Dokumentacja projektowa
Portal aukcyjny „Alledrogo”

Podstawy Teleinformatyki

Autor: Łukasz Górny

1. Założenia projektu

Projekt w założeniu jest prostym portalem aukcyjnym, wzorowanym na popularnym Allegro w kwestii ułożenia elementów oraz kolorystyki. Posiada podstawowe funkcjonalności dla zwykłego użytkownika, tj:
- rejestracja
- logowanie
- wystawianie przedmiotów w ramach różnych kategorii
- wyszukiwarkę
- składanie ofert
- wyświetlanie czasu pozostałego do końca aukcji
- wyświetlanie historii złożonych ofert w ramach pojedynczej aukcji
- wyświetlanie losowych 4 aukcji na stronie głównej

Dodatkowo w systemie istnieje prosty panel administracyjny, który umożliwia użytkownikowi z uprawnieniami administratora na dodawanie nowych kategorii.

2. Opis techniczny projektu

Projekt został napisany w języku **Java** z wykorzystaniem wersji **8** (Java Development Kit wersja 1.8_111). Ponadto użyte technologie to:

a) **Spring MVC w wersji 4.3.4** – główny framework odpowiedzialny m.in. za obsługę żądań wysyłanych przez przeglądarkę, przygotowywanie odpowiedzi, dependency injection

b) **Spring Security w wersji 4.2.0 RELEASE** – moduł frameworku Spring odpowiedzialny za bezpieczeństwo (obsługę ról i uprawnień użytkowników)

c) **LogBack w wersji 1.1.3** – biblioteka służąca do wyświetlania oraz zapisywania logów

d) **Hibernate w wersji 5.2.5** – ORM wykorzystywany w warstwie bazodanowej aplikacji, służący do pobierania, zapisywania, aktualizacji oraz usuwania danych z bazy danych

e) **GSON w wersji 2.8.0** – biblioteka ułatwiająca obsługę JSON w aplikacji Javowej

f) **Thymeleaf w wersji 3.0.2 RELEASE –** silnik frontendowy w znaczny sposób ułatwiający budowanie skomplikowanego interfejsu użytkownika

g) **Quartz w wersji 2.2.3** – biblioteka do tworzenia zadań i wykonywania zadań cyklicznych w tle

h) **MySQL Connector Java w wersji 6.0.5** – biblioteka umożliwiająca nawiązanie połączenia z serwerem MySQL z poziomu aplikacji Java

i) **Serwer MySQL** – serwer bazodanowy

j) **Maven w wersji 3.2.0** – aplikacja do zarządzania zależnościami oraz wersjami bibliotek użytych w projekcie, służy również do budowania pliku wynikowego w formacie .war

k) **Tomcat w wersji 9.0.0 M3** – kontener aplikacyjny służący do uruchomienia aplikacji

W projekcie zostały użyte również następujące biblioteki Javascriptowe:

a) **JQuery w wersji 2.1.1**

b) **AngularJS w wersji 1.5.8**

c) **JQuery Growl w wersji 1.3.2** – biblioteka umożliwiająca wyświetlanie wiadomości dla użytkownika

d) **Countdown.js w wersji 2.2.0**

e) **JQuery Mask w wersji 1.14.3**

Ponadto do stworzenia responsywnego interfejsu użytkownika została użyta biblioteka **MaterializeCSS w wersji 0.97.8**.

3. Model aplikacji

Każda klasa opisująca model w aplikacji dziedziczy po klasie abstrakcyjnej **AbstractModel**, przedstawionej poniżej:

```
package pl.gorny.model;

import javax.persistence.*;
import java.time.LocalDateTime;

@Inheritance(strategy = InheritanceType.JOINED)
@MappedSuperclass
public abstract class AbstractModel {

@Column(name = "create_date")
protected LocalDateTime createDate;

@Column(name = "update_date")
protected LocalDateTime updateDate;

@Column(name = "deleted")
protected boolean deleted;

@PrePersist
protected void onCreate() {
updateDate = createDate = LocalDateTime.now();
}
```

```
@PreUpdate
protected void onUpdate() {
updateDate = LocalDateTime.now();
}


}
```

Adnotacje **@PrePersist** oraz **@PreUpdate**, występujące opowiednio na metodach **onCreate** in **onUpdate** mają za zadanie zapewnić nam wykonanie ciała metody przed zapisaniem bądź aktualizacją krotki w bazie danych. Zostało to przeze mnie wykorzystane do zapewnienia, iż w bazie danych zawsze zostanie zapisany dokładny czas utworzenia bądź aktualizacji krotki.

