Projede Olması Gereken Entity'ler:

FileStats

- Her dosya için satır sayısı, karakter sayısı, hangi thread tarafından işlenildiği, ne zaman başladığı, ne zaman bittiği, toplam süre bilgilerini tutacaktır.

ArchiveInfo

- Arşivlenmiş dosya bilgilerini tutacaktır.
- Dosya adı, hangi thread tarafından işlendiği, ne zaman başladığı, ne zaman bittiği, toplam süre bilgilerini tutacaktır.

AnalysResult

- Ekrana yazdırmak için toplam satır sayısını toplam karakter sayısını, ne zaman başladığı, ne zaman bittiği, toplam süre bilgilerini tutacaktır.

Projede Olması Gereken Servisler

FileProcessingService

- Analiz işlemlerini yapacak
- Dosya okuma işlemi
- Satır sayısı hesaplama metodu **countLines();** olacaktır.
- Karakter sayısı hesaplama metodu **countCharacters();** olacaktır.
- Bu her iki metodu kullanacak (Çatı gibi düşünebilirsiniz) **analyzeFile()** adında bir metot daha olacak

  Örnek:

```
   private FileStats analyzeFile(Path filePath){
         int lineCount = countLines();
         int characterCount = countCharacters();
      return new FileStats(filePath.getFileName().toString(), lineCount,
characterCount);
   }
}
```

- Thread tarafından çalıştırılabilecek Callable<FileStats> dönüş değeri olan bir bir metot yazılacak bu metot analiz işleminin thread tarafından yapılmasını sağlayacaktır.

- Bu metot yeni bir metot da olabilir (analyzeFile metodunu sarmalayacak şekilde düşünebilirsiz) veya direkt analyzFile metodunu Callable<FileStats> dönüş değerine getirebiliriz

ArchiveService

- Arşivleme işlemlerini yapacaktır.
- **createArchive()** adında zip işlemi için metot olacak
- Zipleme metodu için Callable<T> yapısı gerekmeyebilir çünkü tek ayrı thread yönetecektir. (Bu kısmı ilgili kişi araştırıp-geliştirip takıma haber verecektir)
- **unzip()** adında (ismi değiştirilebilir) bir metot olacaktır zipten çıkarma işlemini yapacaktır
- **findTextFiles()** gibi dosyadaki txt dosyalarını bulan yardımcı metotlar eklenecektir.
- **deleteSourceFiles()** adında arşivleme sonrası dosyaları silmeye yarayan metot yazılacaktır (Bonus özellik öncelik açısından arkaplanda durabilir)

ThreadManagerService

- Şu ana kadar sadece metotlarımızı threadlerin kullanabileceği şekilde ayarladık(Callable). Herhangi bir thread kullanımı yapmadık.
- Dolayısıyla dosya analiz işleminin hangi thread havuzu tarafından (config dosyasında 2 tane thread havuzu olacaktır) başlatılacağını sağlayan kod burada olacaktır.
- Zipleme işleminin de aynı şekilde hangi thread havuzu tarafından başlatılacağını sağlayan kod burada olacaktır.

ThreadManagerService'in Tam anlaşılması için kod örneği alttaki sayfadadır

ThreadManagerService.java

ThreadManagerService.java

```java
/**
 * Thread yonetimi servis sinifi
 * Thread'lerin baslatilmasi, koordinasyonu ve beklenmesi islemlerini yonetir
 */
@Service
public class ThreadManagementService {

    @Autowired
    @Qualifier("fileAnalysisExecutor")
    private ExecutorService fileAnalysisExecutor;

    @Autowired
    @Qualifier("archiveExecutor")
    private ExecutorService archiveExecutor;

    @Autowired
    private FileProcessingService fileProcessingService;
    @Autowired
    private ArchiveService archiveService;

    /**
     * Dosya analizi thread'lerini çalıştırır
     */
    private AnalysisResult executeFileAnalysis(List<Path> filePaths) {
        logger.info("Dosya analizi başlatılıyor. Thread sayısı: {}", filePaths.size());

        AnalysisResult result = new AnalysisResult();
        List<Future<FileStats>> futures = new ArrayList<>();

        // Her dosya için ayrı thread başlat
        for (Path filePath : filePath ) {
            Future<FileStats> future = fileAnalysisExecutor.submit(
                fileProcessingService.analyzFile(filePath)
            );
            futures.add(future);
        }

        // Tüm thread'lerin tamamlanmasını bekle (join() mantığı)
        for (Future<FileStats> future : futures) {
            try {
                FileStats stats = future.get(); // Blocking call
                result.addFileStats(stats.getFileName(), stats);
                logger.info("Dosya işlendi: {}", stats.toString());
            } catch (Exception e) {
                logger.error("Dosya işleme hatası", e);
                result.setFailedFiles(result.getFailedFiles() + 1);
            }
        }

        logger.info("Dosya analizi tamamlandı");
        return result;
    }
```

