**Threads Management Proje Dokümantasyonu**

**1. Proje Hakkında Genel Bilgi**

* **Proje Adı**: Thread Management System
* **Açıklama**: Bu proje, kullanıcıların belirli sayıda gönderici (Sender) ve alıcı (Receiver) thread oluşturmasına, bu thread'lerin durumlarını yönetmesine ve queue yapısını takip etmesine olanak tanır.
* **Teknolojiler**:
  + **Frontend**: React
  + **Backend**: Spring Boot
  + **Veri Yapısı**: Java BlockingQueue
  + **Hedef Kitle**: Uygulama, thread yönetimi ve veri iletişimi senaryolarını öğrenmek ve simüle etmek isteyen kullanıcılar için tasarlanmıştır.
* **Github Linkleri**:
  + **Frontend**: https://github.com/zulalkarin/threads-frontend.git
  + **Backend**: https://github.com/zulalkarin/threads-backend.git

**2. Kurulum**

**Frontend (React)**

1. **threads-frontend proje dizimine gidin.**cd threads-frontend
2. **Gerekli bağımlılıkları yükleyin:**npm install
3. **Uygulamayı başlatın:**npm start

**4. Konfigürasyon**:

-config.js dosyasındaki BACKEND\_URL ve WEBSOCKET\_URL

değerinin doğru olduğundan emin olun.  
  
-Frontend uygulaması varsayılan olarak http://localhost:3000 adresinde çalışır.

**Backend (Spring Boot)**

1. **Threads-backend project dizimine gidin:**cd threads-backend
2. **Maven bağımlılıklarını yükleyin:**mvn clean install
3. **Uygulamayı başlatın:**mvn run:spring-boot  
   Backend uygulaması varsayılan olarak http://localhost:8080 adresinde çalışır.

**3. Kullanıcı Kılavuzu**

**Temel İşlevler**

1. **Thread Oluşturma**:
   * Kullanıcı arayüzünde "Sender" ve "Receiver" thread sayılarını seçin ve “Start Theads" butonuna basın.
2. **Thread Kontrolü**:
   * Thread tablosu üzerinden her bir thread'i durdurma, başlatma ve priority değiştirme işlemlerini gerçekleştirin.
3. **Queue Durumu İzleme**:
   * Queue'nun doluluk oranını çizgi grafiği üzerinden canlı olarak izleyebilirsiniz.

**4. API Entegrasyonu**

**- API İstekleri:**

|  |  |  |
| --- | --- | --- |
| **Yöntem** | **URL** | **Açıklama** |
| POST | api/threads/create?senderCount={count}&receiverCount={count} | Yeni thread'ler oluşturur. |
| PUT | api/threads/{id}/active?active={true|false} | Thread'in durumunu günceller. |
| PUT | api/threads/{id}/priority?priority={1-10} | Thread önceliğini değiştirir. |
| GET | api/threads | Tüm thread’leri listeler. |
| GET | Api/threads/active | Sadece active olan threadleri listeler |
| GET | api/queue/status | Queue'nun mevcut durumunu döner. (Doluluk oranı, size, max-size) |
| DELETE | api/threads | Tüm thread’leri siler. |
| DELETE | Api/quque/clear | Queue’yu temizler |

**- Websocet İletişimi:**

WebSocket üzerinden iki ana kanal dinlenir:  
 - Thread Güncellemeleri: **/topic/threads/updates** - Queue Durumu Güncellemeleri: **/topic/queue/updates**

**5. React Hooks**

**1. `useQueueStatus` Hook**

Amaç: Queue durumunu yönetir ve günceller.  
 İşlevler:  
 - fetchQueueStatus(): Queue durumunu API'den alır.  
 - clearQueue(): Queue'yu temizler.

**2. `useThreadControl` Hook**

Amaç: Thread'lerin durumunu ve önceliğini günceller.  
 İşlevler:  
 - updateThreadActive(threadId, active): Thread'in aktif durumunu günceller.  
 - updateThreadPriority(threadId, priority): Thread'in önceliğini günceller.

**3. `useThreads` Hook**

Amaç: Tüm thread'leri yönetir.  
 İşlevler:  
 - fetchThreads(): Tüm thread'leri API'den alır.  
 - createThreads(senderCount, receiverCount): Yeni thread'ler oluşturur.  
 - deleteThreads(): Tüm thread'leri siler.

**6. WebSocketService**

**Yöntemler**

* `connect()`: WebSocket bağlantısını başlatır.  
  - `addThreadHandler(handler)`: Thread güncellemelerini dinlemek için handler ekler.  
  - `removeThreadHandler(handler)`: Thread handler'ı kaldırır.  
  - `addQueueHandler(handler)`: Queue güncellemelerini dinlemek için handler ekler.  
  - `removeQueueHandler(handler)`: Queue handler'ı kaldırır.  
  - `disconnect()`: WebSocket bağlantısını kapatır.

**7. Ana Backend Yapılandırması  
  
config**

* **CorsConfig.java**: Uygulamanın CORS (Cross-Origin Resource Sharing) yapılandırmasını içerir.
* **SwaggerConfig.java**: Uygulamanın Swagger dokümantasyonu için yapılandırma dosyasıdır. API'ın interaktif belgelerinin oluşturulmasını sağlar.
* **WebSocketConfig.java**: WebSocket bağlantılarının kurulması, dinlenmesi ve mesajların iletilmesi için yapılandırma sağlar.

**controller**

* **QueueController.java**: Queue durumunu almak, temizlemek veya güncellemek için HTTP isteklerini yöneten endpoint'ler içerir.
* **ThreadController.java**: Thread'lerin oluşturulması, silinmesi, aktiflik durumunun değiştirilmesi gibi işlemleri yöneten endpoint'ler içerir.

**model**

* **ThreadInfo.java**: Thread bilgilerini temsil eden model sınıfıdır. Thread ile ilgili tüm verilerin (örneğin, durum, öncelik, alıcı ve gönderici sayısı) tutulduğu sınıftır.

**service**

* **QueueService.java**: Queue'nun durumunu almak, temizlemek gibi işlemleri gerçekleştirir.
* **ThreadManagerService.java**: Thread'lerin oluşturulması, güncellenmesi ve silinmesi işlemlerini yönetir.

**thread**

* **CustomThread.java**: Thread'ler için özelleştirilmiş mantıkların bulunduğu sınıftır.
* **ThreadType.java**: Thread'lerin türlerini tanımlayan bir enum.

**ThreadManagementApplication.java**

* Uygulamanın başlatılmasını sağlayan ana sınıf.

**8. Hata Yönetimi**Her API isteği ve WebSocket işlemi hata yönetimi ile korunur. Kullanıcıya anlamlı hata mesajları gösterilir.

**9. Testler**

Backend testleri yazılmıştır ve aşağıdaki komut ile çalıştırılabilir:

mvn test  
 **10. Geliştirici Bilgileri**

* **Hazırlayan**: Zülal Karın
* **Tarih**: 10/11/2024
* **İletişim**: zülal.karin19@gmail.com

**11. Sonuç**  
Bu proje, thread yönetimi ve queue durumu gibi süreçleri dinamik olarak yöneten ve kullanıcı dostu bir arayüz sunan bir uygulamadır. WebSocket ile anlık veri alımı ve API ile güvenilir veri yönetimi sağlar.