W przedstawionych poniżej klasach zostały pominięte gettery i settery dla czytelności dokumentacji.

Model aukcji:

```
package pl.gorny.model;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.List;

@Entity
@Table(name = "auctions")
public class Auction extends AbstractModel {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column(length = 150)
private String title;

@Column(length = 5000)
private String description;

@Column(name = "items_amount")
private Integer itemsAmount;

@Column(name = "end_date")
private LocalDateTime endDate;

@Column(name = "has_ended")
private Boolean hasEnded;

@Column
private Double price;
```

```
@OneToOne
private Category category;

@ManyToOne(fetch = FetchType.EAGER)
private User seller;

@OneToMany(mappedBy = "auction")
private List<Bid> bids;

@Transient
private Bid winningBid;
```

Model oferty:

```
package pl.gorny.model;

import javax.persistence.*;

@Entity
@Table(name = "bids")
public class Bid {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long id;

@OneToOne
private User buyer;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "auction_id")
private Auction auction;

@Column(name = "is_winning")
private Boolean isWinning;

@Column
private Double amount;
```

Model kategorii:

```
package pl.gorny.model;

import javax.persistence.*;
import java.util.List;

@Entity
@Table(name = "categories")
```

```
public class Category extends AbstractModel {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long id;

@Column(name = "name", nullable = false)
private String name;

@Column(name= "is_top")
private Boolean isTop;

@OneToOne(optional = true)
private Category parent;

@Transient
private List<Category> children;
```

Model użytkownika:

```
package pl.gorny.model;

import javax.persistence.*;
import java.util.List;

@Entity
@Table(name = "users")
public class User extends AbstractModel {

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@Column
private String password;

@Column
private String email;

@Column
private String name;

@Column
private String surname;

@Column
private String nickname;

@Column
private String city;

@Column
```

```
private String zipcode;

@Column
private String address;

@OneToMany(mappedBy = "seller")
private List<Auction> createdAuctions;

@Column(name = "role")
@Enumerated(EnumType.STRING)
private Role role;
```

4. Logika biznesowa

W aplikacji występują następujące rodzaje klas:

a) **Akcja** – oznaczona poprzez przyrostek „Action” w kodzie aplikacji, jest to klasa obsługująca akcję wywołaną przez użytkownika, np. dodanie aukcji, dodanie oferty do aukcji. Akcja korzysta z metod udostępnionych przez Serwisy. Każda akcja w systemie dziedziczy po abstrakcyjnej klasie **AbstractAction**, której kod znajduje się poniżej:

```
package pl.gorny.action;

import org.slf4j.Logger;
import pl.gorny.dto.ResponseDto;

public abstract class AbstractAction<T> {

protected ResponseDto responseDto;

protected String json;

protected Logger logger;

protected T dto;

public abstract void execute();

public void setJson(String json) {
this.json = json;
}

public ResponseDto getResponseDto() {
return this.responseDto;
}
}
```

