An Auction Web App

Group: 3-5 students
Mark: 15%
Deadline: Friday 29 November

Description: Your task is to develop an **auction website** (think of Ebay) using the Django framework. The website should provide the following functionalities:

1. Users can create an account on the site and and able to login into their account.
2. The user's profile should contain at least an email and their date of birth.
3. Users should be able to post a new item for auction. Items should contain a title, a description, a picture and the date/time the auction finishes.
4. Users of the site should then be able to bid for an item, before the end date/time of that item.
5. At the end of the auction, the highest bidder is able to "buy" the item. You do not need to implement the "payment" process, but your app should alert the winner, e.g. via a message page or as part of their profile.
6. The site must contain a page listing all the items that are currently available, with the ability for "search" for items based on a given keyword. For instance, searching for "table" should return the list of items that have "table" as part of the title or the description. The searching mechanism should be done using ajax (so no page refreshes).
7. You should also have a page containing "closed" auctions, detailing the list of biddings, and the winner.
8. The frontend should use Bootstrap, and be responsive.
9. <u>For top marks</u>, apart from the basic features above, your group should implement at least one extra feature. Feel free to include any extra feature you can think of. Here are three examples of possible extra features:
    a) At the end of the auction, the winner receives an email confirming that they should proceed to purchase the item. You might need to use a cron job, so you regularly check for "closed" auctions.
    b) Users are able to send questions to the item owner about the condition of the item, and the owner is able to reply to those questions.
    c) Users are able to reset their password, in case they have forgotten it. This would be done via a "reset password link" sent to their email.

Emails: If you app is sending emails, please create temporary Gmail account, and use that to send emails, as you will need to submit the code for your app which might contain the password for the email account (unless you are using environment variables on the server to store your passwords).

Outcome: Once fully tested, your application should be deployed to the school's OpenShift web servers (to be discussed in week 7) — one deployed app per team. Each group should submit the **<u>code</u>** together with a **<u>one-page report</u>** describing their extra feature (no more than a few paragraphs). Submit these to QM+ as a single zip file. Remember to include in the report the URL of your deployed application, and the username and password for the admin page.

MARKING SCHEME ON NEXT PAGE

Marking scheme

Maximal mark (100 marks)	**Different degrees of achievement**	
Deployment (10 marks)	App successfully deployed	**10**
	Deployment attempted but not fully working	5
	Deployment attempted but not working at all	2
	Deployment not attempted	0
Functionality (35 marks)	All features working as expected	**35**
	Most of the features working	20 - 30
	Most of the features NOT working	5 - 20
	None of the features working	0
Data modelling (10 marks)	Excellent use of Django models, e.g. using Django's User model, class methods and relationship fields	**10**
	Incorrectly modelled data or no use of Django's model features	4 - 9
Frontend (10 marks)	Excellent use of Bootstrap, jQuery and ajax requests	**10**
	No use of Bootstrap, jQuery or ajax	3 - 9
Views + Templates (20 marks)	Code duplication avoided using decorators and template hierarchy. Good use of URL reversing.	**20**
	Some duplication of code and/or hardcode URLs	5 - 19
Extra feature (15 marks)	Extra feature is complex, is fully working and the report explains it clearly and concisely.	**15**
	Extra feature is not too complex or is not fully working, or is not explained in the report.	0 - 14
Total		**100**