# Project Phase -1

- 
- 

## Team-5


- Arth Raj (2019101094)
- Ashish Gupta (2019101061)
- Isha Gupta (2019101111)
- Soma Datta Reddy (2019111008)
- Vaibhav Kashera (2019111003)


---

## Task 1: Mining the Repository

### ▼ User Mangement:

There are a lot of classes that make user management possible, but the essential classes that help to understand the working of the app are:

1. UserDao: This class essentially has all the functions needed to interact with the database.
2. User: This class basically defines the User entity.
3. UserMapper: This class essentially contains a single function that basically maps the data returned from a database query to the defined “User” object.
4. UserDto: This class defines a UserDto entity that contains information of the user (such as id, username, email etc) and getter and setter functions for some things like id, username, email etc.
5. UserDtoMapper: This class essentially contains a single function that basically maps the data returned from a database query to the defined “UserDto” object.
6. UserCriteria: This class lets us know if a user is registered on Last.fm
7. UserResource: This class helps in handling User REST resources.
8. AuthenticationToken: This class creates a “AuthenticationToken” object that helps keep track of user sessions.
9. AuthenticationTokenDao: This class essentially has all the functions needed to interact with the database.

There are a lot of functions related to user management but the essential ones are:

1. **In User class:**
   a. User(String id, String localeId, String roleId, String username, String password, String email, Integer maxBitrate, String lastFmSessionToken, boolean lastFmActive, boolean firstConnection, Date createDate, Date deleteDate): This function takes in 12 parameters and creates a new instance of a 'User' class.
2. **In UserDao class:**
   a. create(User user): Takes in a User object and creates an account for that user and returns the user's Id.
   b. update(User user): This functions is used to update any information of the user.
   c. updatePassword(User user): This function takes in a User object whose password is to be updated, hashes the new password and returns the updated User object.
   d. updateLastFmSessionToken(User user): Updates the user Last.fm session token.
   e. getActiveById(String id): Takes in an id and returns the user corresponding to that Id.
   f. getActiveByUsername(String username): Takes in a username and returns the user corresponding to that username.
   g. getActiveByPasswordResetKey(String passwordResetKey): Takes in a passwordResetKey and returns the user corresponding to that passwordResetKey.
   h. delete(String username): Deletes a users account with the given username.
   i. hashPassword(String password): Takes in a clear password and returns a hashed password.
3. **In AuthenticationToken class:**
   a. AuthenticationToken(String id, String userId, boolean longLasted, Date createDate, Date lastConnectionDate): creates a new instance of an 'AuthenticationToken' class.
4. **In AuthenticationTokenDao class:**
   a. get(String id): gets an auth token by its id.

b. create(AuthenticationToken authenticationToken): creates a new auth token and returns its id.

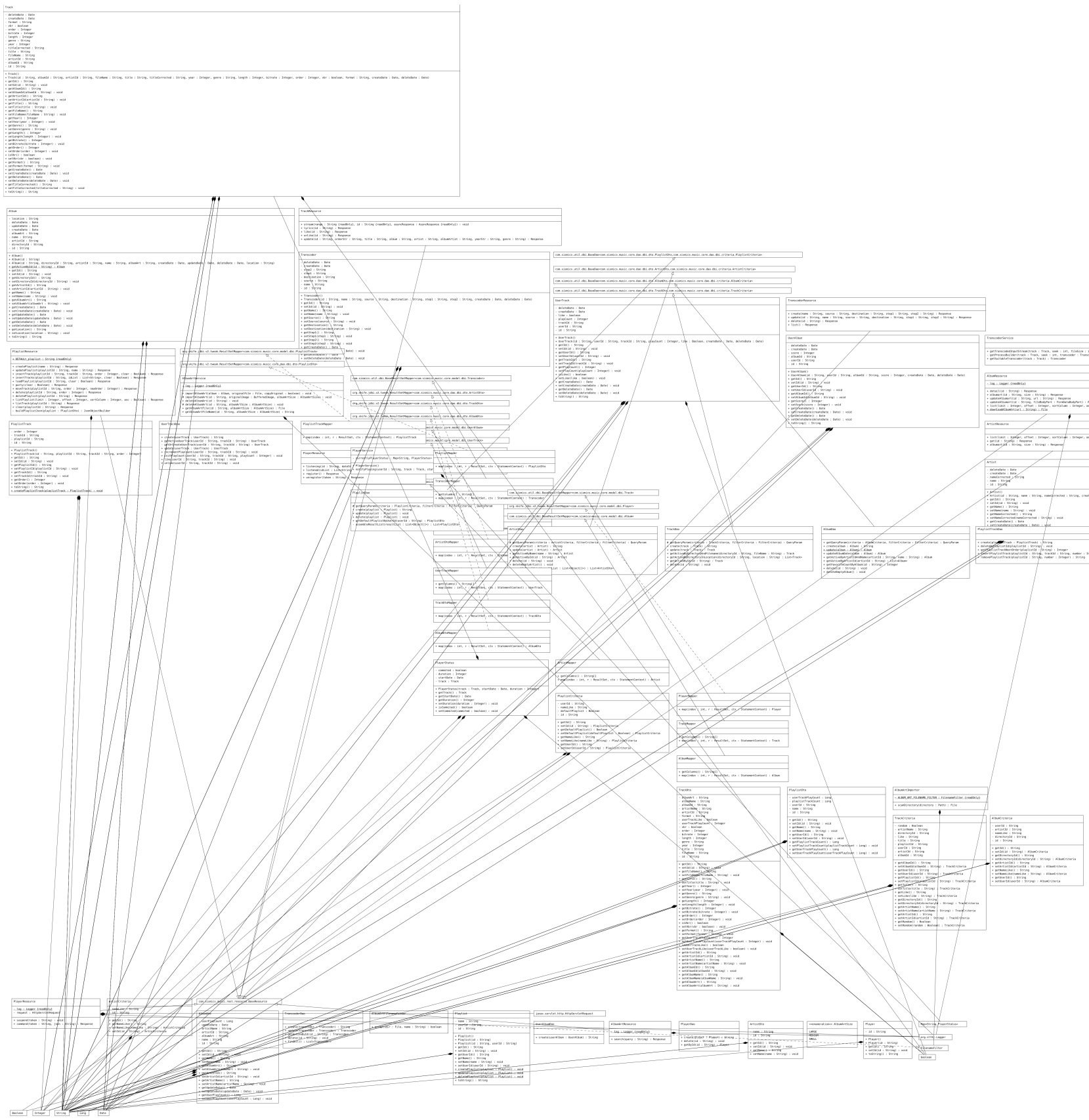
c. delete(String authenticationTokenId): deletes an existing auth token

Note: the other classes ( i.e. UserMapper, UserDto, UserDtoMapper and UserCriteria) have very limited functions whose functionalities we have already discussed while describing these classes itself.

**Library Management:**

The essential classes that make library management possible are:

- Album, AlbumDao, AlbumCriteria, AlbumDto, AlbumDtoMapper, AlbumMapper
- UserAlbum, UserAlbumDao
- UserTrack, UserTrackDao, UserTrackMapper
- Artist, ArtistDao, ArtistMapper, ArtistDtoMapper, ArtistDto, ArtistCriteria
- Player, PlayerDao
- Playlist, PlaylistDao, PlaylistMapper, PlaylistDto, PlaylistCriteria
- PlaylistTrack, PlaylistTrackDao
- Track, TrackDao, TrackCriteria, TrackDto, TrackdtoMapper, TrackMapper
- Transcoder, TranscoderDao, TranscoderMapper



As you can see the above classes, there is a pattern.