Bu kodda örnek olarak sadece dosya analiz threadlerini çalıştırma örneği verilmiştir arşivleme thread'ini de çalıştıracak metot yazılacaktır.

FileAnalysisService

- Tam analiz sürecini başlatacak olan servistir
- Önceki servisler her biri bağımsız olarak işlevini yerine getirmektedir. Uygulamanın akışı
    1. Dosyaları Analiz Eder (Multithread) -- FileProcessingService
    2. Sonuçları Toplar
    3. Arşivleme İşlemini Başlatır –ArchiveService
    4. Analiz sonucunu döndürür

Bu nedenle bu servisleri tek akış içerisinde toplamamız gerekecektir. FileAnalysisService bu akışı tek bir yerde toplayacaktır. performCompleteAnalysis(); adında bir metod olabilir

FileUploadServie

- Kullanıcıların dosya yükleyebilmesi sağlanacak.
- Zip veya normal dosyalar yüklenebilecek
- Zip yüklendiğinde otomatik arşivden çıkarıp analize gönderecek

FileDownloadServie

- Kullanıcıların çıktı olarak zip dosyasını indirebilmesini sağlayacak

ThreadConfig

- Thread havuzlarını oluşturur.

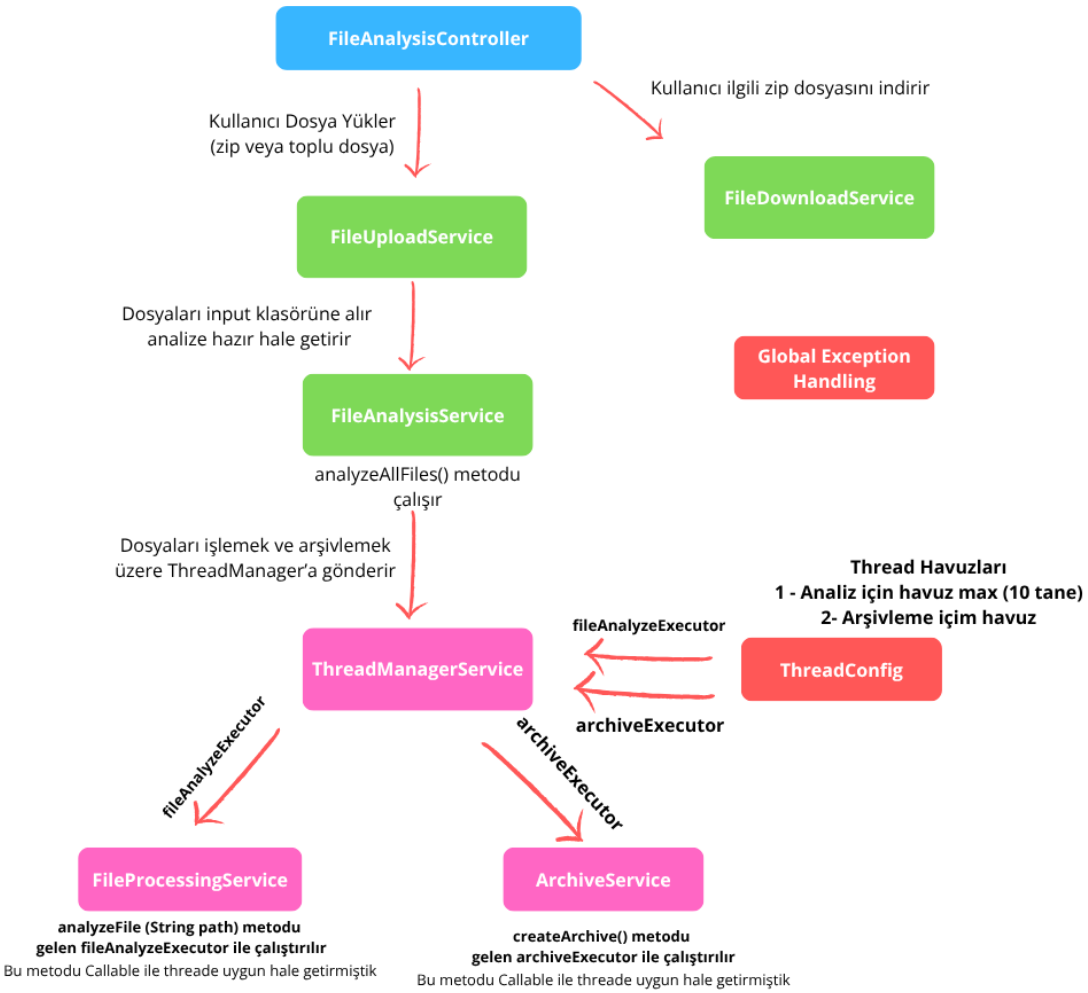
```java
public class ThreadPoolConfig {                                                    ⚠ 3 ^

    1 usage
    private static final int MAX_THREADS = 10; // Supports up to 10 files
    1 usage
    private static final String ANALYZE_THREAD_NAME_PREFIX = "FileAnalysis-";
    1 usage
    private static final String ARCHIVE_THREAD_NAME_PREFIX = "Archive-";

    /**
     * ExecutorService bean for file analysis
     * Uses a fixed thread pool with support for up to 10 files
     */
    umutsefkansak
    @Bean(name = "fileAnalysisExecutor")
    public ExecutorService fileAnalysisExecutor() {
        return Executors.newFixedThreadPool(MAX_THREADS, r -> {
            Thread thread = new Thread(r);
            thread.setName(ANALYZE_THREAD_NAME_PREFIX + thread.getId());
