BO·TT·EGA
IT minds

Spring

...

Brown Brothers Harriman
Kraków
20-22.11.2017

BO·TT·EGA
IT minds

Agenda

1. REST/Spring MVC
2. Spring AOP
3. Aplikacja szkoleniowa do rozwiązywania zadań
4. Spring Security
5. Architektura zorientowana na zdarzenia i pluginy
6. JMS
7. Async
8. Spring Data
9. Transakcje
10. Profile uruchomieniowe aplikacji

BO·TT·EGA
IT minds

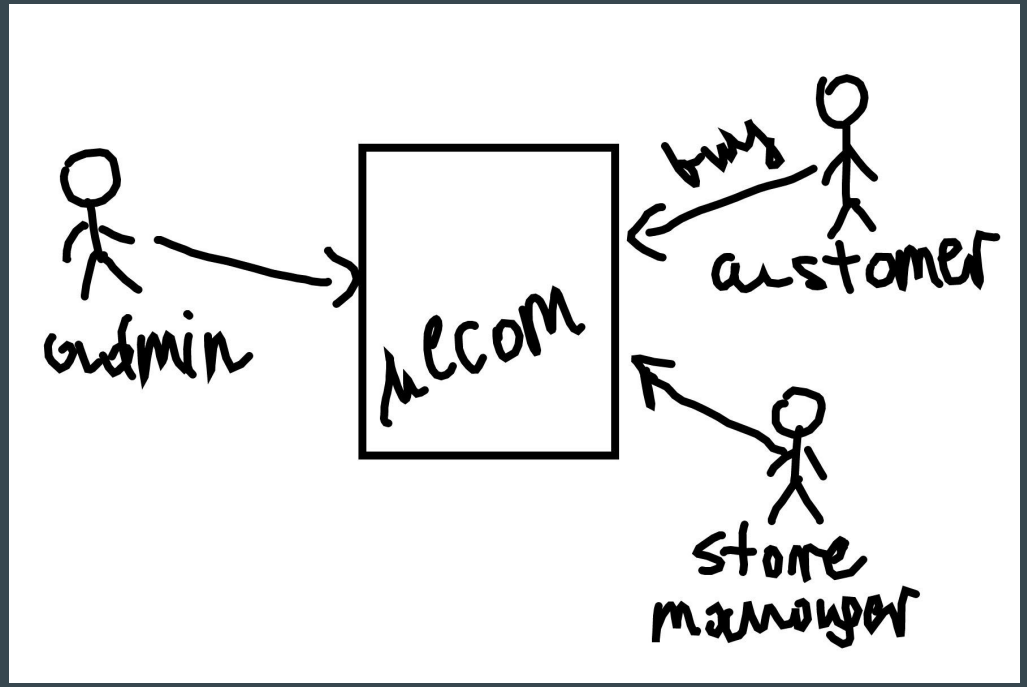
Przedstawienie siebie

- doświadczenie zawodowe
- znane technologie
- projekty
- znajomość zagadnień z agendy
- oczekiwania odnośnie szkolenia
- oczekiwania odnośnie prowadzącego