The classes ending with ‘Dao’ (i.e., Data Access Object, a design pattern in java) contain functions for interacting with the database for that given entity (i.e. for example, AlbumDao will have functions to perform CRUD for Album object).

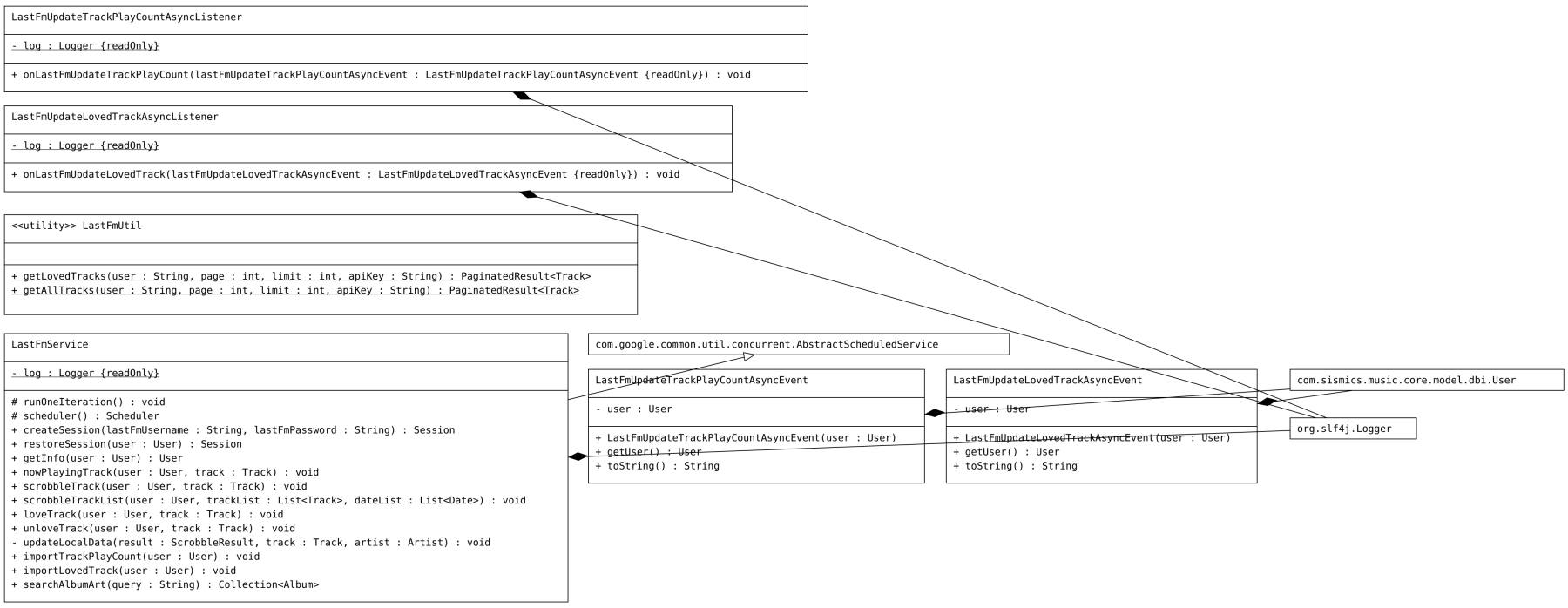
The classes ending with ‘Dto’ are used to define a ‘Data Transfer Object’ that helps to transfer data between different layers.

The classes ending with ‘mapper’ are essentially for mapping the data returned by a database query to the given object/model.

The other classes which end with none of the above mentioned words nor ‘criteria’ contain a constructor method to create the given entity (i.e., Album class has a constructor to create Album entity) and some getter and setter functions.

**Last.fm Integration:**

The functionalities required for the integration of the Last.fm are contained in the class LastFmService.



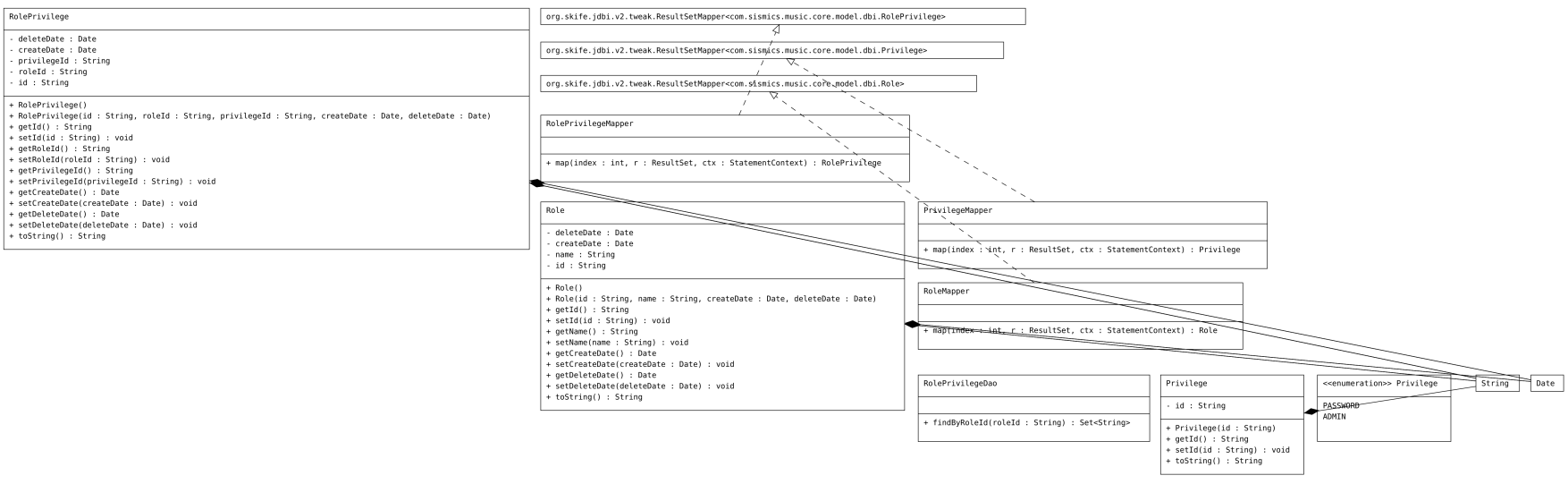

The functions present in this class are:

1. createSession(String lastFmUsername, String lastFmPassword): This function is responsible for creating a new user session. It takes in the Last.fm credentials of the user and returns a Last.fm session.
2. restoreSession(User user): This function restores the session from the stored session token. It returns a Last.fm session.
3. getInfo(User user): This function returns the Last.fm information of a user.
4. nowPlayingTrack(User user, Track track): This function updates the currently playing track of a given user.
5. scrobbleTrack(User user, Track track): This function takes in a track to scrobble and a user object and scrobbles the given track for the user.
6. scrobbleTrackList(User user, List<Track> trackList, List<Date> dateList): This function takes in a tracklist to scrobble and a user object and scrobbles the given tracklist for the user.
7. loveTrack(User user, Track track): This function allows the given user to 'love' a track.
8. unloveTrack(User user, Track track): This function allows the given user to 'unlove' a track that the user had previously loved.
9. updateLocalData(ScrobbleResult result, Track track, Artist artist): This function allows an artist to update local data from Last.fm corrected labels.
10. importTrackPlayCount(User user): Given a User, this function imports all track play counts from Last.fm.
11. importLovedTrack(User user): Given a User, this function imports all loved tracks from Last.fm.
12. searchAlbumArt(String query): Given a search query, this function searches for the related album art and returns the albums.