Przykładowa akcja:

```
package pl.gorny.action;

import com.google.gson.Gson;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import pl.gorny.dto.BidDto;
import pl.gorny.dto.ResponseDto;
import pl.gorny.model.Auction;
import pl.gorny.model.Bid;
import pl.gorny.model.User;
import pl.gorny.service.AuctionService;
import pl.gorny.service.SecurityService;
import pl.gorny.utils.NumberUtils;

@Component
public class BidAction extends AbstractAction<BidDto> {

private Bid bidToPersist;

@Autowired
private AuctionService auctionService;

@Autowired
private SecurityService securityService;

public BidAction() {
logger = LoggerFactory.getLogger(BidAction.class);
}

@Override
public void execute() {
try {
responseDto = new ResponseDto();
parseJsonToObject();
if(validate()) {
prepareBidObjectFromDto();
setLastBidWinningStateToFalseByAuctionId();
saveBid();
parseWinningBidToJson();
responseDto.success = true;
}
} catch (Exception e) {
responseDto.success = false;
logger.error(e.getMessage());
}
```

```
}
private boolean validate() {
if(!checkIsBidAmountValid()) {
return false;
}
if(checkIfBuyerEqualsSeller()) {
return false;
}
if(!checkIsBidLowerThanOrEqualToWinningBidOrOriginalPrice()) {
return false;
}
if(checkIsAuctionEnded()) {
return false;
}
return true;
}
private boolean checkIsBidAmountValid() {
if(NumberUtils.isDouble(dto.getAmount())) {
return true;
}
responseDto.message = "Podana kwota nie jest w właściwym formacie.";
responseDto.success = false;
return false;
}
private boolean checkIfBuyerEqualsSeller() {
Auction auction = getAuctionById();
User user = getUserByEmail();
if(auction.getSeller().getId().equals(user.getId())) {
responseDto.message = "Nie możesz licytować aukcji której jesteś wystawiającym.";
responseDto.success = false;
return true;
}
return false;
}
private boolean checkIsBidLowerThanOrEqualToWinningBidOrOriginalPrice() {
Bid winningBid = auctionService.getWinningBidByAuctionId(dto.getAuctionId());
if(winningBid != null) {
if(winningBid.getAmount() >= Double.parseDouble(dto.getAmount())) {
responseDto.message = "Wartość oferty nie może być mniejsza bądź równa wygrywającej
ofercie.";
```

```
responseDto.success = false;
return false;
}
} else {
Auction auction = auctionService.getOne(dto.getAuctionId());
if(auction.getPrice() >= Double.parseDouble(dto.getAmount())) {
responseDto.message = "Wartość oferty nie może być mniejsza bądź równa cenie początkowej.";
responseDto.success = false;
return false;
}
}
return true;
}
private boolean checkIsAuctionEnded() {
Auction auction = getAuctionById();
if(auction.getHasEnded()) {
responseDto.message = "Ta aukcja już się zakończyła.";
responseDto.success = false;
return true;
}
return false;
}
private void parseJsonToObject() {
Gson gson = new Gson();
dto = gson.fromJson(json, BidDto.class);
}
private void prepareBidObjectFromDto() {
bidToPersist = new Bid();
bidToPersist.setAmount(Double.parseDouble(dto.getAmount()));
bidToPersist.setAuction(getAuctionById());
bidToPersist.setBuyer(getUserByEmail());
bidToPersist.setWinning(true);
}
private void setLastBidWinningStateToFalseByAuctionId() {
auctionService.setLastBidWinningStateToFalseByAuctionId(dto.getAuctionId());
}
private Auction getAuctionById() {
return auctionService.getOne(dto.getAuctionId());
}
private User getUserByEmail() {
```

```
return securityService.getUserByEmail(dto.getBuyerEmail());
}
private void saveBid() {
auctionService.saveBid(bidToPersist);
}
private void parseWinningBidToJson() {
BidDto winningBidDto = new BidDto();
Bid winningBid = auctionService.getWinningBidByAuctionId(dto.getAuctionId());
if (winningBid != null) {
Gson gson = new Gson();
winningBidDto.setAmount(winningBid.getAmount().toString());
responseDto.body = gson.toJson(winningBidDto);
}
}
}
```

Kod wyżej przestawionej akcji odpowiada za walidację wprowadzonych danych przy składaniu oferty oraz przekazanie dalszej odpowiedzialności do odpowiednich klas serwisowych. Każda akcja zaczyna się od wywołania metody **execute**, w której znajduje się blok try-catch. Jeżeli jakakolwiek operacja poskutkuje wystąpieniem wyjątku, zostanie on złapany, a do użytkownika zostanie przesłana informacja o niepowodzeniu operacji.

b) **Serwis** – klasa oznaczona poprzez przyrostek „Service”. W klasach serwisowych dokonywane są mniej lub bardziej skomplikowane operacje na danych pobieranych z bazy danych. Klasa serwisowa korzysta z metod dostarczonych przez inne klasy serwisowe oraz klasy DAO. Klasy serwisowe zostały zaprogramowane na interfejsach. Każda klasa serwisowa, implementująca interfejs, oznaczona jest przyrostkiem „Impl”.