BO·TT·EGA
IT minds

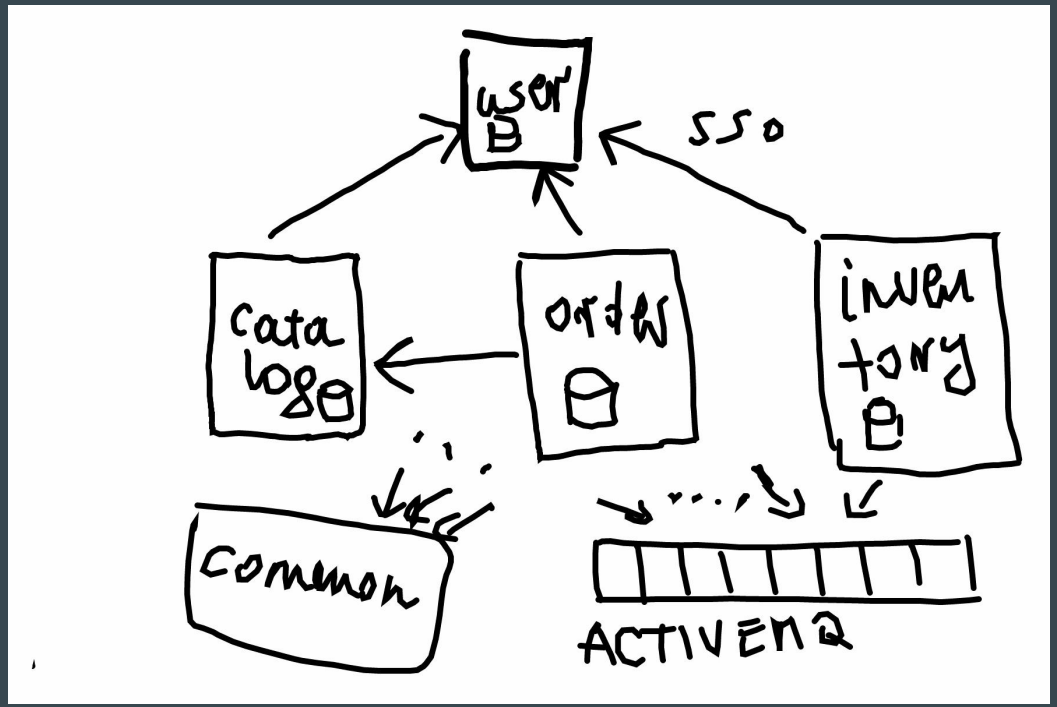
Aplikacja szkoleniowa do rozwiązywania zadań

Architektura od strony użytkownika



Aplikacja szkoleniowa do rozwiązywania zadań

Mikroserwisy

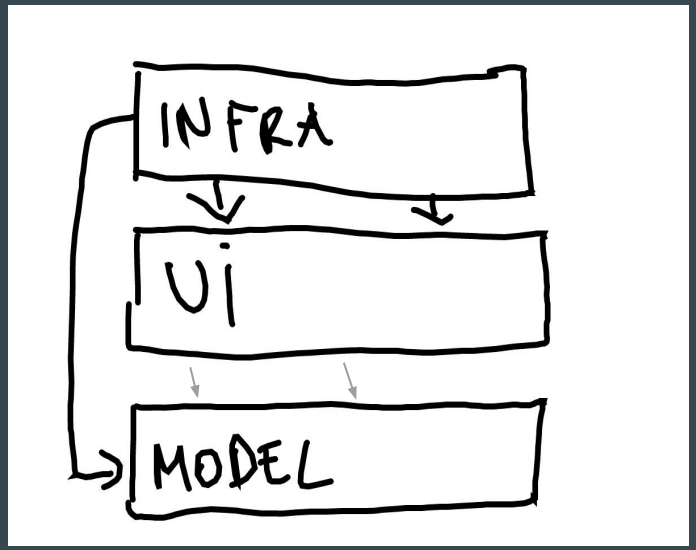


BOTTEGA
IT minds

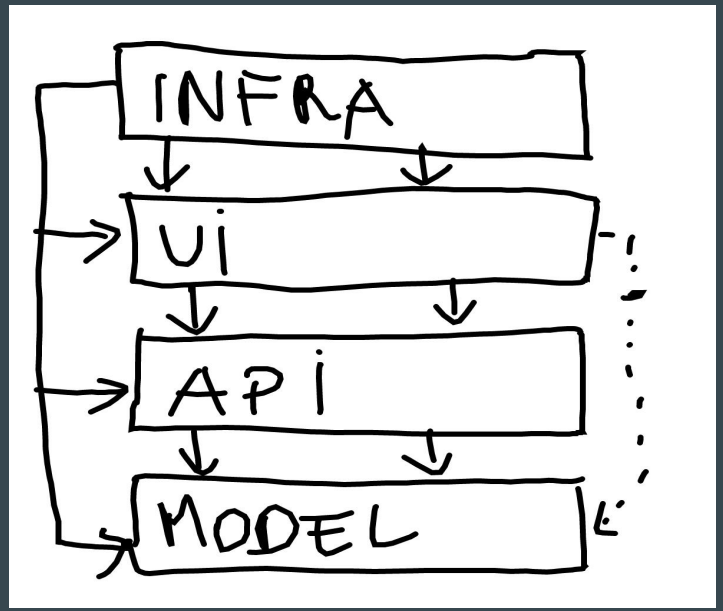
Aplikacja szkoleniowa do rozwiązywania zadań