**Administrator features:**

There are a lot of classes that make administrator features possible, but the essential classes that help to understand the working of the app are:

1. Role
2. RoleMapper
3. RolePrivilege
4. RolePrivilegeDao
5. RolePrivilegeMapper
6. Privilege
7. PrivilegeMapper




The functions that are crucial for understanding the working of this feature are:

1. **In RolePrivilege class:**
   a. RolePrivilege(String id, String roleId, String privilegeId, Date createDate, Date deleteDate): this constructor helps create the RolePrivilege object.
2. **In RolePrivilegeDao class:**
   a. findByRoleId(String roleId): This function helps find the set of privileges of a role.
3. **In Role class:**
   a. Role(String id, String name, Date createDate, Date deleteDate): this constructor helps to create instances of the Role object.
4. **In Privilege class:**
   a. Privilege(String id): This constructor helps create an instance of the Privilege entity.

Note: All the Mapper classes mentioned contain a single function map(int index, ResultSet r, StatementContext ctx): This function maps the result set from a database query to the given object.

---

# Task 2- Analysis

- Sonarqube, based on Java 17, was used for analysis.
- A total of 12 bugs and 237 code smells were detected, including all types

## Task 2a: Identification of design smells based on code smells

1. **Insufficient Modularization**

   - `com.sismics.util.mime`: `MimeType` is an example of deficient encapsulation and insufficient modularization. It is supported by high cyclomatic complexity as shown in the code metrics.

```
public static String guessMimeType(File file) throws Exception {
        InputStream is = null;
        try {
            is = new FileInputStream(file);
            byte[] headerBytes = new byte[64];
            int readCount = is.read(headerBytes, 0, headerBytes.length);
            if (readCount <= 0) {
                throw new Exception("Cannot read input file");
            }
            String header = new String(headerBytes, "US-ASCII");

            if (header.startsWith("PK")) {
                return MimeType.APPLICATION_ZIP;
            } else if (header.startsWith("GIF87a") || header.startsWith("GIF89a")) {
                return MimeType.IMAGE_GIF;
            } else if (headerBytes[0] == ((byte) 0xff) && headerBytes[1] == ((byte) 0xd8)) {
                return MimeType.IMAGE_JPEG;
            } else if (headerBytes[0] == ((byte) 0x89) && headerBytes[1] == ((byte) 0x50) && headerBytes[2]
== ((byte) 0x4e) && headerBytes[3] == ((byte) 0x47) &&
                    headerBytes[4] == ((byte) 0x0d) && headerBytes[5] == ((byte) 0x0a) && headerBytes[6] ==
((byte) 0x1a) && headerBytes[7] == ((byte) 0x0a)) {
                return MimeType.IMAGE_PNG;
            } else if (headerBytes[0] == ((byte) 0x00) && headerBytes[1] == ((byte) 0x00) && headerBytes[2]
== ((byte) 0x01) && headerBytes[3] == ((byte) 0x00)) {
                return MimeType.IMAGE_X_ICON;
            }
        } finally {
            if (is != null) {
                is.close();
            }
        }
        return null;
    }
```

2. **Unnecessary abstraction**

   - `/com/sismics/music/core/service/importaudio/ImportAudio.java`: enum is implicitly static in a class, still static keyword is used. The "static" keyword is used to define class-level variables and methods that can be accessed without creating an instance of the class. Since enums are a special kind of class in Java, they are inherently static and their values (i.e., the enum constants) are static final variables. We can access enum constants directly without instantiating the enum class.

3. **Insufficient Modularization**

   - ImportAudioService.java has void run() method which shows insufficient modularization and can also be seen as a violation of single responsibility principle. Indicated by Long Method code smell.

```
protected void run() throws Exception {
        while (isRunning()) {
            // Wait for a new URL to import
            ImportAudio importAudio = importAudioQueue.take();
```

```
            try {
                log.info("Start importing a new URL: " + importAudio);

                // Starting YouTube-DL
                String output = DirectoryUtil.getImportAudioDirectory().getAbsolutePath() + File.separator
                String command = "youtube-dl -v --prefer-ffmpeg --encoding UTF-8 --newline -f bestaudio -x
                        + " --audio-quality " + importAudio.getQuality() + " -o";
                List<String> commandList = new LinkedList<>(Arrays.asList(StringUtils.split(command)));
                commandList.add(output);
                commandList.add(importAudio.getUrl());
                Process process = new ProcessBuilder(commandList)
                        .redirectErrorStream(true)
                        .start();

                // Import status is now in progress
                importAudio.setStatus(ImportAudio.Status.INPROGRESS);
                importAudio.setProcess(process);
                importAudioList.add(importAudio);

                // Reading standard output to update import status
                try (BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStre
                    for (String line = reader.readLine(); line != null && isProcessRunning(process); line
                        // Update progression
                        Matcher matcher = PROGRESS_PATTERN.matcher(line);
                        if (matcher.find()) {
                            importAudio.setProgress(Float.valueOf(matcher.group(1)));
                            importAudio.setTotalSize(matcher.group(2));
                            importAudio.setDownloadSpeed(matcher.group(3));
                        }

                        // Check if the line is an error
                        if (line.contains("ERROR")) {
                            importAudio.setStatus(ImportAudio.Status.ERROR);
                        }

                        // New working file
                        matcher = WORKING_FILE_PATTERN.matcher(line);
                        if (matcher.find()) {
                            File file = new File(matcher.group(1));
                            String name = file.getName();
                            if (!Strings.isNullOrEmpty(name)) {
                                importAudio.addWorkingFiles(name);
                            }
                        }

                        // Debug output
                        importAudio.setMessage(importAudio.getMessage() + "\n" + line);
                    }
                }

                // The process has not been terminated properly
                if (process.waitFor() != 0) {
                    importAudio.setMessage(importAudio.getMessage() + "\nProcess exit with code: " + proce
                    importAudio.setStatus(ImportAudio.Status.ERROR);
                }

                // Import is done
                if (importAudio.getStatus() != ImportAudio.Status.ERROR) {
                    importAudio.setStatus(ImportAudio.Status.DONE);
                } else {
                    // Remove working files if they still exist
                    for (String name : importAudio.getWorkingFiles()) {
                        File file = new File(DirectoryUtil.getImportAudioDirectory().getAbsolutePath() + F
                        if (file.exists()) {
                            file.delete();
                        }
                    }
                }

                process.destroy();
                importAudio.setProcess(null);
            } catch (Exception e) {
```

```
                // Error while importing
                importAudio.setStatus(ImportAudio.Status.ERROR);
                importAudio.setMessage(importAudio.getMessage() + "\n" + e.getMessage());
                log.error("Error importing: " + importAudio, e);
            }
        }
    }
```