Przykładowy interfejs:

```
package pl.gorny.service;
import pl.gorny.model.Auction;
import pl.gorny.model.Bid;
import java.util.List;
public interface AuctionService {
void save(Auction auction);
List<Auction> getAll();
List<Auction> getTopFourAuctions();
Auction getOne(Long id);
```

```
void saveBid(Bid bid);

void setLastBidWinningStateToFalseByAuctionId(Long id);

Bid getWinningBidByAuctionId(Long id);

List<Bid> getBidsForAuction(Long id);

List<Auction> getNotEndedAuctionsByCriteria(String item, Long id);
}
```

Implementacja w/w interfejsu:

```
package pl.gorny.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import pl.gorny.dao.AuctionDao;
import pl.gorny.model.Auction;
import pl.gorny.model.Bid;

import java.util.List;

@Service
public class AuctionServiceImpl implements AuctionService {

@Autowired
private AuctionDao auctionDao;

@Override
public void save(Auction auction) {
auctionDao.save(auction);
}

@Override
public List<Auction> getAll() {
return auctionDao.findAll();
}

@Override
public List<Auction> getTopFourAuctions() {
List<Auction> auctions = auctionDao.getTopFour();
if(auctions != null && !auctions.isEmpty()) {
for(Auction auction : auctions) {
auction.setWinningBid(getWinningBidByAuctionId(auction.getId()));
}
}

return auctions;
}

@Override
```

```
public Auction getOne(Long id) {
return auctionDao.getOne(id);
}

@Override
public void saveBid(Bid bid) {
auctionDao.saveBid(bid);
}

@Override
public void setLastBidWinningStateToFalseByAuctionId(Long id) {
auctionDao.setLastBidWinningStateToFalseByAuctionId(id);
}

@Override
public Bid getWinningBidByAuctionId(Long id) {
return auctionDao.findWinningBidByAuctionId(id);
}

@Override
public List<Bid> getBidsForAuction(Long id) {
return auctionDao.findBidsForAuction(id);
}

@Override
public List<Auction> getNotEndedAuctionsByCriteria(String item, Long id) {
List<Auction> auctions = auctionDao.findNotEndedAuctionsByCriteria(item, id);

for(Auction auction : auctions) {
auction.setWinningBid(getWinningBidByAuctionId(auction.getId()));
}

return auctions;
}

}
```

c) **Data Access Object (DAO)** – klasa odpowiedzialna za komunikację z źródłem danych, w tym przypadku bazą danych MySQL. Oznaczona poprzez przyrostek „Dao”. Każda klasa dao dziedziczy po klasie abstrakcyjnej AbstractDao. Tak jak klasy serwisowe, klasy dao są zaprogramowane na interfejsach. Każda klasa Dao ma odpowiadający jej interfejs.

Klasa AbstractDao:

```
package pl.gorny.dao;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;

public abstract class AbstractDao {

@Autowired
private SessionFactory sessionFactory;

protected Session getSession() {
return sessionFactory.getCurrentSession();
}

public void persist(Object entity) {
getSession().persist(entity);
}

public void delete(Object entity) {
getSession().delete(entity);
}
}
```

Przykładowy interfejs:

```
package pl.gorny.dao;

import pl.gorny.model.Auction;
import pl.gorny.model.Bid;

import java.util.List;

public interface AuctionDao {
void save(Auction auction);

List<Auction> findAll();

List<Auction> getTopFour();

Auction getOne(Long id);

void saveBid(Bid bid);

void setLastBidWinningStateToFalseByAuctionId(Long id);

Bid findWinningBidByAuctionId(Long id);

List<Bid> findBidsForAuction(Long id);

void endAuction(Long id);

List<Auction> findNotEndedAuctionsByCriteria(String item, Long categoryId);
```

