Spring

Spring, açık kaynaklı (open source), ücretsiz, modüler bir yapıda olan ve Java ile geliştirme yapmayı kolaylaştıran frameworktür. Spring'in çözdüğü problemler arasında bağlantı yönetimi, veritabanı etkileşimi, güvenlik, bağımlılık enjeksiyonu, konfigürasyon yönetimi, ve MVC (Model-View-Controller) gibi pek çok konu bulunmaktadır.

Bağımlılık Enjeksiyonu (Dependency Injection):
Spring, bağımlılıkları enjekte etmek ve nesne yaşam döngüsünü yönetmek için IoC (Inversion of Control) prensibini kullanır. Bu, nesneler arasındaki bağımlılıkları azaltır ve test edilebilir, esnek kod yazmayı sağlar.

```
@Service
public class MyService {
    private MyRepository repository;

   @Autowired
    public MyService(MyRepository repository) {
        this.repository = repository;
    }
}
```

Spring MVC:
Spring MVC, Model-View-Controller mimarisine dayanan bir web uygulama geliştirme çerçevesidir. Bu, kullanıcı arabirimi tasarımını ve iş mantığını ayırarak uygulama geliştirmeyi kolaylaştırır.

```
@Controller
public class MyController {
    @GetMapping("/hello")
    public String hello(Model model) {
        model.addAttribute("message", "Hello World!");
        return "hello-page";
    }
}
```

Veritabanı Etkileşimi:
Spring Data JDBC ve Spring Data JPA gibi modüller aracılığıyla, veritabanı işlemleri daha kolay hale getirilir. Bu, SQL sorgularını azaltır ve nesne ilişkisel eşleme (ORM) sağlar.

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private double price;
}
```

Güvenlik:
Spring Security, kullanıcı kimlik doğrulama, yetkilendirme ve diğer güvenlik konularını ele almak için kullanılır.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/public/**").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }
}
```

JSF(Java Server Faces)

JavaServer Faces (JSF), Java tabanlı bir web uygulama çerçevesidir ve kullanıcı arayüzü bileşenlerini geliştirmek, yönetmek ve sunmak için kullanılır.

Komponent Tabanlı Geliştirme:
JSF, kullanıcı arayüzü bileşenleri (component) üzerine odaklanarak geliştirme yapmayı sağlar. Bu, kullanıcı arayüzü öğelerini daha kolay yönetmeyi ve tekrar kullanmayı mümkün kılar.

```
  <h:commandButton value="Gönder" action="#{myBean.submit}" /
```

Bağlam (Binding) ve Olay Yönetimi:
JSF, kullanıcı arayüzü bileşenleri ve arka uygulama kodu arasında bağlantı kurmayı sağlar. Böylece, bir bileşenin değeri veya olayları değiştiğinde arka taraftaki işlemi kolayca gerçekleştirebiliriz.

```
<h:inputText value="#{myBean.userName}" />

@ManagedBean
public class MyBean {
    private String userName;
}
```

Form Doğrulama:
JSF, kullanıcıdan gelen form verilerini doğrulamak için kolaylık sağlar. Bu, giriş kontrolleri, zorunlu alanlar ve özel doğrulama kuralları gibi işlemleri gerçekleştirmemizi sağlar.

```
<h:inputText id="username" value="#{myBean.userName}" required="true" />
<h:message for="username" />
```

Navigation Yönetimi:
JSF, kullanıcıların bir sayfadan diğerine geçişini kolaylaştıran bir navigasyon yönetimi sağlar. Belirli bir olay gerçekleştiğinde veya bir işlem tamamlandığında farklı sayfalara yönlendirme yapabiliriz.

```
<h:commandButton value="Devam" action="#{myBean.nextPage}" />

public String nextPage() {
    return "success";
}
```

Internationalization (i18n) ve Localization (l10n):
JSF, çok dilli uygulamalar geliştirmek için entegre bir şekilde çalışır. Mesajları, etiketleri ve diğer metinleri çoklu dilde yönetmek kolaydır.

```
<h:outputText value="#{msg.welcome}" />

# messages.properties dosyasında mesaj örneği
welcome=Hoş geldiniz!
```

Hibernate

Hibernate, Java tabanlı bir ORM (Object-Relational Mapping) frameworkudur ve ilişkisel veritabanı sistemleri ile nesne modelleri arasındaki eşleştirmeyi kolaylaştırarak bir dizi problemi çözer.
Veritabanı İlişkilerinin Yönetimi:
Hibernate, Java sınıfları ile veritabanı tabloları arasında doğrudan eşleştirmeyi sağlar. Bu sayede, ilişkisel veritabanındaki tablolar ve veri yapıları arasında kolay bir geçiş yapabiliriz.

```
@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
}
```

Bağlantı Yönetimi:
Hibernate, veritabanı bağlantılarını otomatik olarak yönetir ve gerekli kaynakları açar veya kapatır. Bu, geliştiricilerin bağlantı yönetimi ile ilgilenmelerine gerek kalmadan veritabanı işlemlerini gerçekleştirmelerini sağlar.

```
SessionFactory sessionFactory = new
Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();
transaction.commit();
session.close();
```