- The design smell involved here is a violation of single responsibility principle `static int consumeQuote(final CharSequence s, final int start)` method in class SQLsplitter. Code smell is long method.

```
static int consumeQuote(final CharSequence s, final int start) {
        if (start >= s.length()) return start;
        char ender;
        switch (s.charAt(start)) {
            case '\'':
                ender = '\'';
                break;
            case '"':
                ender = '"';
                break;
            case '[':
                ender = ']';
                break;

            case '`':
                ender = '`';
                break;
            case '$': {
                int quoteEnd = start + 1;
                for (; s.charAt(quoteEnd) != '$'; ++quoteEnd)
                    if (quoteEnd >= s.length())
                        return quoteEnd;
                int i = quoteEnd + 1;
                while (i < s.length()) {
                    if (s.charAt(i) == '$') {
                        boolean match = true;
                        for (int j = start; j <= quoteEnd && i < s.length(); ++j, ++i) {
                            if (s.charAt(i) != s.charAt(j)) {
                                match = false;
                                break;
                            }
                        }
                        if (match)
                            return i;
                    } else
                        ++i;
                }
                return i;
            }

            default:
                return start;
        }

        boolean escaped = false;

        for (int i = start + 1; i < s.length(); ++i) {
            if (escaped) {
                escaped = false;
                continue;
            }
            char c = s.charAt(i);
            if (c == '\\')
                escaped = true;
            else if (c == ender)
                return i + 1;
        }
```

```
        return s.length();
    }
```

4. **Missing Abstraction**

   Lots of classes like `User` , `track` , `Artist`, `Album` have common properties like createDate, deleteDate, updateDate(just for Album) and their two corresponding getter and setter functions. Since these are common across numerous classes as identified, they can be created as a separate class altogether.

```
// LOC for constructors in different classes
this.createDate = createDate;
this.updateDate = updateDate;
this.deleteDate = deleteDate;
```

5. **Dead code:**

   - /com/sismics/music/core/dao/dbi/PlaylistDao.java

```
private List<PlaylistDto> assembleResultList(List<Object[]> resultList) {
        List<PlaylistDto> playlistDtoList = new ArrayList<>();
        for (Object[] o : resultList) {
            int i = 0;
            PlaylistDto playlistDto = new PlaylistDto();
            playlistDto.setId((String) o[i++]);
            playlistDto.setName((String) o[i++]);
            playlistDto.setUserId((String) o[i]);
            playlistDtoList.add(playlistDto);
        }
        return playlistDtoList;
    }
```

   - src/.../com/sismics/music/core/dao/dbi/ArtistDao.java

```
private List<ArtistDto> assembleResultList(List<Object[]> resultList) {
        List<ArtistDto> artistDtoList = new ArrayList<>();
        for (Object[] o : resultList) {
            int i = 0;
            ArtistDto artistDto = new ArtistDto();
            artistDto.setId((String) o[i++]);
            artistDto.setName((String) o[i++]);
            artistDtoList.add(artistDto);
        }
        return artistDtoList;
    }
```

   Here private method assembleResultList is not being used in both classes, an example of duplicated code and dead code.

   - CollectionService.java has two overridden functions startUp() and startDown() which do not have any function body in them.
   - public void updateScore() in CollectionService.java has a similar problem. Although it is being called from other classes, it does nothing and impacts the codebase negatively when we consider parameters like compile times and code quality.

6. **Primitive Obsession**

While primitive obsession is a code smell, here usage of an excessive number of parameters in a method makes it more difficult to understand and maintain, hence affecting the code quality.

- `public Track(String id, String albumId, String artistId, String fileName, String title, String titleCorrected, Integer year, String genre, Integer length, Integer bitrate, Integer order, boolean vbr, String format, Date createDate, Date deleteDate)` (src/.../com/sismics/music/core/model/dbi/Track.java)

## Task 2b: Code Metrics

Code Metrics for the project have been determined using tools **Checkstyle, Designite, CodeMR.**

Detailed generated code metrics report by Designite is in `docs` folder. It lists out several metrics. Important ones are listed below.

1. **Parameter Count (PC):**

   It refers to the number of parameters (inputs) that a function or method has in a software program. It is a quantitative measure of the complexity of the function or method.

   Implications: Functions with large number of parameters can be difficult to use and comprehend. They are often indicative of poor design, and may be a sign that the functionality needs to be broken down into smaller, more manageable parts.

| Project Name | Package Name | Type Name | MethodName | LOC | CC | PC |
|---|---|---|---|---|---|---|
| music-core | com.sismics.music.core.model.dbi | Track | Track | 17 | 1 | 15 |
| music-core | com.sismics.music.core.model.dbi | User | User | 14 | 1 | 12 |
| music-core | com.sismics.music.core.model.dbi | Album | Album | 11 | 1 | 9 |
| music-core | com.sismics.music.core.model.dbi | Transcoder | Transcoder | 10 | 1 | 8 |
| music-core | com.sismics.music.core.util | ImageUtil | resizeImage | 23 | 6 | 7 |
| music-core | com.sismics.music.core.service.importaudio | ImportAudioService | tagFile | 37 | 2 | 7 |
| music-core | com.sismics.music.core.model.dbi | UserTrack | UserTrack | 9 | 1 | 7 |
| music-core | com.sismics.music.core.util.dbi | QueryParam | QueryParam | 9 | 1 | 7 |
| music-core | com.sismics.music.core.model.dbi | UserAlbum | UserAlbum | 8 | 1 | 6 |
| music-core | com.sismics.music.core.util | ImageUtil | resizeImage | 3 | 1 | 6 |
| music-core | com.sismics.music.core.util.dbi | QueryParam | QueryParam | 3 | 1 | 6 |
| music-core | com.sismics.music.core.model.dbi | Artist | Artist | 7 | 1 | 5 |
| music-core | com.sismics.music.core.model.dbi | Directory | Directory | 7 | 1 | 5 |
| music-core | com.sismics.music.core.model.dbi | RolePrivilege | RolePrivilege | 7 | 1 | 5 |
| music-core | com.sismics.music.core.model.dbi | AuthenticationToken | AuthenticationToken | 7 | 1 | 5 |
| music-core | com.sismics.music.core.service.player | PlayerService | notifyPlaying | 17 | 3 | 4 |
| music-core | com.sismics.util.lastfm | LastFmUtil | getLovedTracks | 4 | 1 | 4 |
| music-core | com.sismics.util.lastfm | LastFmUtil | getAllTracks | 4 | 1 | 4 |
| music-core | com.sismics.util.log4j | LogEntry | LogEntry | 6 | 1 | 4 |
| music-core | com.sismics.util.dbi | BaseDao | findByCriteria | 3 | 1 | 4 |

   Here we can see that `Track` and `User` class have PC of upto 15 which is too large and can be modularized by passing objects.

2. **Cyclomatic Complexity (CC):**

   Cyclomatic complexity counts number of independent paths through program's code.The higher the cyclomatic complexity, the more difficult it is to understand, maintain, and test the code.

