.reader.core.model.jpa.FeedSynchronization;
import com.sismics.reader.core.model.context.AppContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.List;

/**
 * Handles event dispatching for feed synchronization.
 */
public class EventDispatcher {
    private static final Logger log = LoggerFactory.getLogger(EventDispatcher.class);

    public void handleSynchronizationResults(List<FeedSynchronization> syncResults) {
        boolean networkDown = syncResults.stream().noneMatch(FeedSynchronization::isS

        if (!networkDown) {
            for (FeedSynchronization sync : syncResults) {
                log.info("Sync result: Feed {} -> {}", sync.getFeedId(), sync.isSucce
            }
        } else {
            log.warn("All feed synchronizations failed. Possible network issue.");
        }
    }
}
```

The code given by chatgpt does not contain a lot of methods contained in the original class, like the article related methods and favicon related methods. The code for synchronize method is also incomplete

LLM Refactoring automated pipeline (3d + bonus)

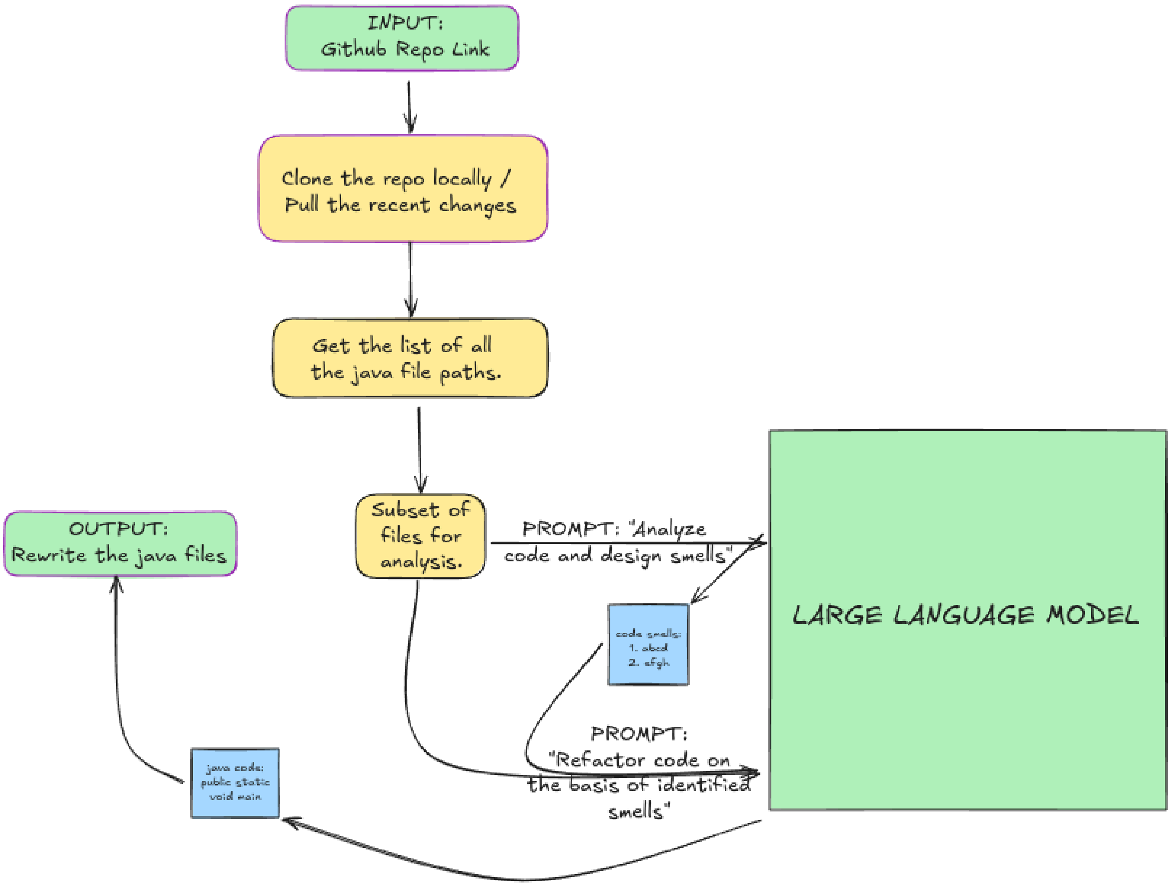
Our primary model used was Gemini.

Given: GitHub public repository link

Pipeline:

1. Select the LLM.
2. Extract any two Java files from the repository.
3. Prompt the LLM to detect code and design smells in each file, along with the corresponding line numbers.
4. Request the LLM to refactor the entire file based on the issues identified in the previous step.

5. Replace the original file with the refactored version and commit the changes to the repository.
6. Create a pull request from a new branch.



Bonus

Models used:

1. Gemini
2. Llama

Code and design smells Analysis

Llama

- Provides a structured and detailed list of design smells with precise line numbers.

- Covers a broad range of traditional issues, such as *Long Method, Duplicate Code, Feature Envy, and Message Chains.*
- Lacks high-level architectural insights but excels at pinpointing localized issues in the code.

Gemini

- Identifies broader architectural problems like *Shotgun Surgery, God Class, and Divergent Change.*
- Offers conceptual insights into maintainability challenges but is less precise with line numbers.
- Some issues overlap under different categories, making the output less structured.

Code Refactoring Analysis

Llama

- Produces a significantly altered structure, often changing the entire testing approach.
- Omits crucial elements from the original code, such as database interactions and assertion methods.
- Introduces inconsistencies with missing implementations for key functions like `getResponse()`.
- May not preserve original functionality.

Gemini

- Retains the original structure while improving readability and maintainability.
- Preserves database interactions and assertions, ensuring functional correctness.
- Maintains overall fidelity to the original intent of the code.

Remaining Code/Design Smells

- **Llama's refactoring** introduces new issues, such as *Incomplete Implementation* and *Loss of Functionality.* It seems to be too aggressive.
- **Gemini's refactoring** still has some *Long Methods* and *Duplicate Code,* though it improves structure without breaking functionality. It seems to be

more reliable.