Ad Soyad: Burcu Başak

Tel No: 05342702939

E-mail Adresi: burcucalgnn@gmail.com

CASE CEVAPLAR

1-MVC, Model-View-Controller'ın kısaltmasıdır. Bu tasarım deseni, bir uygulamanın yapısını organize etmek için kullanılır.

Model (Model): Veri ve iş mantığıyla ilgilenir. Veri üzerinde değişiklik yapar ve bu değişiklikleri View ve Controller'a bildirir.

View (Görünüm): Kullanıcı arayüzü ile ilgilenir. Model'in verilerini kullanarak kullanıcıya bilgi gösterir. Model'den gelen değişiklikleri algılar ve güncellenir.

Controller (Denetleyici): Kullanıcının girişlerini işler, Model'i günceller ve View'i güncellenmesi için yönlendirir. Model ve View arasındaki iletişimi sağlar.

MVC'nin ihtiyaç duyulmasının temel nedenleri şunlar olabilir:

Her bir bileşen kendi sorumluluklarına odaklanır, bu da kodun daha temiz ve sürdürülebilir olmasını sağlar.

Yeniden Kullanılabilirlik Model, View ve Controller'ın bağımsız olması, bunları başka projelerde veya farklı bağlamlarda yeniden kullanmayı kolaylaştırır.

Java'da MVC kurgusu genellikle şu şekilde olur:

Model: Java'da genellikle POJO (Plain Old Java Object) sınıflarını temsil eder. Veritabanı işlemleri, veri manipülasyonları gibi işlemler burada gerçekleşir.

View: Kullanıcı arayüzü bileşenleri, genellikle Swing veya JavaFX gibi GUI kütüphaneleri kullanılarak oluşturulur.

Controller:Java'da genellikle bir sınıf veya servlet olarak uygulanır. Kullanıcının girişlerini alır, Model'i günceller ve View'i güncellemesi için yönlendirir.

Object Oriented katmanları şunları içerebilir:

Veri Erişim Katmanı (Data Access Layer): Veritabanı ile iletişim kurar, veri erişim operasyonlarını gerçekleştirir.

İş Mantığı Katmanı (Business Logic Layer): Uygulama özel iş mantığını içerir, genellikle servis sınıfları veya iş mantığı sınıfları olarak ifade edilir.

Kullanıcı Arayüzü Katmanı (User Interface Layer): Kullanıcı ile etkileşimi sağlar, View bileşenlerini içerir.

Her katman, belirli bir amaca hizmet eder ve birbirinden bağımsız olmalıdır, bu da uygulamanın modüler ve bakımı kolay olmasını sağlar.

2-Birbirinden bağımsız iki platform arasında iletişim kurmak için genellikle birkaç yaygın yaklaşım vardır:

API (Application Programming Interface):Her iki platform üzerinde birer API oluşturarak, bu API'lar aracılığıyla iletişim sağlayabilirsiniz. RESTful API veya SOAP gibi standart protokoller kullanarak, uygulamalar arasında veri alışverişi gerçekleştirilebilir. Bu durumda, X platformundaki Java ve Y platformundaki C# uygulamaları, belirlenen API üzerinden birbirleriyle haberleşebilir.

Veri Serileştirme ve Deserileştirme: İki platform arasında veri alışverişi için ortak bir veri formatı belirlenir, örneğin, JSON veya XML. Her iki platform da bu formatta veriyi seri hale getirip (serialize) ve karşı tarafta bu veriyi alıp orijinal haline getirir (deserialize). Bu sayede, iki farklı dilde yazılmış uygulama arasında veri paylaşımı sağlanabilir.

Mesaj Kuyrukları (Message Queues):İki platform arasında asenkron iletişim sağlamak için mesaj kuyrukları kullanılabilir. X platformundaki Java uygulaması bir mesaj kuyruğuna mesaj gönderir, ve Y platformundaki C# uygulaması bu kuyruktan mesajları alır. Bu sayede, uygulamalar birbirleriyle doğrudan haberleşmese de, mesajlar aracılığıyla etkileşim sağlanabilir.

Hangi yöntemin kullanılacağı, projenizin ihtiyaçlarına ve gereksinimlerine bağlıdır. API tabanlı iletişim genellikle geniş kullanıma sahiptir, veri serileştirme ve deserializasyon daha basit senaryolarda kullanılabilir, mesaj kuyrukları ise asenkron iletişim gereksinimlerini karşılayabilir.

3-Bu tür anlık güncellemeleri sağlamak için genellikle iki temel yöntem kullanılır:

AJAX (Asynchronous JavaScript and XML):Bu teknikle, JavaScript kullanarak sayfa üzerindeki belirli bir bölgeyi dinamik olarak güncelleyebilirsiniz. JavaScript, arka planda sunucu ile iletişim kurarak veri alabilir ve bu veriyi kullanarak sayfa içeriğini güncelleyebilir. Modern web uygulamalarında XML yerine JSON sıkça kullanılır. Bu sayede sayfa yeniden yüklenmeden değişiklikleri anında gösterebilirsiniz.

WebSockets: Bu yöntem, bir sürekli bağlantı (persistent connection) sağlar ve sunucu ile istemci arasında çift yönlü iletişimi mümkün kılar. WebSockets kullanarak, sunucu tarafında yapılan güncellemeler anında istemcilere iletilir. Bu sayede, sayfa yeniden yüklenmeden gerçek zamanlı güncellemeler elde edilebilir.