   Implications: It can result in difficulty of code reuse, decreased maintainability, increased time and cost of testing.

| Project Name | Package Name | Type Name | MethodName | LOC | CC | PC |
|---|---|---|---|---|---|---|
| music-core | com.sismics.util.dbi | SQLSplitter | consumeQuote | 51 | 18 | 2 |
| music-core | com.sismics.music.core.dao.dbi | TrackDao | getQueryParam | 70 | 15 | 2 |
| music-core | com.sismics.util.dbi | SQLSplitter | consumeComment | 27 | 12 | 2 |
| music-core | com.sismics.util | ResourceUtil | list | 45 | 11 | 3 |
| music-core | com.sismics.music.core.service.importaudio | ImportAudioService | run | 61 | 11 | 0 |
| music-core | com.sismics.music.core.service.collection | CollectionWatchService | run | 35 | 9 | 0 |
| music-core | com.sismics.util.mime | MimeTypeUtil | guessMimeType | 33 | 8 | 1 |
| music-core | com.sismics.music.core.service.collection | CollectionService | readTrackMetadata | 74 | 8 | 3 |
| music-core | com.sismics.music.core.service.albumart | AlbumArtImporter | scanDirectory | 24 | 7 | 1 |
| music-core | com.sismics.music.core.dao.dbi | AlbumDao | getQueryParam | 36 | 7 | 2 |
| music-core | com.sismics.music.core.util | ImageUtil | getFileFormat | 30 | 7 | 1 |
| music-core | com.sismics.music.core.util | ImageUtil | makeMosaic | 30 | 7 | 2 |
| music-core | com.sismics.music.core.util | DirectoryUtil | getBaseDataDirectory | 24 | 7 | 0 |
| music-core | com.sismics.util | LyricUtil | getLyrics | 24 | 6 | 2 |
| music-core | com.sismics.music.core.dao.dbi | PlaylistDao | getQueryParam | 26 | 6 | 2 |
| music-core | com.sismics.music.core.util | ImageUtil | resizeImage | 23 | 6 | 7 |
| music-core | com.sismics.util | LocaleUtil | getLocaleIdFromAcceptLanguag | 21 | 5 | 1 |
| music-core | com.sismics.util.log4j | MemoryAppender | find | 25 | 5 | 2 |
| music-core | com.sismics.util.dbi | SQLSplitter | consumeParentheses | 15 | 5 | 2 |
| music-core | com.sismics.util.dbi | SQLSplitter | iterator | 34 | 5 | 0 |
| music-core | com.sismics.util.dbi | SQLSplitter | next | 22 | 5 | 0 |

As we can see that there are some classes which have CC value of upto 18 or 15 so in the refactoring process we can try to reduce the cyclomatic complexity.

3. **Weighted Methods per Class (WMC):**

   The weighted sum of all class' methods an represents the McCabe complexity of a class. It is equal to number of methods, if the complexity is taken as 1 for each method. Research suggests that a WMC of 25 or less is desirable for a well-designed and maintainable class.

   Implications: In inheritance if base class has high number of method, it affects its' child classes and all methods are represented in sub-clasess. If number of methods is high, that class possibly domain spesific. Thefore they are less reusable.

| Project Nam | Package Name | Type Name | NOF | NOPF | NOM | NOPM | LOC | WMC |
|---|---|---|---|---|---|---|---|---|
| music-core | com.sismics.util.dbi | SQLSplitter | 3 | 0 | 13 | 5 | 180 | 60 |
| music-core | com.sismics.music.core.service.impor | ImportAudioService | 6 | 0 | 17 | 11 | 370 | 49 |
| music-core | com.sismics.music.core.dao.dbi.dto | TrackDto | 17 | 0 | 34 | 34 | 176 | 34 |
| music-core | com.sismics.music.core.model.dbi | Track | 15 | 0 | 33 | 33 | 298 | 33 |
| music-core | com.sismics.music.core.service.collec | CollectionWatchSer | 4 | 0 | 16 | 7 | 218 | 32 |
| music-core | com.sismics.music.core.service.collec | CollectionService | 1 | 0 | 12 | 7 | 204 | 30 |
| music-core | com.sismics.music.core.util.dbi | PaginatedLists | 2 | 0 | 12 | 6 | 173 | 30 |
| music-core | com.sismics.music.core.model.dbi | User | 12 | 0 | 27 | 27 | 241 | 27 |
| music-core | com.sismics.music.core.util | ImageUtil | 2 | 0 | 7 | 7 | 185 | 26 |
| music-core | com.sismics.music.core.service.lastfm | LastFmService | 1 | 0 | 14 | 11 | 214 | 24 |
| music-core | com.sismics.music.core.model.dbi | Album | 9 | 0 | 23 | 23 | 196 | 23 |
| music-core | com.sismics.music.core.service.impor | ImportAudio | 11 | 0 | 21 | 21 | 206 | 21 |
| music-core | com.sismics.music.core.dao.dbi | TrackDao | 0 | 0 | 7 | 6 | 135 | 21 |
| music-core | com.sismics.music.core.model.dbi | Transcoder | 8 | 0 | 19 | 19 | 165 | 19 |
| music-core | com.sismics.music.core.dao.dbi.criteri | TrackCriteria | 9 | 0 | 18 | 18 | 105 | 18 |
| music-core | com.sismics.util.dbi | DbOpenHelper | 4 | 0 | 9 | 7 | 115 | 18 |
| music-core | com.sismics.music.core.model.dbi | UserTrack | 7 | 0 | 17 | 17 | 146 | 17 |
| music-core | com.sismics.music.core.model.dbi | Directory | 5 | 0 | 16 | 16 | 123 | 17 |
| music-core | com.sismics.music.core.model.context | AppContext | 13 | 0 | 15 | 12 | 195 | 17 |
| music-core | com.sismics.music.core.dao.dbi | AlbumDao | 0 | 0 | 9 | 9 | 122 | 17 |
| music-core | com.sismics.music.core.model.dbi | UserAlbum | 6 | 0 | 15 | 15 | 127 | 15 |

   As we can see here the value of WMC is much higher in the import and dto classes so this can be handled during refactoring.

4. **Lack of cohesion on methods (LCOM):**

   Lack of Cohesion of Methods (LCOM) is based on the principle of high cohesion, which states that modules should be composed of elements that have a strong functional relationship and that work together to achieve a single goal. LCOM measures the degree to which the methods in a class are related to each other, and is used to identify classes that are not cohesive.

