🚀 Project: Science Pixel (Final Production Edition)

File Type: Vibe Coding Master Manifest

Version: 1.0.0 (Production Ready)

Updates: RSS Rome 파서, FFmpeg Concat 로직, 파일 청소(Cleanup), 유튜브 업로드 준비

📂 1. Project Architecture (Refined)

1.1 Tech Stack

- **Core:** Spring Boot 3.2, Spring Batch 5, Kotlin
- **Data:** MariaDB (Meta), MongoDB (Business)
- **Ops:** Docker Compose
- **External:** YouTube Data API v3, Pexels, Gemini, Edge-TTS

1.2 Improved Workflow

1. **Reader:** Rome 라이브러리로 실제 RSS 피드(XML) 파싱.
2. **Processor:**
    - Gemini Script & Vision Check.
    - Scene별 Resource 다운로드 & TTS.
    - Scene별 Clip 생성.
    - **[New]** ffmpeg concat으로 전체 Clip 병합.
3. **Writer:** MongoDB에 메타데이터 저장 (Status: PENDING_UPLOAD).
4. **Cleanup (Tasklet):** workspace 임시 파일 삭제.
5. **Scheduler (Separate):** DB를 조회하여 PENDING_UPLOAD 영상을 유튜브에 업로드 (Quota 관리).

💻 2. Spring Boot Application

build.gradle.kts (Dependencies Added)

```
// ... 기존 의존성 유지 ...

dependencies {
    // 1. RSS Parsing
    implementation("com.rometools:rome:2.1.0")

    // 2. YouTube Upload
    implementation("com.google.apis:google-api-services-youtube:v3-rev222-1.25.0")
```

```
    implementation("com.google.auth:google-auth-library-oauth2-http:1.19.0")

    // ... MariaDB, MongoDB, Web, Batch 등 ...
}
```

src/main/kotlin/com/sciencepixel/batch/RssItemReader.kt (Real Implementation)

실제 RSS URL에서 데이터를 가져오는 Reader입니다.

```
package com.sciencepixel.batch

import com.rometools.rome.io.SyndFeedInput
import com.rometools.rome.io.XmlReader
import org.springframework.batch.item.ItemReader
import java.net.URL
import java.util.LinkedList

class RssItemReader(private val feedUrl: String) : ItemReader<NewsItem> {
    private val items = LinkedList<NewsItem>()
    private var initialized = false

    override fun read(): NewsItem? {
        if (!initialized) {
            fetchFeed()
            initialized = true
        }
        return items.poll()
    }

    private fun fetchFeed() {
        try {
            val feed = SyndFeedInput().build(XmlReader(URL(feedUrl)))
            feed.entries.take(3).forEach { entry -> // 최신 3개만
                items.add(NewsItem(
                    title = entry.title,
                    summary = entry.description?.value ?: entry.title
                ))
            }
        } catch (e: Exception) {
            println("❌ RSS Fetch Error: ${e.message}")
        }
```

```
    }
}
```

src/main/kotlin/com/sciencepixel/service/ProductionService.kt (Enhanced FFmpeg Logic)

클립 병합(Concat) 로직이 추가되었습니다.

```
package com.sciencepixel.service

import org.springframework.stereotype.Service
import java.io.File

@Service
class ProductionService {

    fun produceVideo(title: String, scenes: List<Scene>): String {
        val workspace = File("workspace/job_${System.currentTimeMillis()}").apply { mkdirs() }
        val clipFiles = mutableListOf<File>()

        // 1. Scene별 클립 생성
        scenes.forEachIndexed { i, scene ->
            // (Mock: Pexels Download & TTS Logic)
            val clip = File(workspace, "clip_$i.mp4")
            // videoEditor.createClip(..., clip) 호출
            clipFiles.add(clip)
        }

        // 2. [New] Merge Clips (Concat Demuxer)
        val fileList = File(workspace, "file_list.txt")
        fileList.bufferedWriter().use { out ->
            clipFiles.forEach { out.write("file '${it.absolutePath}'\n") }
        }

        val finalOutput = File(workspace, "final_output.mp4")

        // ffmpeg -f concat -safe 0 -i file_list.txt -c copy output.mp4
        // 주의: -c copy는 코덱이 동일할 때만 가능. 다르면 재인코딩 필요.
        val cmd = listOf(
            "ffmpeg", "-y", "-f", "concat", "-safe", "0",
            "-i", fileList.absolutePath,
            "-c", "copy", // 재인코딩 없이 빠르게 병합 (모든 클립 설정이 동일해야 함)
```

```
            finalOutput.absolutePath
        )

        val process = ProcessBuilder(cmd).redirectErrorStream(true).start()
        process.waitFor()

        return finalOutput.absolutePath
    }

    // 작업 완료 후 폴더 삭제 메서드
    fun cleanupWorkspace(path: String) {
        File(path).parentFile.deleteRecursively()
    }
}
```

src/main/kotlin/com/sciencepixel/batch/ShortsBatchConfig.kt (Updated Job)