Architektura pojedynczych serwisów

CRUD



Rich Domain Model



Spring Security - features

- oficjalny moduł springa
- autentykacja i autoryzacja (role based)
- web security
  - servlet api
  - spring mvc
- method security, domain object security ACL
- zabezpieczenia przed atakami
  - CSRF
  - Session Fixation
  - password encoding
- różne strategie autentykacji
  - inMemory, HTTP Basic, JDBC, LDAP, CAS, OAuth, OpenID, custom
- testability

BOTTEGA
IT minds

Spring Security - zależności

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-ldap</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-cas</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-openid</artifactId>
</dependency>
```

Spring Security - Core

1. SecurityContextHolder, ThreadLocal
2. SecurityContext
3. AuthenticationManager
4. Authentication
5. GrantedAuthority
6. UserDetailsService, UserDetails

Spring Security - Autentykacja

1. Użytkownik podaje login i hasło
2. System pomyślnie weryfikuje hasło dla podanego loginu lub zwraca błąd autentykacji
3. System pobiera informacje o uprawnieniach użytkownika
4. System ustawia kontekst zabezpieczeń dla zalogowanego użytkownika
5. Użytkownik używa aplikacji, system sprawdza uprawnienia użytkownika do wykonywania poszczególnych operacji (autoryzacja)

1. `Authentication`
2. `AuthenticationManager waliduje Authentication`
3. `AuthenticationManager zwraca instancję Authentication ze wszystkimi detalami użytkownika`
4. `SecurityContextHolder.getContext().setAuthentication(… )`

Spring Security - Autentykacja Web

1. Użytkownik odwiedza stronę www
2. Użytkownik klika w link wymagający autoryzowanego dostępu
3. Serwer odsyła komunikat mówiący o konieczności zalogowania (HTTP 401 lub redirect)
4. Przeglądarka pobiera w jakiś sposób credentiale
5. Przeglądarka wysyła żądanie autentykacji
6. Serwer sprawdza poprawność credentiali, ew. wracamy do punktu 3
7. Oryginalny request który wymagał autoryzacji jest powtarzany (ew. HTTP 403)

BO·TT·EGA
IT minds

Spring Security - Request Filtering

1. ChannelProcessingFilter
2. SecurityContextPersistenceFilter
3. ConcurrentSessionFilter
4. LogoutFilter
5. **Mechanizm autentykacji - UsernamePasswordAuthenticationFilter, CasAuthenticationFilter, BasicAuthenticationFilter, Ldap, oAuth, custom..., AuthenticationSuccessHandler and AuthenticationFailureHandler**
6. RememberMeAuthenticationFilter
7. AnonymousAuthenticationFilter
8. ExceptionTranslationFilter, AuthenticationEntryPoint
9. FilterSecurityInterceptor

BOTTEGA IT minds

Spring Security - Mechanizmy autentykacji

• HTTP basic • Username Password	potrzebują dostępu do bazy danych użytkowników: • in memory • dao (UserDetailsService) • ldap
• Central Authentication Server • OpenId • OAuth	logowanie następuje u zewnętrznego dostawcy tożsamości (Single Sign On)
Custom	pełna dowolność skąd pobierzemy użytkownika, musimy go ustawić w SecurityContextHolder

Spring Security - Remember me

- cookie z danymi pozwalającymi rozpoznać użytkownika

```
base64(
        username + ":" + expirationTime + ":" +
        md5Hex(username + ":" + expirationTime + ":" password + ":" + key)
)
```

- RememberMeFilter

Spring Security - Autoryzacja

na poziomie requestów

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  protected void configure(HttpSecurity http) throws Exception {
     http
        .authorizeRequests()
        .antMatchers("/resources/**", "/signup", "/about").permitAll()
        .antMatchers("/admin/**").hasRole("ADMIN")
        .antMatchers("/db/**").access("hasRole('ADMIN') and hasRole('DBA')")
        .anyRequest().authenticated();
  }


}
```