   Implication: A lower LCOM4 value indicates higher cohesion, while a higher LCOM4 value indicates lower cohesion. In general, an LCOM4 value greater than 1 is considered to indicate low cohesion, while a value less than or equal to 1 is considered to indicate high cohesion.

| B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Package Name | Type Name | NOF | NOPF | NOM | NOPM | LOC | WMC | NC | DIT | LCOM | FANIN |
| com.sismics.util.dbi | ObjectMapper | 1 | 1 | 5 | 3 | 36 | 5 | 0 | 0 | 1 | 2 |
| com.sismics.music.core.util | TransactionUtil | 1 | 0 | 2 | 2 | 67 | 6 | 0 | 0 | 1 | 13 |
| com.sismics.music.core.util | ImageUtil | 2 | 0 | 7 | 7 | 185 | 26 | 0 | 0 | 0.714285714285714 | 3 |
| com.sismics.util.dbi | DBIF | 3 | 0 | 6 | 5 | 82 | 10 | 0 | 0 | 0.666666666666667 | 3 |
| com.sismics.util.context | ThreadLocalContext | 2 | 1 | 5 | 4 | 51 | 6 | 0 | 0 | 0.6 | 20 |
| com.sismics.music.core.service.collec | CollectionService | 1 | 0 | 12 | 7 | 204 | 30 | 0 | 0 | 0.583333333333333 | 2 |
| com.sismics.util | EnvironmentUtil | 6 | 0 | 10 | 10 | 85 | 10 | 0 | 0 | 0.5 | 3 |
| com.sismics.util.log4j | LogCriteria | 3 | 0 | 6 | 6 | 60 | 6 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.service.impor | ImportAudioFile | 7 | 0 | 14 | 14 | 54 | 14 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.dao.dbi.criteri | PlaylistCriteria | 4 | 0 | 8 | 8 | 50 | 8 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.dao.dbi.criteri | ArtistCriteria | 2 | 0 | 4 | 4 | 28 | 4 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.dao.dbi.criteri | TrackCriteria | 9 | 0 | 18 | 18 | 105 | 18 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.dao.dbi.criteri | AlbumCriteria | 5 | 0 | 10 | 10 | 61 | 10 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.dao.dbi.dto | TrackDto | 17 | 0 | 34 | 34 | 176 | 34 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.dao.dbi.dto | UserDto | 5 | 0 | 10 | 10 | 56 | 10 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.dao.dbi.dto | ArtistDto | 2 | 0 | 4 | 4 | 26 | 4 | 0 | 0 | 0.5 | 2 |
| com.sismics.music.core.dao.dbi.dto | AlbumDto | 7 | 0 | 14 | 14 | 76 | 14 | 0 | 0 | 0.5 | 1 |
| com.sismics.music.core.dao.dbi.dto | PlaylistDto | 5 | 0 | 10 | 10 | 56 | 10 | 0 | 0 | 0.5 | 2 |
| com.sismics.music.core.service.impor | ImportAudioService | 6 | 0 | 17 | 11 | 370 | 49 | 0 | 0 | 0.470588235294118 | 1 |
| com.sismics.util.log4j | MemoryAppender | 2 | 0 | 7 | 6 | 88 | 14 | 0 | 0 | 0.428571428571429 | 0 |
| com.sismics.util.dbi.filter | FilterColumn | 2 | 0 | 7 | 7 | 32 | 7 | 0 | 0 | 0.428571428571429 | 0 |

   In the given code max LCOM value is 1 which indicates the presence of high cohesion in our code.

5. **Number of Methods and Number of Public Methods:**

   This metric can provide insight into how complex the codebase is, and how much effort would be required to modify or maintain it. The number of public methods also provides a measure of how much of the code is exposed and available for use by external classes.

   Implication: A higher number of public methods could indicate a more complex or poorly designed codebase, as it can make it more difficult to maintain or understand the code. It can also make the code more difficult to test.

| Project Nan | Package Name | Type Name | NOF | NOPF | NOM | NOPM | LOC | WMC | NC | DIT | LCOM |
|---|---|---|---|---|---|---|---|---|---|---|---|
| music-core | com.sismics.music.core.dao.dbi.dto | TrackDto | 17 | 0 | 34 | 34 | 176 | 34 | 0 | 0 | 0 |
| music-core | com.sismics.music.core.model.dbi | Track | 15 | 0 | 33 | 33 | 298 | 33 | 0 | 0 | 0.0606060606060 |
| music-core | com.sismics.music.core.model.dbi | User | 12 | 0 | 27 | 27 | 241 | 27 | 0 | 0 | 0.0740740740740 |
| music-core | com.sismics.music.core.model.dbi | Album | 9 | 0 | 23 | 23 | 196 | 23 | 0 | 0 | 0.086956521739 |
| music-core | com.sismics.music.core.service.impor | ImportAudio | 11 | 0 | 21 | 21 | 206 | 21 | 0 | 0 | |
| music-core | com.sismics.music.core.model.dbi | Transcoder | 8 | 0 | 19 | 19 | 165 | 19 | 0 | 0 | 0.1052631578947 |
| music-core | com.sismics.music.core.dao.dbi.criteri | TrackCriteria | 9 | 0 | 18 | 18 | 105 | 18 | 0 | 0 | 0 |
| music-core | com.sismics.music.core.model.dbi | UserTrack | 7 | 0 | 17 | 17 | 146 | 17 | 0 | 0 | 0.1176470588235 |
| music-core | com.sismics.music.core.model.dbi | Directory | 5 | 0 | 16 | 16 | 123 | 17 | 0 | 0 | 0.1 |
| music-core | com.sismics.music.core.model.dbi | UserAlbum | 6 | 0 | 15 | 15 | 127 | 15 | 0 | 0 | 0.1333333333333 |
| music-core | com.sismics.music.core.service.impor | ImportAudioFile | 7 | 0 | 14 | 14 | 54 | 14 | 0 | 0 | 0 |
| music-core | com.sismics.music.core.dao.dbi.dto | AlbumDto | 7 | 0 | 14 | 14 | 76 | 14 | 0 | 0 | 0 |
| music-core | com.sismics.music.core.model.dbi | Playlist | 3 | 0 | 13 | 13 | 86 | 13 | 0 | 0 | 0.3076923076923 |
| music-core | com.sismics.music.core.model.dbi | Artist | 5 | 0 | 13 | 13 | 108 | 13 | 0 | 0 | 0.1538461538461 |
| music-core | com.sismics.music.core.model.dbi | RolePrivilege | 5 | 0 | 13 | 13 | 108 | 13 | 0 | 0 | 0.1538461538461 |
| music-core | com.sismics.music.core.model.dbi | AuthenticationToker | 5 | 0 | 13 | 13 | 108 | 13 | 0 | 0 | 0.1538461538461 |
| music-core | com.sismics.music.core.model.dbi | PlaylistTrack | 4 | 0 | 12 | 12 | 93 | 12 | 0 | 0 | 0.1666666666666 |
| music-core | com.sismics.music.core.model.context | AppContext | 13 | 0 | 15 | 12 | 195 | 17 | 0 | 0 | 0.1333333333333 |
| music-core | com.sismics.music.core.service.impor | ImportAudioService | 6 | 0 | 17 | 11 | 370 | 49 | 0 | 0 | 0.4705882352941 |
| music-core | com.sismics.music.core.service.lastfm | LastFmService | 1 | 0 | 14 | 11 | 214 | 24 | 0 | 0 | 0.3571428571428 |
| music-core | com.sismics.music.core.model.dbi | Role | 4 | 0 | 11 | 11 | 89 | 11 | 0 | 0 | 0.1818181818181 |

   As shown in the code, there are many methods in some of the classes i.e. Track, User, TrackDto

6. **Depth of Inheritance of Tree:**

   The position of the class in the inheritance tree. Has 0 (zero) value for root and non-inherited classes.For the multiple inheritance, the metric shows the maximum length.

