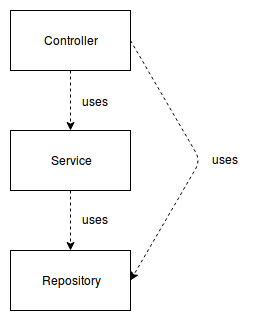
**RickAndMorty Uygulaması**

Demo uygulama için seçilen Rick And Morty çizgi filminin api'si

(GET - https://rickandmortyapi.com/api/ ) seçilmiştir. Bu api üzerinde bulunan tüm endpoint’leri RickAndMorty uygulaması da içermektedir.

* Java 11 ve MongoDB 4 kullanılmıştır.
* Uygulama Spring Boot ile yapılmıştır. **8080** portunda çalışmaktadır.
* Uygulamada Layered Architecture yapısı kullanılmıştır.



* Spring Data (JPA) kullanılmıştır. MongoDB ile her bir document için “com/egemsoft/rickandmorty/domain” klasöründe bir **class** oluşturulmuştur. Veritabanı bağlantıları için ise “com/egemsoft/rickandmorty/repository” klasöründe her bir dökümanın **repository**’leri oluşturulmuştur.
* MongoDB için Mongo Atlas’tan bir cluster açılmıştır. Herhangi bir mongodb kurulumu gerektirmemektedir. Aşağıda verdiğim URI ile MongoDB Compass üzerinen izlenilebilir. **Boş bir veritabanıyla** çalışmaya başlayacaktır.

MongoDB Compass İndirme Linki: <https://www.mongodb.com/try/download/compass>

Mongo Atlas Cluster URI: mongodb+srv://**testuser:testuser@**cluster0-fyb4u.mongodb.net/RickAndMorty

MongoDB için 2 profil oluşturuldu;

1.local -> localhost'ta çalışan bir MongoDB varsa

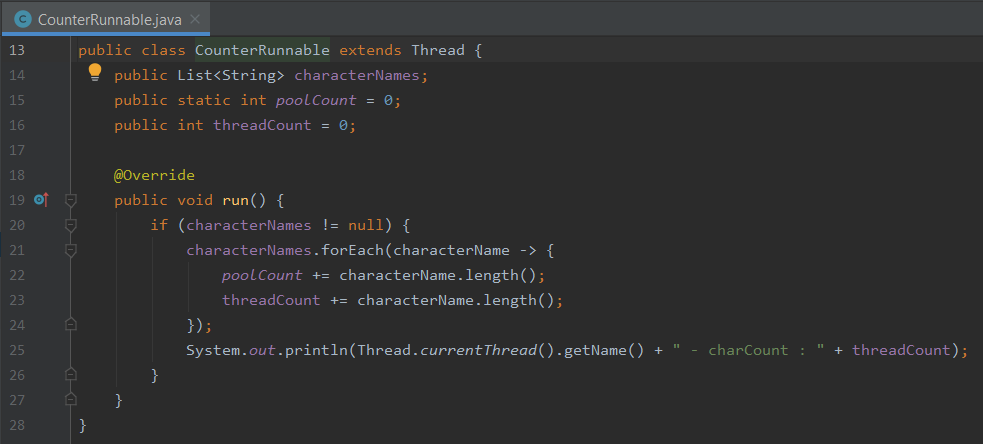
2.test -> Mongo Atlas'ta bir cluster'a bağlanıp otomatik başlar. Compiling 1 dakikayı bulabilir

* Verilen api örnek alındığından oluşan tüm endpointler ve sonuçları **exception**’ları ile beraber uygulamada bulunmaktadır.
* Javadoc’lar dökümanın devamı olarak okunabilir
* Hata yönetimleri ve uygulanan işlemler ve için bazı yerlerde **org.slf4j.Logger** kullanılmıştır. Console izlenilebilir
* Thread örneği için bir endpoint yaratılmamıştır. Thread ile ilk defa uğraştığım için basit bir örnek yaptım. Endpoint olmadığı için uygulama başlarken apiden tüm veriler çekildikten sonra Thread’ler çalıştırılmaya başlar.

“com/egemsoft/rickandmorty/thread/CharacterNameCounterExecuterService.java”

adresindeki class içerisinde tüm Character objelerinin “name” alanlarını listesini çekip

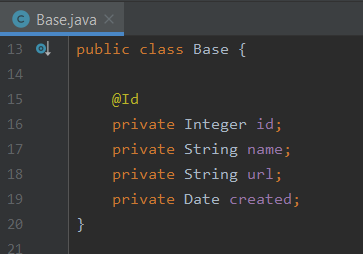
25 eşit büyüklükte listeye böldükten sonra her bir listeye 1 Thread açtım. Her bir thread aşağıdaki ekran görüntüsünde gösterilmiştir.



bitane static bitane de kendi count’unu tutmaktadır. Ayrıca uygulama başlarken çalıştığı için apiden verilerin çekilmesini bekler. Verileri çekme işlemi tamamlanınca karakter sayıları toplanır, console’a yazılır. Uygulama açıldığında console üzerinden incelenebilir.

Uygulamadaki adımlar aşağıdaki gibi olmalıdır.

1. Modelleme
   1. GET - https://rickandmortyapi.com/api/ api yolu ile belirtilmiş olan adresten oluşacak bütün endpointlerin response tipleri modellenmiştir..
   2. Database için MongoDB kullanılmıştır. Api üzerinden oluşan tüm Dökümanlar için ortak olan 4 değer için ayrı bir **Base** class oluşturulmuştur.



Character, Episode ve Location class’ları bu class’a extend edilmiştir.