Spring Security - Autoryzacja

na poziomie metod (AOP)

```
@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
```

```
public interface BankService {
  @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
  Account readAccount(Long id);

  @Secured("IS_AUTHENTICATED_ANONYMOUSLY")
  Account[] findAccounts();

  @Secured("ROLE_TELLER")
  Account post(Account account, double amount);
}
```

Spring Security - Autoryzacja

spring security expressions

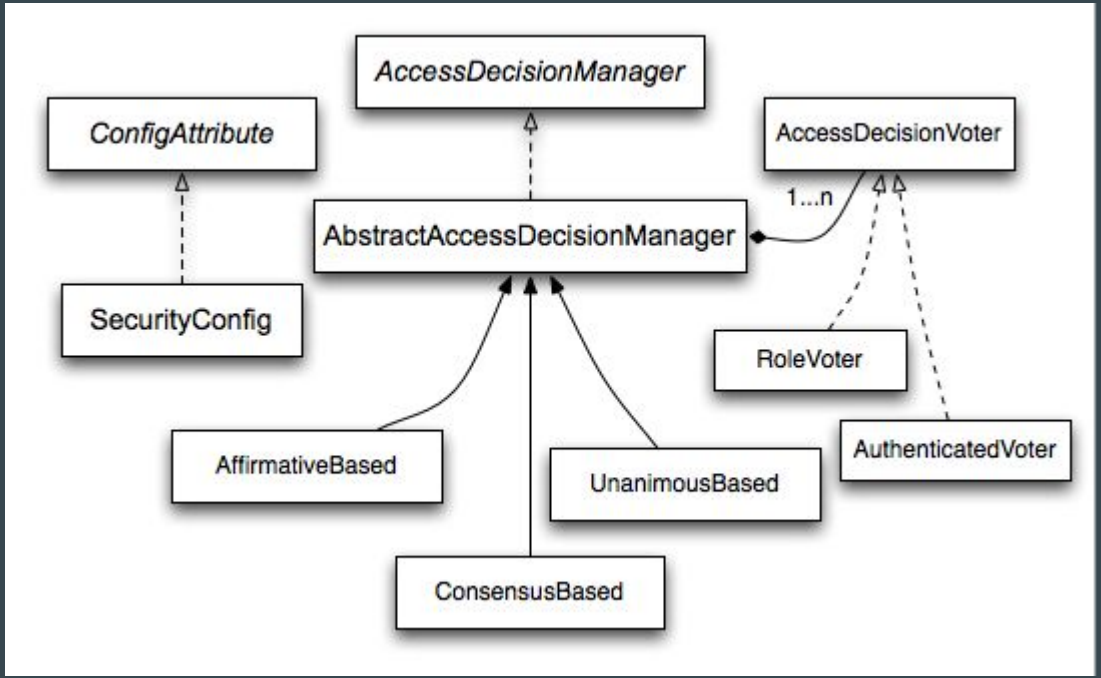
```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
```

```
public interface BankService {
  @PreAuthorize("isAnonymous()")
  Account readAccount(Long id);

  @PreAuthorize("hasAuthority('ROLE_TELLER')")
  Account[] findAccounts();

  @PreAuthorize("hasAuthority('ROLE_TELLER') || isRememberMe() || @myBean.someMethod(account, amount)")
  Account post(Account account, double amount);
}
```

BO·TT·EGA
IT minds

Spring Security - Autoryzacja

pod maską



Spring Security - konfiguracja

- domyślne ustawienia

```
protected void configure(HttpSecurity http) throws Exception {
        http.
                authorizeRequests().
                   anyRequest().authenticated().
        and().
                formLogin().
        and().
                httpBasic();
}
```

Spring Security - konfiguracja

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

   protected void configure(HttpSecurity http) throws Exception {
      http
          .addFilterBefore(myFilter, UsernamePasswordAuthenticationFilter.class)
         .authorizeRequests()
         .antMatchers("/resources/**", "/signup", "/about").permitAll()
         .antMatchers("/admin/**").hasRole("ADMIN")
         .antMatchers("/db/**").access("hasRole('ADMIN') and hasRole('DBA')")
         .anyRequest().authenticated();
   }


