

Vervolg opdracht Auctions

We maken een toepassing voor veilingen.

Gebruikers kunnen via het systeem een bod uitbrengen op items.

Je kan voor deze opdracht kiezen of je JEE of Spring Boot gebruikt.

JEE implementatie

Je kan de bestaande opdracht AuctionsExercise verder uitbreiden.

Spring implementatie

Maak een nieuw Spring Boot project. Kopieer alle functionaliteit vanuit de AuctionsExercise-oplossing naar je nieuwe Spring Boot applicatie. Je kan indien je dat wenst gebruikmaken van Spring Data. Pas ook de implementatie voor de Rest endpoints aan. Test je Rest endpoints om gebruikers toe te voegen en op te vragen uit. Daarna ben je klaar om verder te werken.

1. Entity klassen

Er zullen items aangeboden in veilingen (auctions). Iedere veiling heeft een beschrijving (description) en een einddatum (endDate). Gebruikers kunnen een bod uitbrengen op een item.

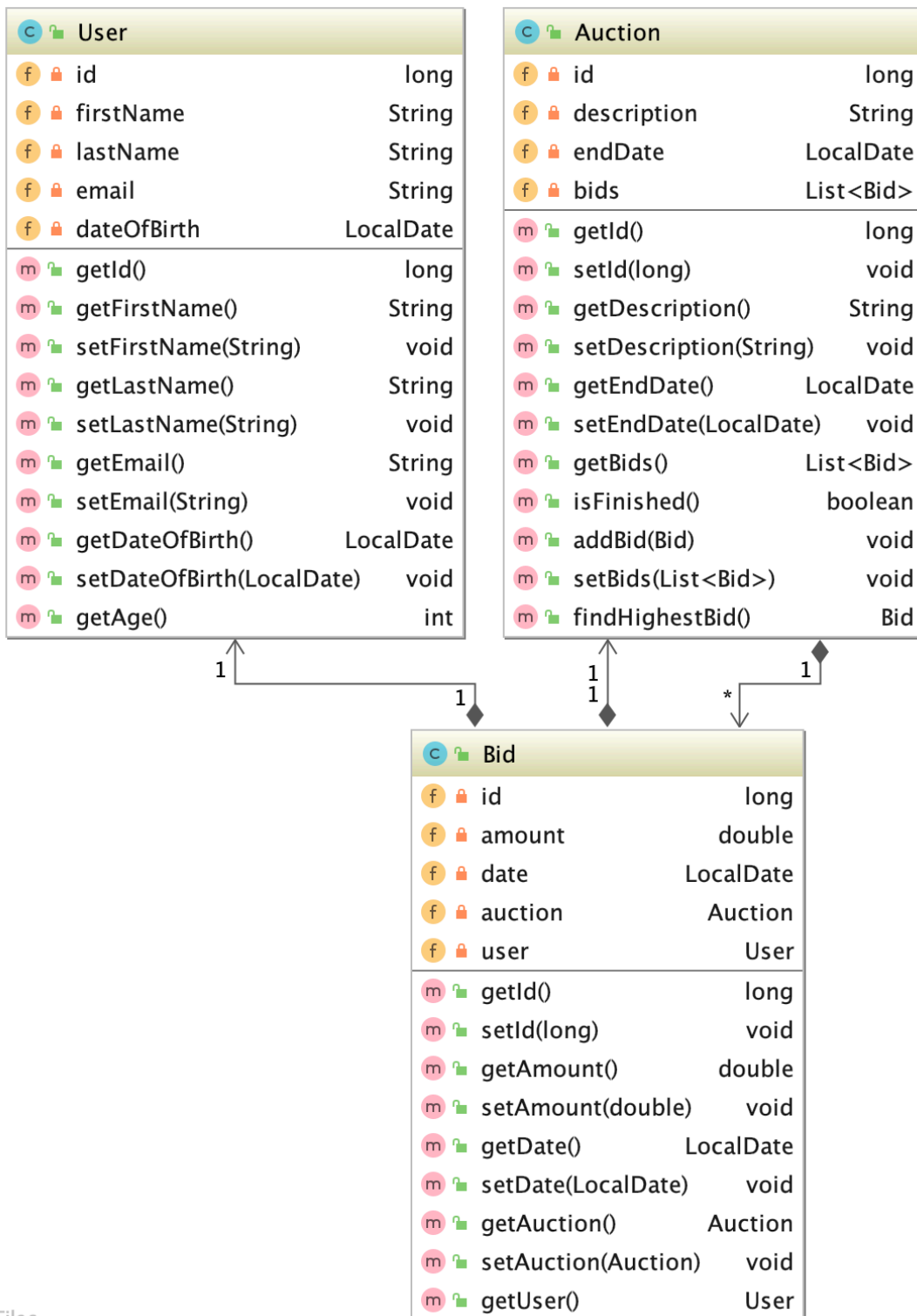
De entity klasse User heb je reeds ontwikkeld in één van de voorgaande oefeningen. Pas nu de klassen Auction en Bid aan zodat dit ook geldige entity klassen worden. Voorzie de bi-directionele relatie tussen Auction en Bid (cascadetype in de klasse Auction wordt ALL). De methode addBid() zorgt ervoor dat deze bi-directionele relatie steeds wordt gerespecteerd. De methode addBid() bevat geen business-logica. De business-logica implementeren we in de business-laag. Zorg er ook voor dat de andere relaties tussen de klassen correct worden geannoteerd.