   This metric value is low for most of the classes but there are few classes for whom value is 2 according to tool `CodeMR`

   Some of these classes are: `AlbumDao, TrackDao, UserDao, ArtistDao, ImportAudioService`

7. **Coupling:**

   Response for a class: The number of the methods that can be potentially invoked in response to a public message received by an object of a particular class. It includes the full call graph of any method called from the originating method.

   Implication: If the number of methods that can be invoked at a class is high, then the class is considered more complex and can be highly coupled to other classes. Therefore more test and maintain effort is required.

   From the following figure it can be concluded that RFC is very high for class `Collection Service` and `LastFmService`

Classes with high coupling, high complexity (#2)

| ID | CLASS | COUPLING | COMPLEXITY | LACK OF COHESION | SIZE | LOC | CBO | WMC | RFC | NOM |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | CollectionService | | | | | 135 | 29 | 34 | 250 | 12 |
| 2 | LastFmService | | | | | 134 | 23 | 30 | 173 | 14 |

# Task 3: Refactoring

## Task 3a: Removing the Design Smells

1. **Insufficient Modularization**

   The class itself was just performing a single function in a long method, now the responsibilities of the long method has been split into multiple methods

   To reduce the cyclomatic complexity of the `guessMimeType` method, one approach is to extract the logic of each `if` condition into separate methods. Here's an example of refactored code:

```
public static String guessMimeType(File file) throws Exception {
    InputStream is = null;
    try {
        is = new FileInputStream(file);
        byte[] headerBytes = new byte[64];
        int readCount = is.read(headerBytes, 0, headerBytes.length);
        if (readCount <= 0) {
            throw new Exception("Cannot read input file");
        }
        String header = new String(headerBytes, "US-ASCII");

        if (isZip(header)) {
            return MimeType.APPLICATION_ZIP;
        }
        if (isGif(header)) {
            return MimeType.IMAGE_GIF;
        }
        if (isJpeg(headerBytes)) {
            return MimeType.IMAGE_JPEG;
        }
        if (isPng(headerBytes)) {
            return MimeType.IMAGE_PNG;
        }
        if (isIco(headerBytes)) {
            return MimeType.IMAGE_X_ICON;
        }
    } finally {
        if (is != null) {
            is.close();
        }
    }
    return null;
}

private static boolean isZip(String header) {
    return header.startsWith("PK");
}

private static boolean isGif(String header) {
    return header.startsWith("GIF87a") || header.startsWith("GIF89a");
}

private static boolean isJpeg(byte[] headerBytes) {
    return headerBytes[0] == ((byte) 0xff) && headerBytes[1] == ((byte) 0xd8);
}

private static boolean isPng(byte[] headerBytes) {
    return headerBytes[0] == ((byte) 0x89) && headerBytes[1] == ((byte) 0x50) && headerBytes[2] == ((byte) 0
x4e) && headerBytes[3] == ((byte) 0x47) &&
           headerBytes[4] == ((byte) 0x0d) && headerBytes[5] == ((byte) 0x0a) && headerBytes[6] == ((byte) 0
x1a) && headerBytes[7] == ((byte) 0x0a);
}

private static boolean isIco(byte[] headerBytes) {
    return headerBytes[0] == ((byte) 0x00) && headerBytes[1] == ((byte) 0x00) && headerBytes[2] == ((byte) 0
x01) && headerBytes[3] == ((byte) 0x00);
}
```

After refactoring the code  the cyclomatic complexity of the `guessMimeType` method has been reduced to 5. This is because there are now five paths through the method, one for each of the new private methods `isZip`, `isGif`, `isJpeg`, `isPng`, and `isIco`. Since each of these methods only contributes a single path to the overall method, the total cyclomatic complexity is reduced to 5.

- Insufficient modularization - **void run() in importMusic**

  The refactored version separates the different responsibilities of the original function into smaller, more focused methods, making the code more modular and easier to understand.

```
protected void run() throws Exception {
    while (isRunning()) {
        ImportAudio importAudio = importAudioQueue.take();
        importAudio.setStatus(ImportAudio.Status.INPROGRESS);

        try {
            log.info("Start importing a new URL: " + importAudio);

            Process process = startImportProcess(importAudio);
            readImportOutput(importAudio, process);
            int exitValue = process.waitFor();

            if (exitValue == 0) {
                importAudio.setStatus(ImportAudio.Status.DONE);
            } else {
                importAudio.setStatus(ImportAudio.Status.ERROR);
                removeWorkingFiles(importAudio);
                importAudio.setMessage(importAudio.getMessage() + "\nProcess exit with code: " + exitValue);
            }

        } catch (Exception e) {
            importAudio.setStatus(ImportAudio.Status.ERROR);
            importAudio.setMessage(importAudio.getMessage() + "\n" + e.getMessage());
            log.error("Error importing: " + importAudio, e);
        } finally {
            importAudio.setProcess(null);
            importAudioList.add(importAudio);
        }
    }
}

private Process startImportProcess(ImportAudio importAudio) throws IOException {
    String output = DirectoryUtil.getImportAudioDirectory().getAbsolutePath() + File.separator + "%(title)
s.%(ext)s";
    String command = String.format("youtube-dl -v --prefer-ffmpeg --encoding UTF-8 --newline -f bestaudio -x
--audio-format %s --audio-quality %s -o %s %s",
            importAudio.getFormat(), importAudio.getQuality(), output, importAudio.getUrl());

    List<String> commandList = new LinkedList<>(Arrays.asList(StringUtils.split(command)));
    ProcessBuilder processBuilder = new ProcessBuilder(commandList);
    processBuilder.redirectErrorStream(true);

    return processBuilder.start();
}

private void readImportOutput(ImportAudio importAudio, Process process) throws IOException {
    try (BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()))) {
        String line;
        while ((line = reader.readLine()) != null && isProcessRunning(process)) {
            updateImportStatus(importAudio, line);
            importAudio.setMessage(importAudio.getMessage() + "\n" + line);
        }
    }
}

private void updateImportStatus(ImportAudio importAudio, String line) {
    Matcher matcher = PROGRESS_PATTERN.matcher(line);
    if (matcher.find()) {
```

```
        importAudio.setProgress(Float.valueOf(matcher.group(1)));
        importAudio.setTotalSize(matcher.group(2));
        importAudio.setDownloadSpeed(matcher.group(3));
    }

    if (line.contains("ERROR")) {
        importAudio.setStatus(ImportAudio.Status.ERROR);
    }

    matcher = WORKING_FILE_PATTERN.matcher(line);
    if (matcher.find()) {
        File file = new File(matcher.group(1));
        String name = file.getName();
        if (!Strings.isNullOrEmpty(name)) {
            importAudio.addWorkingFiles(name);
        }
    }
}

private void removeWorkingFiles(ImportAudio importAudio) {
    for (String name : importAudio.getWorkingFiles()) {
        File file = new File(DirectoryUtil.getImportAudioDirectory().getAbsolutePath() + File.separator + na
me);
        if (file.exists()) {
            file.delete();
        }
    }
}
```

The refactored version separates the different responsibilities of the original function into smaller, more focused methods, making the code more modular and easier to understand.

`startImportProcess()` method creates a `Process` object to run the YouTube-DL command with the provided importAudio object.

`readImportOutput()` method reads the standard output of the process and updates the status of the importAudio object accordingly.