Hangi yöntemin kullanılacağı, projenizin gereksinimlerine ve karmaşıklığına bağlıdır. AJAX, basit güncellemelerde genellikle yeterli olabilirken, daha karmaşık ve hızlı güncellemeler için WebSockets gibi teknolojilere başvurabilirsiniz.

4- java

```
for (int i = 1; i <= 5; i++) {
   for (int j = 1; j <= 2 * i - 1; j++) {
      System.out.print("*");
   }
   System.out.println();
}
```

5-Erişim testi yapmak için Terminal veya SSH (Secure Shell) kullanabilirsin. Öncelikle, aşağıdaki adımları izleyebilirsin:

Erişim Testi:

- Terminal veya komut istemcisini aç.
- ssh komutunu kullanarak server'a bağlan. Örneğin:

  ssh kullanici_adi@ip_adresi -p port

  Burada "kullanici_adi" yerine gerçek kullanıcı adını, "ip_adresi" ve "port" değerlerini kullanmalıyız.
- Şifreyi girerek server'a bağlanılır.

Dosya Aktarımı:

- Dosya atmak için scp (Secure Copy) komutunu kullanılabilir. Örneğin:

  scp dosya.txt kullanici_adi@ip_adresi:/uzak/dizin

  Burada "dosya.txt" yerine göndermek istediğin dosyanın adını, "/uzak/dizin" kısmını ise dosyanın nereye kopyalanacağını belirtilmeli.
- Dosya çekmek için de yine scp komutunu kullanılabilir.. Örneğin:

  scp kullanici_adi@ip_adresi:/uzak/dizin/uzak_dosya.txt .

  Burada "/uzak/dizin/uzak_dosya.txt" dosyasını mevcut dizine çeker.Bu adımlar, server'a SSH ile erişim sağlamak, dosya atmak ve dosya çekmek için kullanılan temel yöntemlerdir. Bu işlemleri gerçekleştirirken güvenlik önlemlerini unutmamalı ve gerektiğinde server yöneticisi ile iletişime geçerek izin alınmalı.

6-

- Projeyi IntellijIDEA kullanarak açtım.
- Spring Initializr kullanarak bir Spring Boot projesi oluşturdum.
- İçerisinde `Spring Web` dependency eklendim.
- Maven dependency management sistemini kullanarak tüm dependencyleri install ettim.
- Postgresql kullanarak database application properties bölümünden proje bağladım.
- Uygulamanızı `8080` portundan ayağa kaldırdım.
- Bir çalışan iş veren uygulaması için rest api dizayn ettim.

- Veri tabanı dizaynımda toplam 2 adet tablo yazdım.
- `Employee` tablosu => (id, isMarried, salary, firstName, gender, lastName,employerId) kolonlarından oluşturdum.
- `id` primary_key olarak işaretledim.
- `Employer` tablosu => (id, ad, last_name, adress) bilgilerini girdim..
- Her çalışan sadece bir tane iş verene sahip olabilir.
- `id` primary_key olarak işaretlendi..
- Bir iş verenin birden çok çalışanı olabilir

- `controller`, `service`, `repostory`, `entity`, isminde 4 adet daha paket oluşturdum.
- `entity` paketinin altına `employee`,`employer`,`gender` isimli üç tane sınıf oluşturdum.
- `Employee`, `employer` sınıfları project lombok annotationlarını kullanarak oluşturdum.
- `Employee` ve `employer` sınıflarının birbirleri arasındaki ilişkiyi tanımlayan annotationları kullandım.(One-to-One,One-to-Many)
- `AddressController`, `CustomerController`, `AccountController` isimli 3 tane controller yazmalısınız.

- Amacım CRUD işlemlerini tanımlayan endpointler yazmak.

- [GET]/case/employee => veri tabanaında kayıtlı tüm employee bilgilerini döner.

- [GET]/case/employee/{id} => İlgili id deki employee objesini döner.

- [POST]/case/employee => Bir adet employee objesini veri tabanına ekler.

- [DELETE]/ case/employee /{id} => İlgili id değerindeki employee objesini siler.

- Bu işlerimlerin tamamı employer içinde yapılır.

Projeme https://github.com/LadyDementor/ChallangeCaseBackend linkinden ulaşabilirsiniz.

7-Solr sorguları için genellikle Lucene sorgu dilini kullanırsınız. Eğer "updatedAt" adında bir alanı ve bu alandaki değerleri tarih cinsinden temsil eden long tipinde bir alanı olduğunu düşünüyorsak, "updatedAt" alanındaki değerlerin 2020 Ocak ayından sonraki tarihleri içermesini istiyorsak bir sorgu şu şekilde olabilir:

plaintext

http://example/solr/core/select?q=updatedAt:[2020-01-01T00:00:00Z TO *]

Bu sorgu, "updatedAt" alanının 2020 Ocak ayından sonraki tarihleri içeren belgeleri getirecektir. Tarih aralığı [2020-01-01T00:00:00Z TO *] ifadesi ile belirtilmiştir.

Burada dikkat edilmesi gereken noktalar:

- http://example/solr/core/select kısmı Solr sorgu servisinin URL'sini temsil eder. "core" kısmı kullanılan indeksi ifade eder. Bu kısmı kendi Solr kurulumunuzdaki yapıya göre düzenlemeniz gerekebilir.

- q=updatedAt:[2020-01-01T00:00:00Z TO *] kısmı sorgu stringidir. Burada updatedAt alanındaki değerlerin belirtilen tarih aralığı içinde olup olmadığını kontrol eder.

Not: Bu örnek sorgu Lucene syntax'ını kullanmaktadır. Gerçek duruma göre Solr sorgularınızı daha spesifik hale getirmeniz gerekebilir.