Implementeer de methoden isFinished() en findHighestBid().

isFinished() geeft false indien de einddatum van de veiling nog niet is verstreken, en true indien de einddatum van de veiling is verstreken.

findHighestBid() geeft het Bid-object met de hoogste waarde. Indien er nog geen biedingen zijn wordt null gegeven.

Schrijf unit testen voor de methoden `isFinished()` en `findHighestBid()`.



2. DAO

De UserDao moet de volgende functionaliteit aanbieden (normaalgezien is deze functionaliteit reeds geïmplementeerd):

```
public interface UserDao {  
    User saveUser(User user);  
    User findUserByEmail(String email);  
    User findUserById(long userId);  
    List<User> findAllUsers();  
}
```

1. Het opslaan van een nieuwe user
2. Het opzoeken van een user adhv een email
3. Het opzoeken van een user adhv de primary key
4. Het opzoeken van alle users in het systeem

De AuctionDao voorziet de volgende functionaliteit:

```
public interface AuctionDao {  
    Auction saveAuction(Auction auction);  
    Auction findAuctionById(long auction);  
    List<Auction> findAllAuctions();  
}
```

1. Het opslaan van een nieuwe veiling (auction)
2. Het opzoeken van een auction adhv de primary key
3. Het opzoeken van alle auctions in het systeem

PS Indien je Spring Data gebruikt mogen de namen van de methoden in je persistence-laag afwijken.

Voeg min. 2 unit testen toe voor de AuctionDao-klasse. Maak gebruik van een in-memory databank.

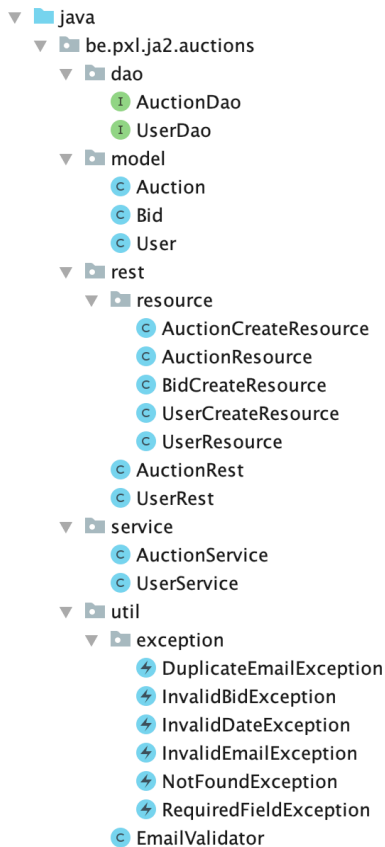
4. Business-laag

De UserService heb je normaalgezien reeds voorzien. Pas deze wel nog aan zodat je business-laag geen entity-objecten doorgeeft aan de aanroepende laag. Maak steeds gebruikmaakt van DTO's.

AuctionService		
f	auctionDao	AuctionDao
f	userDao	UserDao
m	doBid(Long, BidCreateResource)	void
m	findAuctions()	List<AuctionResource>
m	mapAuction(Auction)	AuctionResource
m	createAuction(AuctionCreateResource)	AuctionResource

UserService		
f	userDao	UserDao
m	getAllUsers()	List<UserResource>
m	getUserById(long)	UserResource
m	createUser(UserCreateResource)	UserResource
m	mapToUser(UserCreateResource)	User
m	mapToUserResource(User)	UserResource

Alle DTO-klassen en exception-klassen krijg je cadeau (packages *be.pxl.ja2.auctions.rest.resource* en *be.pxl.ja2.auctions.util.exception*). Hieronder zie je een overzicht van de package-structuur die je uiteindelijk in je oplossing zal bekommen.



Implementeer nu de `AuctionService`. De methoden `createAuction()` en `findAllAuctions()` worden respectievelijk gebruikt om een nieuwe veiling aan te maken en een overzicht op te vragen van alle veilingen in het systeem.

In de methode `doBid()` gaan we volgende business-regels implementeren:

- Gooi een exception op indien de auction en de user niet bestaan.
- Gooi een exception op indien het nieuwe bod lager is dan het huidige hoogste bod.
- Gooi een exception op indien de user reeds het hoogste bod heeft.
- Gooi een exception op indien de auction reeds afgelopen is.