}
```

Spring Security - hashowanie haseł

- PasswordEncoder
- 5f4dcc3b5aa765d61d8327deb882cf99
- salt
- BcryptPasswordEncoder

```
class OrderService {

  private OrderRepository orderRepository;
  private OrderMailer mailer;
  private AdminMailer mailer;
  private WarehouseFacade warehouseFacade;
  private EngravingFacade engravingFacade;
  private boolean warehouseModuleActive;
  private boolean engravingEnabled;

  @Transactional
  public void place(Long orderId) {
    Order order = orderRepository.get(orderId);
    order.place();
    mailer.sendConfirmationEmail(order);
    boolean reminderSent = false;
    if(warehouseModuleActive)
      warehouseFacade.scheduleShipping(order);
    else {
      adminMailer.sendOrderRemainder(order);
      reminderSent = true;
    }
    if(order.hasEngravableItems())
      if(engravingEnabled)
        engravingFacade.engrave(order.getItems());
      else
      if(!reminderSent)
        adminMailer.sendOrderRemainder(order);
  }

}
```

Zdarzenia i pluginy

Zmniejszamy coupling, zwiększamy kohezję

```
class OrderProcess {

   private OrderRepository orderRepository;

   @Transactional
   public void place(Long orderId) {
      Order order = orderRepository.get(orderId);
      order.place();
   }

}
```

```
@Entity
public class Order {

   @Transient
   private EventPublisher eventPublisher;

   public void place() {
      //.. some domain logic
      eventPublisher.publish(new OrderPlacedEvent(this));
   }

}
```

Spring - zdarzenia

publikacja zdarzeń

```
package org.springframework.context;

public interface ApplicationEventPublisher {

  void publishEvent(ApplicationEvent event);

  void publishEvent(Object event);

}
```

Spring - zdarzenia

odbiór zdarzeń

```
@Component
public class OrderPlacedCustomerNotifier {

   private OrderMailer orderMailer;
   private OrderRepository orderRepository;

   @EventListener
   public void orderPlaced(OrderPlacedEvent event) {
      Order order = orderRepository.get(event.getAggregateId());
      orderMailer.notifyCustomer(order);
   }

}
```

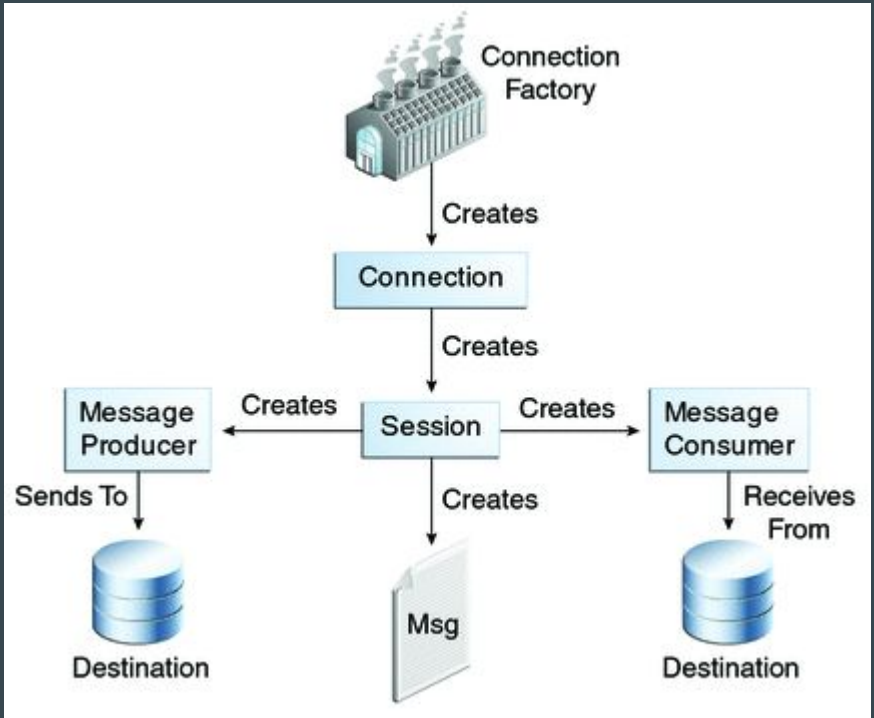
BO·TT·EGA
IT minds

Spring - pluginy

Jak aktywować listenery w zależności od wdrożenia?

- include/not include on classpath
- XML
- @Conditional + config
- Profiles

JMS - programming model



- Messaging domains
  - P2P
  - Pub/Sub
- Administrative objects (destinations, ConnectionFactory)
  - Qeue
  - Topic

Spring - JMS

- uproszczenie korzystania z JMS API
- wysyłanie wiadomości
  - JmsTemplate
- odbiór wiadomości
  - message listeners
  - message driven beans
- konwersja
- wsparcie dla transakcji

Spring JmsTemplate

- wysyłanie
- odbieranie synchroniczne
- przeglądanie kolejek