1. Data hazırlık
   1. Uygulama başlarken belirtilen api üzerinden tüm kaynağı çekip veritabanına kaydetmektedir. Bu işlem uygulama başlarken ilk sırada çalışmaktadır. Uygulama her başlatıldığında bu datalar yenilenmektedir.
   2. Data çekilirken Scheduler çalıştırmaktadır. Scheduler ile her gece saat 00:00’da tüm apiyi tekrar çekip update etmektedir. Update işlemi api’den silinmiş dosyaları silme işlemi yapmamaktadır. Detaylı bilgi için “com/egemsoft/rickandmorty/service/InitService.java” servisi incelenebilir. Ayrıca GET - <localhost:8080/api/systemInfo> ile veritabanının son güncellendiği tarih görüntülenebilir.
   3. Aşağıdaki yazılacak endpointler istek anında “rickandmortyapi” apisini tetikleyip gelen cevapları işlemeyecektir. Data hazırlık aşamasında hazırlanmış olan datalardan cevap vermektedir.
2. RestFull WebService

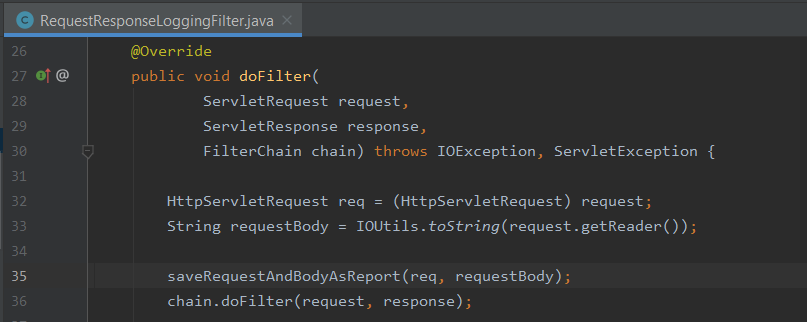
Bu gereksinim için “/thread-demo” hariç hepsi yapılmıştır. Hepsinde “orderBy” query parametresi ile sıralanacağı alan verilerek sıralanabilir. Örneğin;

GET - <http://localhost:8080/api/character/?page=1&orderBy=name> isteği ile sıralanmış bir 20 adet Character içeren 1. page dönmektedir. “page” ve “orderBy” parametreleri required değildir.

– Thread

* 1. Character
     1. /character
        1. Yukarıda belirtildiği gibi bir endpoint açılmalıdır. Sisteme kayıtlı tüm kayıtların listesini dönmelidir.
        2. Örnek alınan apideki gibi pagination yapısı bulunması artı sebebidir.
        3. Endpoint içerisine sıralama için bir parametre belirtilebilmelidir. Bu parametre ile gelen kayıtlar isme göre veya oynadığı bölüm sayısına göre sıralanabilmelidir. Bu alan zorunlu olmamalıdır.
     2. /character/{id}
        1. Yukarıda belirtildiği gibi bir endpoint açılmalıdır. Sisteme kayıtlı belirtilen karakterin bilgilerini dönmelidir.
  2. Episode
     1. /episode
        1. Yukarıda belirtildiği gibi bir endpoint açılmalıdır. Sisteme kayıtlı tüm kayıtların listesini dönmelidir.
        2. Örnek alınan apideki gibi pagination yapısı bulunması artı sebebidir.
        3. Endpoint içerisine sıralama için bir parametre belirtilebilmelidir. Bu paramerte ile gelen kayıtlar isme göre veya oynayan karakter sayısına göre sıralanabilmelidir. Bu alan zorunlu olmamalıdır.
     2. /episode/{id}
        1. Yukarıda belirtildiği gibi bir endpoint açılmalıdır. Sisteme kayıtlı belirtilen episodların bilgilerini dönmelidir.
  3. Location
     1. /location
        1. Yukarıda belirtildiği gibi bir endpoint açılmalıdır. Sisteme kayıtlı tüm kayıtların listesini dönmelidir.
        2. Örnek alınan apideki gibi pagination yapısı bulunması artı sebebidir.
        3. Endpoint içerisine sıralama için bir parametre belirtilebilmelidir. Bu paramerte ile gelen kayıtlar isme göre veya oynayan karakter sayısına göre sıralanabilmelidir. Bu alan zorunlu olmamalıdır.
     2. /location/{id}
        1. Yukarıda belirtildiği gibi bir endpoint açılmalıdır. Sisteme kayıtlı belirtilen locationların bilgilerini dönmelidir.
  4. Report
     1. /report
        1. Endpointler raporlanabilir olmalıdır. Bu endpoint ise bu raporları gösteren bir endpoint olacaktır. Response olarak:
        2. Hangi endpoint’e toplamda kaç istek geldi?
        3. Gelen/giden header ve body’leri nelerdir? Bunların detaylarını zamanları ile birlikte bir history şeklinde göstermelidir.
  5. SystemInfo
     1. /systemInfo
        1. SystemInfo class’ında bir obje döner. Sistemin veritabanı güncellenme tarihini tutar.

1. Report oluşturma için kullanılması gereken header ve body’leri çekme gereksinimi için **Filter** kullanılmıştır. Bu filter aşağıdaki ekran görüntüsünde verilmektedir.



Bu filter içerisinde yakalanan her bir isteğin hangi controller’a gittiğine bakılarak farklı ResponseType’a sahip Report objeleri oluşturulmaktadır. Objelerin ve ilgili fonksiyonların tamamı açıklayıcı şekilde yazılmıştır. “saveRequestAndBodyAsReport(req,requestBody)” fonksiyonu izlenerek ilgili adımlar incelenebilir.

**Eksik Gereksinimler:**

* “/thread-demo” endpoint
* Yazılan endpointlerin karmaşıklıklıkları(BigO) yorumlar halinde javadoc’a eklenmelidir.