Schrijf unit testen om de methode `doBid()` grondig te testen.

5. Rest endpoints


Voorzie de onderstaande Rest endpoints. Je kan deze testen door ze aan te roepen met postman, insomnia of een andere tool.

a. Auction aanmaken

POST	http://localhost:8080/auctions/rest/auctions	Send	202	207 ms	0 B			
JSON	Auth	Query	Header 1	Docs	Preview	Header 2	Cookie	Timeline
<pre>1 { 2 "endDate": "03/06/2020", 3 "description": "something nice" 4 }</pre>					No body returned for response			

Geef HTTP status 202 (accepted) of 201 (created) terug als resultaat.

b. Overzicht van alle veilingen

GET	http://localhost:8080/auctions/rest/auctions	Send	200	5.55 ms	427 B			
Body	Auth	Query	Header	Docs	Preview	Header 3	Cookie	Timeline
 Select a body type from above					<pre>1 [2 { 3 "id": 1, 4 "description": "something nice", 5 "endDate": "2020-05-22", 6 "finished": false, 7 "numberOfBids": 1, 8 "highestBid": 21.5, 9 "highestBidBy": "nele.custers@pxl.be" 10 }, 11 { 12 "id": 2, 13 "description": "Waardevol stripboek", 14 "endDate": "2020-05-19", 15 "finished": false, 16 "numberOfBids": 0, 17 "highestBid": 0.0, 18 "highestBidBy": null 19 }, 20 { 21 "id": 3, 22 "description": "Antieke theepot", 23 "endDate": "2020-05-25", 24 "finished": false, 25 "numberOfBids": 0,</pre>			

Geef een lijst terug met alle veilingen (ook die reeds afgelopen zijn). Voorzie van iedere veiling de gevraagde informatie zoals numberOfBids, highestBid en highestBidBy.

c. Bod plaatsen

Insomnia - Do bid								
POST	http://localhost:8080/auctions/rest/auctions/1/bids	Send	202	16.4 ms	0 B	Just Now		
JSON	Auth	Query	Header 1	Docs	Preview	Header 2	Cookie	Timeline
<pre>1 { 2 "email": "nele.custers@pxl.be", 3 "price": 21.5 4 } 5</pre>					No body returned for response			

Indien het plaatsen van het bod is gelukt geef je HTTP status 202 (accepted) of 201 (created) terug als resultaat.

Indien er zich een foutmelding voordoet geef je een gepast foutmelding terug (foutmelding in json formaat zoals in de voorbeelden hieronder is niet noodzakelijk).

POST http://localhost:8080/auctions/rest/auctions/2/bids Send 400 5.3 ms 170 B Just Now

JSON Auth Query Header 1 Docs

```
1 {
2   "email": "nele.custers@pxl.be",
3   "price": 22.5
4 }
5
```

Preview Header 4 Cookie Timeline

```
1 {
2   "timestamp": "2020-05-19T13:22:22.917+0000",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "Auction {Antieke theepot} is finished.",
6   "path": "/auctions/rest/auctions/2/bids"
7 }
```

Je kan geen bod plaatsen als de veiling is afgelopen.

POST http://localhost:8080/auctions/rest/auctions/1/bids Send 400 7.62 ms 168 B Just Now

JSON Auth Query Header 1 Docs

```
1 {
2   "email": "nele.custers@pxl.be",
3   "price": 22.5
4 }
5
```

Preview Header 4 Cookie Timeline

```
1 {
2   "timestamp": "2020-05-19T13:21:54.393+0000",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "You're already winning this auction.",
6   "path": "/auctions/rest/auctions/1/bids"
7 }
```

Je kan geen bod plaatsen als je zelf reeds het hoogste bod hebt geplaatst.

POST http://localhost:8080/auctions/rest/auctions/1/bids Send 400 7.56 ms 179 B Just Now

JSON Auth Query Header 1 Docs

```
1 {
2   "email": "sofie@pxl.be",
3   "price": 14.5
4 }
5
```

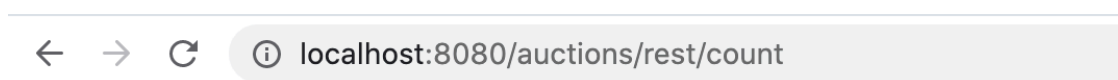
Preview Header 4 Cookie Timeline

```
1 {
2   "timestamp": "2020-05-19T13:21:35.506+0000",
3   "status": 400,
4   "error": "Bad Request",
5   "message": "Bid not allowed. Your bid should exceed €21.5",
6   "path": "/auctions/rest/auctions/1/bids"
7 }
```

Je kan geen bod plaatsen dat lager is dan het hoogste bod.

6. Servlet

Implementeer een servlet (GET) die in een html-pagina weergeeft hoeveel veilingen momenteel actief zijn.



Er zijn momenteel 8 veilingen actief.