HQL (Hibernate Query Language) Kullanımı:
Hibernate, nesne odaklı sorgulama yapmak için HQL kullanır. Bu, geliştiricilere SQL yerine Java sınıflarını ve nesnelerini kullanarak veritabanı sorguları oluşturma imkanı tanır.

```
Query query = session.createQuery("FROM Product WHERE category =
:category");
query.setParameter("category", "Electronics");
List<Product> products = query.list();
```

İkinci Seviye Önbellekleme (Second Level Caching):
Hibernate, ikinci seviye önbellekleme kullanarak veritabanı erişimini optimize eder. Bu, sık kullanılan verilerin bellekte saklanarak performans artışı sağlar.

```
  <property name="hibernate.cache.use_second_level_cache">true</property>
  <property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCach
eRegionFactory
  </property>
```

Apache Struts

Apache Struts, Java tabanlı web uygulamaları geliştirmek için kullanılan bir web uygulama frameworkudur. Struts, özellikle Model-View-Controller (MVC) mimarisini benimseyerek, web uygulamalarının daha organize ve sürdürülebilir bir şekilde geliştirilmesine yardımcı olur.

MVC Tasarım Deseni:
Struts, Model-View-Controller (MVC) tasarım desenini destekler. Bu, uygulamanın farklı katmanlara (veritabanı işlemleri, iş mantığı, kullanıcı arayüzü) ayrılmasını sağlar.

```
  public class LoginAction extends Action {
     public ActionForward execute(ActionMapping mapping, ActionForm form,
                                  HttpServletRequest request, HttpServletResponse response)
{
        return mapping.findForward("success");
     }
  }
```

Form Nesneleri:
Struts, kullanıcı girişlerini almak ve işlemek için form nesnelerini kullanır. Bu, gelen verilerin doğrulanmasını ve işlenmesini kolaylaştırır.

```
  public class LoginForm extends ActionForm {
     private String username;
     private String password;
     //getter ve setter methodları
  }
```

Veri Doğrulama:
Struts, form nesneleri üzerinde veri doğrulama (validation) işlemlerini destekler. Bu, kullanıcı girişlerinin doğrulanmasını ve hatalı girişlerin ele alınmasını sağlar.

```
  public ActionErrors validate(ActionMapping mapping, HttpServletRequest request)
{
    ActionErrors errors = new ActionErrors();
    if (getUsername() == null || getUsername().length() < 1) {
        errors.add("username", new ActionMessage("error.username.required"));
    }
    return errors;
  }
```

Apache Wicket

Apache Wicket, Java tabanlı bir web uygulama çerçevesidir ve özellikle component-tabanlı geliştirme ve stateful web uygulamaları için tasarlanmıştır.

Component-Tabanlı Geliştirme:
Wicket, web uygulamalarını component (bileşen) tabanlı bir şekilde geliştirmemize olanak tanır. Her sayfa veya parça, bir ya da daha fazla bileşen içerir. Bu, bileşenlerin tekrar kullanılabilir olmasını sağlar.

```
  public class HelloComponent extends WebComponent {
    public HelloComponent(String id, IModel<String> model) {
        super(id, model);
        add(new Label("message", model));
    }
  }

  <span wicket:id="message">Mesaj burada görünecek</span>
```

Stateful (Durum Saklayan) Mimarisi:
Wicket, web sayfaları arasında durumu (state) otomatik olarak saklar. Böylece, uygulamanızın durumunu yönetmek ve kullanıcının geçmiş etkileşimlerini takip etmek daha kolay hale gelir.

```
  public class MyForm extends Form<Void> {
    private String userInput;

    public MyForm(String id) {
        super(id);
        add(new TextField<>("userInput", new PropertyModel<>(this, "userInput")));
    }

    @Override
    protected void onSubmit() {
```

```
        System.out.println("Kullanıcı girdisi: " + userInput);
    }
}

<form wicket:id="myForm">
    <input type="text" wicket:id="userInput" />
    <input type="submit" value="Gönder" />
</form>
```

Dropwizard

Dropwizard, Java tabanlı web hizmetleri ve mikro servis uygulamaları geliştirmek için kullanılan bir frameworktur. Dropwizard, uygulama geliştirme sürecinde karşılaşılan bazı yaygın problemleri çözmeyi amaçlar.

Uygulama Konfigürasyonu:
Dropwizard, YAML dosyaları üzerinden uygulama konfigürasyonunu kolayca yönetir. Bu, uygulamanın yapılandırılabilir olmasını ve farklı ortamlar için uygun konfigürasyonların kullanılabilmesini sağlar.

```
server:
  applicationConnectors:
    - type: http
      port: 8080
  adminConnectors:
    - type: http
      port: 8081
```

HTTP Hizmeti Sunma:
Dropwizard, Jetty entegrasyonu ile HTTP hizmeti sunmayı kolaylaştırır. Jersey frameworku ile entegre olmuş bir RESTful servis sunma yeteneği sağlar.

```
@Path("/hello")
public class HelloResource {
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayHello() {
        return "Hello, Dropwizard!";
    }
}
```

Loglama ve Günlük Kaydı:
Dropwizard, SLF4J ve Logback gibi popüler loglama kütüphaneleriyle entegre olmuştur. Bu sayede uygulama üzerindeki logları etkili bir şekilde yönetebiliriz.

```
  private static final Logger LOGGER =
LoggerFactory.getLogger(MyResource.class);

  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String getSomething() {
     LOGGER.info("İşlem başarıyla gerçekleştirildi.");
     return "Result";
  }
```