```
public class JmsTemplate {
        public <T> T execute(SessionCallback<T> action, boolean startConnection) {...}
        public void send(MessageCreator messageCreator) {...}
        public void send(final Destination destination, final MessageCreator messageCreator) {...}
        public void send(final String destinationName, final MessageCreator messageCreator) {...}
        public void convertAndSend(String destinationName, final Object message) {...}
        public void convertAndSend(Destination destination, final Object message) {...}
        public void convertAndSend(Object message, MessagePostProcessor postProcessor) {...}
        public <T> T browse(BrowserCallback<T> action) {...}
        public <T> T browse(Queue queue, BrowserCallback<T> action) {...}
        public <T> T browse(String queueName, BrowserCallback<T> action) {...}
        public Message receive() {...}
        public Message receive(Destination destination) {...}
        ….
}
```

Spring JmsTemplate

```
public class JmsQueueSender {

   private JmsTemplate jmsTemplate;
   private Queue queue;

   public void setConnectionFactory(ConnectionFactory cf) {
      this.jmsTemplate = new JmsTemplate(cf);
   }

   public void setQueue(Queue queue) {
      this.queue = queue;
   }

   public void simpleSend() {
      this.jmsTemplate.send(this.queue, new MessageCreator() {
         public Message createMessage(Session session) throws JMSException {
            return session.createTextMessage("hello queue world");
         }
      });
   }
}
```

Spring Jms MessageConverter

- konwersja Message <-> user Object

```
public interface MessageConverter {

  Message toMessage(Object object, Session session);

  Object fromMessage(Message message);

}
```

- SimpleMessageConverter
- MappingJacksonToMessageConverter
- MarshallingMessageConverter

Spring Message Driven Beans

- asynchroniczne odbieranie wiadomości

```
public class ExampleListener implements MessageListener {

   public void onMessage(Message message) {
      if (message instanceof TextMessage) {
         try {
            System.out.println(((TextMessage) message).getText());
         }
         catch (JMSException ex) {
            throw new RuntimeException(ex);
         }
      }
      else {
         throw new IllegalArgumentException("Message must be of type TextMessage");
      }
   }

}
```

```
<bean id="messageListener" class="jmsexample.ExampleListener" />
<bean id="jmsContainer"
class="org.springframework.jms.listener.DefaultMessageListenerContainer">
            <property name="connectionFactory" ref="connectionFactory"/>
            <property name="destination" ref="destination"/>
            <property name="messageListener" ref="messageListener" />
</bean>
```

Spring Message Driven Beans

- asynchroniczne odbieranie wiadomości - annotation driven

```
@Component
class SomeService {
   @JmsListener(destination = "my-queue")
   public void somethingHappened(String data) {

   }
}
```

- odbiór i wysyłanie odpowiedzi

```
@JmsListener(destination = "myDestination")
@SendTo("status")
public OrderStatus processOrder(Order order) {
   // order processing
   return status;
}
```

Spring JMS konfiguracja

```
@Configuration
@EnableJms
public class SpringConfig {
@Bean
public JmsListenerContainerFactory<?> myFactory(ConnectionFactory connectionFactory,
                                    DefaultJmsListenerContainerFactoryConfigurer configurer) {
  DefaultJmsListenerContainerFactory factory = new DefaultJmsListenerContainerFactory();
  // This provides all boot's default to this factory, including the message converter
  configurer.configure(factory, connectionFactory);
  // You could still override some of Boot's default if necessary.
  return factory;
}