}

Przykładowa implementacja w/w interfejsu:

```
package pl.gorny.dao;

import org.hibernate.query.Query;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
import pl.gorny.model.Auction;
import pl.gorny.model.Bid;
import pl.gorny.utils.Queries;

import java.util.List;

@Transactional
@Repository("AuctionDao")
public class AuctionDaoImpl extends AbstractDao implements AuctionDao {

@Override
public void save(Auction auction) {
persist(auction);
}

@Override
public List<Auction> findAll() {
Query query =
getSession().createSQLQuery(Queries.GET_ALL_NOT_ENDED_AUCTIONS).addEntity(Auctio
n.class);
return query.list();
}

@Override
public List<Auction> getTopFour() {
Query query =
getSession().createSQLQuery(Queries.GET_ALL_NOT_ENDED_AUCTIONS).addEntity(Auctio
n.class);
query.setMaxResults(4);
return query.list();
}

@Override
@Transactional
public Auction getOne(Long id) {
Query query =
getSession().createSQLQuery(Queries.GET_SINGLE_AUCTION).addEntity(Auction.class);
query.setParameter("id", id);
```

```
return (Auction) query.uniqueResult();
}

@Override
public void saveBid(Bid bid) {
persist(bid);
}

@Override
public void setLastBidWinningStateToFalseByAuctionId(Long id) {
Query query =
getSession().createSQLQuery(Queries.UPDATE_LAST_WINNING_STATE_BY_AUCTION_ID)
;
query.setParameter("id", id);

query.executeUpdate();
}

@Override
public Bid findWinningBidByAuctionId(Long id) {
Query query =
getSession().createSQLQuery(Queries.GET_WINNING_BID_BY_AUCTION_ID).addEntity(Bid.
class);
query.setParameter("id", id);

return (Bid) query.uniqueResult();
}

@Override
public List<Bid> findBidsForAuction(Long id) {
Query query =
getSession().createSQLQuery(Queries.GET_BIDS_BY_AUCTION_ID).addEntity(Bid.class);
query.setParameter("id", id);

return query.list();
}

@Override
public void endAuction(Long id) {
Query query = getSession().createSQLQuery(Queries.END_AUCTION);
query.setParameter("id", id);

query.executeUpdate();
}

@Override
public List<Auction> findNotEndedAuctionsByCriteria(String item, Long categoryId) {
Query query =
getSession().createSQLQuery(Queries.GET_NOT_ENDED_AUCTIONS_BY_CRITERIA).addEn
tity(Auction.class);
query.setParameter("id", categoryId);
```

```
query.setParameter("item", "%" + item + "%");
return query.list();
}
}
```

d) **Kontroler** – kontroler jest to klasa, która zajmuje się delegowaniem odpowiedzialności za operacje wywoływane przez wysyłanie żądań na odpowiednie adresy URL do odpowiednich akcji. W projekcie występują dwa rodzaje kontrolerów: zwykłe oraz RESTowe. Pierwszy rodzaj kontrolera używany jest do zwracania odpowiednich formatek, które są wyświetlane użytkownikowi oraz ewentualnie uzupełniania ich odpowiednimi danymi. Drugi rodzaj kontrolera jest używany w przypadku operacji asynchronicznych, czyt. zapis, aktualizacja danych. Kontroler RESTowy przyjmuje dane w postaci JSON i takie też dane zwraca.

Przykładowy kod zwykłego kontrolera:

```
package pl.gorny.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
@RequestMapping("/admin")
public class AdminController {
@GetMapping
public String displayAdminPanel() {
return "fragments/admin/panel";
}
@GetMapping("/add-category")
public String displayAddCategory() {
return "fragments/admin/add-category";
}
}
```

Przykładowy kod kontrolera RESTowego:

```
package pl.gorny.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import pl.gorny.action.AddCategoryAction;
import pl.gorny.dto.ResponseDto;

@RestController
@RequestMapping("/rest/admin")
public class AdminRestController {

@Autowired
private AddCategoryAction addCategoryAction;

@ResponseBody
@PostMapping("/add-category")
public ResponseDto saveCategory(@RequestBody String body) {
addCategoryAction.setJson(body);
addCategoryAction.execute();

return addCategoryAction.getResponseDto();
}
}
```

e) **Data Transfer Object (DTO)** – klasa używana jako przekaźnik danych w projekcie. Klasy te charakteryzują się przyrostkiem „Dto”. Używane są głównie w akcjach przy konwersji danych z formatu JSON na obiekt Javowy (Plain Old Java Object – POJO).

Przykładowy kod klasy DTO, bez setterów i getterów:

```
package pl.gorny.dto;

public class AuctionDto {

private String title;
private String description;
private Integer amount;
private Long category;
private Integer durationInDays;
private String userEmail;
private Double price;
```