            thread.setDaemon(false); // Let the main thread wait
            return thread;
        });
    }

    /**
     * Single-threaded ExecutorService for archiving operations
     */
    umutsefkansak
    @Bean(name = "archiveExecutor")
    public ExecutorService archiveExecutor() {
        return Executors.newSingleThreadExecutor(r -> {
```

Daha iyi anlaşılabilmesi adına aşağıda kod akışı şekillerle gösterilmiştir.



1. Kısım Görev Dağılımı

Umut Sefkan SAK

- Github repository oluşturma.
- ThreadConfig dosyasının oluşturulması, Entitylerin oluşturulması, Paketlerin oluşturulması

Barış Emre

- Global Exception Handling
- README.md dosyasının oluşturulması

Ağcanur KAYNAR

- FileProccessingService geliştirilmesi

Berk ÇIRAK

- ArchiveService geliştirilmesi

2. Kısım Görev Dağılımı

Umut Sefkan SAK

- Tespit edilen eksiklerin giderilmesi (eksik anotasyon, metot, interface vs.)
- ThreadManagementService geliştirilmesi

Barış Emre

- Geliştirilen service'lere Global Exception Handling entegrasyonu
- FileAnalysisService geliştirilmesi

Ağcanur KAYNAR

- Frontend geliştirilmesi için gerekli teknoloji araştırması

Berk ÇIRAK

- Hatalı branch isimlerinin düzeltilmesi
- Opsiyonel olan Unzip metodunun geliştirilmesi

3. Kısım Görev Dağılımı

Umut Sefkan SAK

- FileUploadService geliştirilmesi
- FileDownloadService geliştirilmesi
- FileUploadService ile ilgili endpointlerin eklenmesi
- FileDownloadService ile ilgili endpointlerin eklenmesi

Barış Emre

- Analiz işlemini yapacak Controller yazılması.
- Controller için gerekli DTO'ların oluşturulması
- Global ExceptionHandling entegrasyonunun devamı

Ağcanur KAYNAR

- Frontendin baştan sona geliştirilmesi

Berk ÇIRAK

- Rapor dosyasının hazırlanması
- README.md dosyasının hazırlanması