@Bean // Serialize message content to json using TextMessage
public MessageConverter jacksonJmsMessageConverter() {
  MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
  converter.setTargetType(MessageType.TEXT);
  converter.setTypeIdPropertyName("_type");
  return converter;
}
```

Spring JMS Transakcje

- lokalne transakcje
  - przetwarzanie message w transakcji jms
  - operacje bazodanowe w transakcji db
- rozproszone transakcje
  - jta (bitronix, atomikos)
  - transakcja jms i db

Spring Async

- przetwarzanie w tle
  - asynchroniczne (AsyncTaskExecutor)
  - planowanie (TaskScheduler)

```
public interface AsyncTaskExecutor extends TaskExecutor {

        void execute(Runnable task, long startTimeout);

        Future<?> submit(Runnable task);

        <T> Future<T> submit(Callable<T> task);
}
```

```
public interface TaskScheduler {

  ScheduledFuture<?> schedule(Runnable task, Trigger trigger);


  ScheduledFuture<?> schedule(Runnable task, Date startTime);

 ScheduledFuture<?> scheduleAtFixedRate(Runnable task, Date
startTime, long period);

  ScheduledFuture<?> scheduleAtFixedRate(Runnable task, long
period);

  ScheduledFuture<?> scheduleWithFixedDelay(Runnable task, Date
startTime, long delay);

  ScheduledFuture<?> scheduleWithFixedDelay(Runnable task, long
delay);

}
```

- SimpleAsyncTaskExecutor, ThreadPoolTaskExecutor....

Spring Async - konfiguracja

```
@Configuration
@EnableAsync
@EnableScheduling
public class AppConfig {

        @Override
        public Executor getAsyncExecutor() {
          ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
          executor.setCorePoolSize(2);
          executor.setMaxPoolSize(2);
          executor.setQueueCapacity(500);
          executor.setThreadNamePrefix("DMS-Async-Executor");
          executor.initialize();
          return executor;
        }

}
```

BO·TT·EGA
IT minds

Spring Async - użycie

```
@Scheduled(fixedDelay=5000)
public void doSomething() {
   // something that should execute periodically
}

@Scheduled(fixedRate=5000)
public void doSomething() {
   // something that should execute periodically
}

@Scheduled(initialDelay=1000, fixedRate=5000)
public void doSomething() {
   // something that should execute periodically
}

@Scheduled(cron="*/5 * * * * MON-FRI")
public void doSomething() {
   // something that should execute on weekdays only
}
```

```
@Async
void doSomething() {
   // this will be executed asynchronously
}

@Async
void doSomething(String s) {
   // this will be executed asynchronously
}

@Async
Future<String> returnSomething(int i) {
   // this will be executed asynchronously
}
```

Spring Async - error handling

```
public class MyAsyncUncaughtExceptionHandler implements AsyncUncaughtExceptionHandler {

   @Override
   public void handleUncaughtException(Throwable ex, Method method, Object... params) {
      // handle exception
   }
}
```

Spring Async - events a transakcje

```
@Component
public class PrintDocumentScheduler {

   @EventListener
   @Async
   public void documentPublished(DocumentPublishedEvent event) {
      Logger.getLogger(PrintDocumentScheduler.class).info("Scheduling document printing!");
   }

}
```

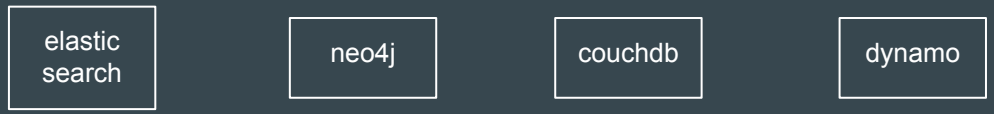
```
@Component
public class PrintDocumentScheduler {

   @TransactionalEventListener
   @Async
   public void documentPublished(DocumentPublishedEvent event) {
      Logger.getLogger(PrintDocumentScheduler.class).info("Scheduling document printing!");
   }

}
```

BO·TT·EGA
IT minds

Spring Data

- abstrakcja dostępu do danych, max redukcja boiler plate
- implementacje dla różnych baz danych

- community



Spring Data - Repository

