WebShop web application - Project requirements

The webshop is a web application where users can buy and sell items. An Item can be on-sale or sold. An item becomes “sold” when it is purchased by another user. For the buyer, the item will show as “purchased”.

There are 2 kinds of users: **registered** and **unregistered**. Unregistered users are anonymous and can only browse or search available items. Registered users are authenticated and can additionally buy, sell and view their own inventory. This is a list of requirements for the webshop.

Pts	Functional Requirements
4p	1. **Site architecture (MANDATORY):** The application should have 2 web pages: a. **Landing page:** accessible via the URL ‘/’ (e.g, http://localhost:8080/). The landing page provides a disclaimer (“By proceeding to the webshop you accept the terms and conditions”) and the total number of items available for sale in the shop. The page should be generated on the server via the HTML templating mechanism of Django. b. **Shop page:** accessible via ‘/shop’ contains all the other functionalities of the webshop. The page should be implemented as a single-page application (SPA) using React.
2p	2. **Automatic DB population (MANDATORY):** on the landing page (for grading purposes) there should be a link or button from where any anonymous visitor should be able to automatically populate the DB with 6 users, of which 3 users (i.e. sellers) own 10 items each. **Before each re-population, the DB should be emptied**. The generated users should have the following format: a. Username: testuser# b. Password: pass# c. Email address: testuser#@shop.aa After the data is generated the landing page should be updated.
3p	3. **Browse (MANDATORY):** Any user can see the list of items for sale. Sold items should only be listed on the seller’s page. a. The item (graphical) component should have at least this information: i. Title ii. Description iii. Price iv. Date added b. The API should return **max 10** items per request. You are free to decide how the next set of items is fetched: click “load more”, pagination, infinite scrolling. Every refresh of the view should result in an API request. c. The items should be ordered by the creation date (new-to-old).

3p	**4. Search:** Any user can search for items by **<u>title</u>**. Sold items should only be listed on the seller's page. The API should return max 10 items per request. You are free to decide how the next set of items is fetched: click "load more", pagination, infinite scrolling. The items should be ordered by the creation date (new-to-old). Note: any search action should result in a request to the API.
2p	**5. Create account (MANDATORY):** Users should be able to create an account by setting username, password, and email address. *<u>For making evaluation easier do not enable strong password authentication and do not check emails against regular expressions.</u>*
2p	**6. Login (MANDATORY):** a registered user can log in with a username and password
2p	**7. Add item (MANDATORY):** an authenticated user can add a new item to sell by providing. a. Title b. Description c. Price The date of creation should be automatically saved on the backend.
3p	**8. Add to cart (MANDATORY):** An authenticated user can select an item for purchase by adding it to the cart. A user (buyer or seller) cannot add to cart its own items. The item should still be available for other users to search for and add to their cart.
1p	**9. Remove from cart:** an item can be removed from the cart by the buyer.
6p	**10. Pay (MANDATORY):** the buyer sees the list of items to be purchased. When pressing the "PAY" button: a. If the price of an item has changed for any item in the cart, the cart transaction is halted and i. a notification will be shown next to the item, and ii. the displayed price should be updated to the new price. b. If an item is no longer available when the user clicks 'Pay', the whole cart transaction is halted and a notification is shown to the user without removing the item from the cart. The user can manually remove the unavailable items and then Pay. c. On a successful Pay transaction, the status of each item in the cart becomes SOLD. The bought items are listed as the buyer's item (but they are not available for sale). d. Upon successful transaction, emails are sent to the seller of every item in the cart and the buyer to notify which items are sold and bought.
3p	**11. Routing:** The Shop page should be implemented as a SPA. One should be able to navigate from the browser (bar) to the following links: a. Shop "/shop" b. SignUp "/signup" c. Login "/login" d. Edit Account "/account" e. MyItems: "/myitems"

1p	**12. Edit Account:** an authenticated user should be able to change the password of the account by providing the old and the new password.
3p	**13. Display inventory:** an authenticated user should be able to visualise his/her own items: on sale, sold, and purchased. Pagination is not required. The inventory should be filtered by the status of the item (sale, sold, purchased). You are free to decide how the filtering is implemented (checkbox, link, button, dropdown, etc)
2p	**14. Edit item:** the seller of an item can edit the price of the item as long as the item is on sale (available), via the *Edit* button, regardless of the item being added to any other buyers' carts.
	Non-Functional Requirements
1p	**15. Responsive:** The web page should look nice both on computer screen width>= 760px and phone width< 760px.
2p	**16. Security:** Only authenticated users can buy and sell. For simplicity, we use HTTP only.
1p	**17. Usability:** the site should be intuitive and easy to use
	Technical Requirements
0p	**18. Backend (Mandatory):** a. Backend uses Django, b. Django serves JSON to the shop page and HTML to the landing page c. We use SQLite database for this project. d. Use console backend for email (for grading purposes)[1]
0p	**19. Frontend (MANDATORY):** uses React JS
0p	**20. Browser (MANDATORY):** The web app should work on the latest Google Chrome.
0p	**21. Project folder (MANDATORY)**: a. the root folder of the project should contain a ***readme.md*** file listing i. Your name and email address ii. Which requirements have been implemented b. root folder must contain **requirements.txt** for backend c. The frontend folder should contain the **package.json** file with all used packages d. The frontend folder should include both **src** and **build** folders

Total: 40 points

- 24 mandatory points → grade 1
- 16 optional points → grade 2-5

[1] *It is meant that you should not use your own email server for sending emails eg via yahoo, gmail, etc but rather a non-production backend that does not require authentication and if we run your project locally we can see easily what emails have been sent on different actions* https://docs.djangoproject.com/en/3.1/topics/email/#console-backend

Note: All MANDATORY requirements must be satisfied in order for the project to be graded. The projects not satisfying them will not be graded. All requirements that are not MANDATORY, are OPTIONAL. You decide which ones to implement to get the desired number of points.

Grading table:

Final grade	Percent of max points	Points
0	0-50%	0-20
1	50-60%	20-24
2	60-70%	24-28
3	70-80%	28-32
4	80-90%	32-36
5	90-100%	36-40