3. **Missing Abstraction**

   - Created 3 new classes: DateCommon, DateCommonUpdate, DateCommonDisabled. The DateCommonUpdate class extends the DateCommon class and represents a model object that includes a Date object representing the time of the last update. Same holds for the DateCommonDisabled class. These 2 are extended from DateCommon class to suit two different model classes: Album and Directory respectively

```
//DateCommon.java
package com.sismics.music.core.model.dbi;

import com.google.common.base.Objects;
import com.sismics.util.context.ThreadLocalContext;
import org.skife.jdbi.v2.Handle;

import java.util.Date;

public class DateCommon
{
    private Date createDate;
    private Date deleteDate;

    DateCommon(){

    }
    DateCommon(Date createDate, Date deleteDate)
    {
        this.createDate=createDate;
        this.deleteDate=deleteDate;
```

```
    }
    /**
     * Getter of createDate.
     *
     * @return createDate
     */
    public Date getCreateDate() {
        return createDate;
    }

    /**
     * Setter of createDate.
     *
     * @param createDate createDate
     */
    public void setCreateDate(Date createDate) {
        this.createDate = createDate;
    }
    /**
     * Getter of deleteDate.
     *
     * @return deleteDate
     */
    public Date getDeleteDate() {
        return deleteDate;
    }

    /**
     * Setter of deleteDate.
     *
     * @param deleteDate deleteDate
     */
    public void setDeleteDate(Date deleteDate) {
        this.deleteDate = deleteDate;
    }
}
```

```
//DateCommonUpdate.java
package com.sismics.music.core.model.dbi;

import com.google.common.base.Objects;
import com.sismics.util.context.ThreadLocalContext;
import org.skife.jdbi.v2.Handle;

import java.util.Date;

public class DateCommonUpdate extends DateCommon
{
    private Date updateDate;

    DateCommonUpdate(){
        super();
    }
    DateCommonUpdate(Date createDate, Date deleteDate, Date updateDate)
    {
        super(createDate,deleteDate);
        this.updateDate=updateDate;
    }
    /**
     * Getter of updateDate.
     *
     * @return updateDate
     */
    public Date getUpdateDate() {
        return updateDate;
    }

    /**
     * Setter of updateDate.
     *
```

```
     * @param updateDate updateDate
     */
    public void setUpdateDate(Date updateDate) {
        this.updateDate = updateDate;
    }
}
```

- Similarly the function calls were now replaced in the Dao Files (updated in the codebase) to suit the function calls.
- The major benefit of this refactoring is that we now have a common class implementing date functions, so the four getter-setter functions need not be present in each class. Also, the code becomes more extensible, for example if we need a function that tells us the hours passed until an album has been created, we can do so in the DateCommon class by adding a simple function, which will be inherited for its subclasses and can be used across all model classes.

4. **consumeQuote - violation of Single Responsibility Principle**

```
/**
     * Skips the index past the quote.
     *
     * @param s     The string
     * @param start The starting character of the quote.
     * @return The index that skips past the quote starting at start. If the quote does not start at that point
     */
    static int consumeQuote(final CharSequence s, final int start) {
        if (start >= s.length()) return start;
        char ender = getQuoteEnder(s, start);
        return findQuoteEnd(s, start, ender);
    }

    private static char getQuoteEnder(final CharSequence s, final int start) {
        switch (s.charAt(start)) {
            case '\'':
                return '\'';
            case '"':
                return '"';
            case '[':
                return ']';
            case '`':
                return '`';
            case '$':
                return '$';
            default:
                return '\0';
        }
    }

    private static int findQuoteEnd(final CharSequence s, final int start, final char ender) {
        if (ender == '$') {
            return findDollarQuoteEnd(s, start);
        }

        boolean escaped = false;
        for (int i = start + 1; i < s.length(); ++i) {
            if (escaped) {
                escaped = false;
                continue;
            }
            char c = s.charAt(i);
            if (c == '\\')
                escaped = true;
            else if (c == ender)
                return i + 1;
        }
        return s.length();
    }
```

```
    private static int findDollarQuoteEnd(final CharSequence s, final int start) {
        int quoteEnd = start + 1;
        for (; s.charAt(quoteEnd) != '$'; ++quoteEnd)
            if (quoteEnd >= s.length())
                return quoteEnd;

        int i = quoteEnd + 1;
        while (i < s.length()) {
            if (s.charAt(i) == '$') {
                boolean match = isDollarQuoteMatch(s, start, quoteEnd, i);
                if (match) {
                    return i + 1;
                }
            } else {
                ++i;
            }
        }
        return i;
    }

    private static boolean isDollarQuoteMatch(final CharSequence s, final int start, final int quoteEnd, int i
        for (int j = start; j <= quoteEnd && i < s.length(); ++j, ++i) {
            if (s.charAt(i) != s.charAt(j)) {
                return false;
            }
        }
        return true;
    }
```

- The consumeQuote method first calls the getQuoteEnder method to determine the type of quote that starts at the specified position, and then calls the findQuoteEnd method to locate the end of the quote.
- Increases the readability and maintainability of the code and deals with the long method code smell.

5. **Dead code and Primitive Obsession**

- The blocks mentioned for dead code are removed above. The benfits involve improved readability, maintainability and a more compact code base. Also improves performance as mentioned above
- There is and overloading of responsibilities primitive obsession in `Track` and `User` class, some of which we have tried to resolve using our `DateCommon` class and its subclasses. Another idea we had was splitting the `Track` class into `Track` and `TrackMetaData` classes, encapsulation related variables and the Dao class calling both of them instead of one such that the data base be not modified. However, that would have messed up any new Tests written and would have changed a little bit of the functionality aspect for a tester, although being unaffected for a user.

## Task 3b: Remeasure of Code Metrics

After remeasuring the metrics, generated report by designite has been uploaded in the `docs` folder, pre and post refactoring. It impacted some of our discussed code metrics.

1. **Parameter Count (PC)**

   We did not reduce the parameter count, on the account that interface for testing would be changed for the `Track` and `User` classes which have absurdly high parameter in their constructor calls.

2. **Cyclomatic Complexity (CC)**

   Maximum Cyclomatic complexity got reduced to 15 after refactoring the code.

   Inititally for SQL Splitter class it was 18 now it got reduced to 2 for the main consume function and 6 for each sub functions. Indicates refactoring was successful

3. **Weighted Methods per Class (WMC)**

The Weighted Methods per Class increases for SQL Splitter but it is not a bad indicator specifically since the delta is very small and is caused only due to modularization

4. **Number of Methods and Number of Public Methods:**

   Number of Public Methods have been reduced by 4 for each of the classes using `DateCommon` class.

5. **Coupling**

   RFC(response of a class dereased by 2%, though not much it is an indication that refactoring was being done in the right direction.

---

# Team Contribution

- Vaibhav Kashera
  - Analysis of code
  - Identification of Design Smells
  - Refactoring of Code
- Soma Datta
  - Mining the repository
  - Identification of classes
  - UML diagrams
- Ashish Gupta
  - UML diagrams
  - Mining the repository
  - Code metrics analysis
- Isha Gupta
  - Identifiation of Code metrics
  - Analysis of code metrics
  - Documentation
- Arth Raj
  - Documentation
  - Identification of Design smells
  - Refactoring of code