```
@NoRepositoryBean
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {
  <S extends T> S save(S id);

  <S extends T> Iterable<S> save(Iterable<S> ids);

  T findOne(ID id);

  boolean exists(ID id);

  Iterable<T> findAll();

  Iterable<T> findAll(Iterable<ID>id);

  long count();

  void delete(ID id);

  void delete(T entity);

  void delete(Iterable<? extends T>entities);

  void deleteAll();
}
```

Spring Data - Repository

```
@NoRepositoryBean
public interface PagingAndSortingRepository<T, ID extends Serializable> extends CrudRepository<T, ID> {
  Iterable<T> findAll(Sort sort);

  Page<T> findAll(Pageable pagable);
}
```

Spring Data - własne repozytorium bazowe

```
@NoRepositoryBean
public interface MyBaseRepo<T, ID extends Serializable> extends Repository<T, ID> {

        <S extends T> S save(S var1);

        T findOne(ID var1);

        default T get(ID id) throws NotFoundException {...}

}
```

Spring Data - definiowanie kwerend

- strategie
  - CREATE
    - find...By, read...By, count...By, query...By, get...By
  - USE_DECLARED_QUERY
    - @Query, @NamedQuery lub inne w zależności od impl
  - CREATE_IF_NOT_FOUND
- @EnableJpaRepositories(...)

Spring Data - przykłady kwerend

```
public interface UserRepository extends Repository<User, Long> {

List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

// Enabling ignoring case for an individual property
List<Person> findByLastnameIgnoreCase(String lastname);
// Enabling ignoring case for all suitable properties
List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

// Enabling static ORDER BY for a query
List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
List<Person> findByLastnameOrderByFirstnameDesc(String lastname);

@Query("SELECT p FROM Person p WHERE p.some = :some")
List<Person> findBySomeCustomQuery(@Param("some") String some);

}
```

Spring Data - przykłady kwerend

```
public interface UserRepository extends Repository<User, Long> {

User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);

}
```

Spring Data - custom functionality

```
public interface UserRepositoryCustomFunc {

        void comeCustomMethod();

}

public interface UserRepository extends CrudRepository<User, Long>, UserRepositoryCustomFunc {
        ...
}

public class UserRepositoryCustomFuncImpl implements UserRepositoryCustomFunc  {

        public void comeCustomMethod() { … }

}
```

Spring Data - custom functionality

```
public interface CustomRepository<T, ID extends Serializable> extends Repository<T, ID> {

   void comeCustomMethod1();
   void comeCustomMethod2();

}

public class CustomRepositoryImpl<T, ID extends Serializable>extends SimpleJpaRepository<T, ID> implements CustomRepository<T, ID>  {

   private final EntityManager entityManager;

   public MyRepositoryImpl(JpaEntityInformation entityInformation, EntityManager entityManager) {
      super(entityInformation, entityManager);
      this.entityManager = entityManager;
   }

   @Override
   public void comeCustomMethod1() {...}

   @Override
   public void comeCustomMethod2() {...}

}
```

```
@EnableJpaRepositories(repositoryBaseClass = CustomRepositoryImpl.class)
```

Spring Boot - profile

```
@Configuration
@Profile("production")
public class ProductionConfiguration {

   @Bean
   SmsSender smsSender() {return new RealSmsSender();}

}

@Configuration
@Profile("development")
public class DevelopmentConfiguration {

   @Bean
   SmsSender smsSender() {return new FakeSmsSender();}

}
```

Spring Boot - profile

```
@Component
@Profile("production")
public class ProdOnlyBean {

}

@Configuration
public class SpringConfiguration {

   @Bean
   @Profile("test")
   SmsSender smsSender() {return new FakeSmsSender();}

}
```

BO·TT·EGA
IT minds

Spring Boot - aktywacja profili

- w pliku application.properties:
  - `spring.profiles.active=dev,hsqldb`
- jvm param
  - `-Dspring.profiles.active=dev,hsqldb`
- programowo
  - SpringApplication.setActiveProfiles(String... profiles)

Spring Boot - koniguracja per profil

- application-production.properties
- application-test.properties
- application-qa.properties