RSS Reader 적용 및 Cleanup 리스너 추가.

```
package com.sciencepixel.batch

import com.sciencepixel.domain.VideoHistory
import com.sciencepixel.repository.VideoHistoryRepository
import com.sciencepixel.service.ProductionService
import org.springframework.batch.core.Job
import org.springframework.batch.core.Step
import org.springframework.batch.core.JobExecution
import org.springframework.batch.core.JobExecutionListener
import org.springframework.batch.core.job.builder.JobBuilder
import org.springframework.batch.core.repository.JobRepository
import org.springframework.batch.core.step.builder.StepBuilder
import org.springframework.batch.item.ItemProcessor
import org.springframework.batch.item.ItemReader
import org.springframework.batch.item.ItemWriter
import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.Configuration
import org.springframework.transaction.PlatformTransactionManager

@Configuration
class ShortsBatchConfig(
    private val jobRepository: JobRepository,
```

```
    private val transactionManager: PlatformTransactionManager,
    private val productionService: ProductionService,
    private val videoHistoryRepository: VideoHistoryRepository
) {

    @Bean
    fun shortsJob(): Job {
        return JobBuilder("shortsJob", jobRepository)
            .start(shortsStep())
            .listener(object : JobExecutionListener {
                override fun afterJob(jobExecution: JobExecution) {
                    println("🧹 Batch Job Finished. Clean up logic can be placed here.")
                }
            })
            .build()
    }

    @Bean
    fun shortsStep(): Step {
        return StepBuilder("step1", jobRepository)
            .chunk<NewsItem, VideoHistory>(1, transactionManager)
            .reader(realRssReader()) // Real RSS
            .processor(videoProcessor())
            .writer(mongoWriter())
            .build()
    }

    @Bean
    fun realRssReader(): ItemReader<NewsItem> {
        // ScienceDaily Top News
        return
RssItemReader("[https://www.sciencedaily.com/rss/top_news.xml](https://www.sciencedaily.com/rss/top_news.xml)")
    }

    // ... Processor & Writer (Same as before) ...
}
```

src/main/kotlin/com/sciencepixel/service/YoutubeUploadScheduler.kt (New!)

배치와 별개로, 생성된 영상을 주기적으로 체크하여 유튜브에 업로드하는

스케줄러입니다.

```
package com.sciencepixel.service

import com.sciencepixel.repository.VideoHistoryRepository
import org.springframework.scheduling.annotation.Scheduled
import org.springframework.stereotype.Service

@Service
class YoutubeUploadScheduler(
    private val repository: VideoHistoryRepository
) {

    // 1시간마다 실행
    @Scheduled(fixedDelay = 3600000)
    fun uploadPendingVideos() {
        val pendingVideos = repository.findAll().filter { it.status == "COMPLETED" } // Upload
Pending state

        pendingVideos.forEach { video ->
            try {
                println("🚀 Uploading to YouTube: ${video.title}")
                // YouTube API Upload Logic Here...
                // uploadService.upload(video.filePath, video.title, video.summary)

                // Update Status
                val updated = video.copy(status = "UPLOADED", youtubeUrl =
"[https://youtu.be/](https://youtu.be/)...")
                repository.save(updated)

                // Cleanup Local File after upload
                // File(video.filePath).delete()

            } catch (e: Exception) {
                println("❌ Upload Failed: ${e.message}")
            }
        }
    }
}
```

💡 Final Checkpoints (Before Run)

1. **FFmpeg** 설치 확인**:** 서버(또는 Docker 이미지)에 ffmpeg가 설치되어 있어야 합니다.
    - DockerFile에 RUN apt-get update && apt-get install -y ffmpeg 추가 필수.
2. 한글 폰트**:** 리눅스 환경에서 나눔고딕(/usr/share/fonts/...)이 없으면 자막이 깨집니다. 폰트 파일을 프로젝트 resources에 넣고 경로를 지정하는 것이 안전합니다.
3. **API Key** 보안**:** application.yml에 키를 넣지 말고, 실행 시 환경변수(GEMINI_API_KEY, PEXELS_API_KEY)로 주입하세요.
    - gemini.api-key: ${GEMINI_API_KEY} 형태로 변경.
4. **Quota** 관리**:** 유튜브 API는 할당량이 매우 적습니다. 테스트 단계에서는 public 대신 private 상태로 업로드하여 할당량을 아끼거나, 하루 업로드 갯수를 제한하는 로직을 스케줄러에 추가하세요.

이 매니페스트는 즉시 개발 착수가 가능한 **Production Level** 설계도입니다.