Bidhub

Μπέλλος Πύρρος 1115202000141

Μελαμπιανάκης Νικόλαος 1115201900109

Πίνακας Περιεχομένων

1. Εισαγωγή

Η εργασία αυτή έχει ως αντικείμενο την ανάπτυξη μίας εφαρμογής ηλεκτρονικών δημοπρασιών. Στόχος είναι να δημιουργηθεί ένα ολοκληρωμένο σύστημα στο οποίο οι χρήστες θα μπορούν να εγγράφονται, να καταχωρούν αντικείμενα προς πώληση και να συμμετέχουν σε δημοπρασίες υποβάλλοντας προσφορές.Επιπλέον υπάρχει η λειτουργικότητα για την αξιολόγηση κάθε χρήστη και έχει αναπτυχθεί και αλγόριθμος συστάσεων.

Στο πρώτο κεφάλαιο θα επικεντρωθούμε στο Front End, αρχίζοντας από το περιβάλλον ανάπτυξης τξς εφαρμογής. Έπειτα θα δούμε τις γενικότερες σχεδιαστικές μας αποφάσεις σε σχέση με την δομή και την εμφάνιση της πλατφόρμας. Επιδιώξαμε να αναπτύξουμε ένα ευδιάκριτο design, με έμφαση στη καθαρότητα και τους λίγους αντιπερισπασμούς, οδηγώντας σε μία προσβάσιμη πλατφόρμα από ένα μεγαλύτερο κοινό.

Στο δεύτερο κεφάλαιο θα κοιτάξουμε το Back End, τη δομή που ακολουθήσαμε στον σχεδιασμό του. Προτεραιότητά μας ήταν να δημιουργήσουμε ένα καθαρό και καλά οργανωμένο Back End, ώστε να αποτελέσει σταθερή βάση για όλη την εφαρμογή. Ακολουθήσαμε αντικειμενοστραφή αξιώματα, με στόχο να υπάρχει σωστή διάκριση αντικειμένων, επαναχρησιμοποίηση κώδικα και ευκολία επεκτασιμότητας στο μέλλ

2. Front-End

Αρχίσαμε τη διαδικασία ανάπτυξης του Front End αποφασίζωντας σε ένα συνδιασμό τεχνολογιών που θα μας επέτρεπε να δουλέψουμε αποδοτικά και να οργανώσουμε τον κώδικα μας κατάλληλα. Χρησιμοποιήσαμε την **React** για την τμηματικότητα της,για την συμβατότητα της με λοιπές τεχνολογίες, που της επιτρέπει να ενσωματώνεται εύκολα σε διαφορετικά εργαλεία και βιβλιοθήκες καθώς και για την δημοτικότητα της που μας έδωσε πολύ βοήθεια σε διάφορα προβλήματα που αντιμετωπίσαμε.

Το build του project πραγματοποιήθηκε με το **Vite**, το οποίο εκτιμήσαμε ιδιαίτερα, καθώς μας εξοικονόμησε πολλά περιττά refresh και rebuilds κατά την ανάπτυξη του Front-End, κάνοντας τη διαδικασία πιο ομαλή και παραγωγική.

Χρησιμοποιήθηκε το **JWT** όπως ζητήθηκε και για τη διαχείριση του δημιουργήσαμε το αρχείο **AuthContext.jsx,** το οπόιο διατηρεί τις πληροφορίες της σύνδεσης του χρήστη, και χρησιμοποιείται από όλη την εφαρμογή.Αντίστοιχο ρόλο επιτελεί το **UseApi.jsx,** το οποίο περιέχει όλες τις κλήσεις προς το API, με το αντίστοιχο handling των tokens και τυχών errors.

Για την αισθητική της σελίδας αποφασίσαμε να κρατήσουμε μια εύκολη στη χρήση εφαρμογή, χωρίς περιττά στοιχεία που θα αφαιρούσαν από τον minimal σχεδιασμό που επιδιώκαμε. Ένα πιο φορτωμένο οπτικά design, θα έκανε τις βασικές λειτουργίες λιγότερο εύκολα προσβάσιμες αλλά και θα αφαιρούσε από την σοβαρότητα και αξιοπιστία που πρέπει να έχει μια σελίδα δημοπρασιών.

Για αυτό το λόγο και για να αποφύγουμε και την τυποποίηση της πλατφόρμας μας, αποφασίσαμε να αποφύγουμε να χρησιμοποιήσουμε ένα styling framework και να βασιστούμε σε custom CSS, παίρνωντας κάποια από αυτά από σελίδες όπως την https://css-tricks.com/.

Αποφασίσαμε να υλοποιήσουμε την απαίτηση του Messaging σε μορφή παρόμοια των social media, προτιμώντας αυτή την λύση για την αμεσότητα της. Επιτρέπει την εύκολη ομαδοποίηση των συζητήσεων με βάση την δημοπρασία για την οποία γίνεται αλλά και την εύκολη χρήση σε περίπτωση συνάντησης των χρηστών για αγοραπωλησία. Είναι μια παραδοχή της εργασίας μας, παρόλα αυτά πιστεύουμε πως σαν design είναι πιο προχωρημένο από ένα Messaging style το οποίο θυμίζει e-mail

3. Back-End

Στο Backend χρησιμοποιήσαμε το Django και προσθέσαμε το Django REST Framework. Το πρώτο μας διαλέχθηκε για τις έτοιμες λειτουργίες σχετικά με την διαχείριση χρηστών και συγχρονισμού βάσεων, ενώ το δεύτερο τη βάση για το API μας με το REST, και τις βασικές λειτουργίες που χρειάστηκαν για τις CRUD ενέργειες που απαιτήθηκαν.

Χρησιμοποιήσαμε και την δοσμένη βιβλιοθήκη SQLite 3, για την ευχρηστία της σε σχέση με την Django, ιδιαίτερα σε φάση ανάπτυξης. Για κατάσταση παραγωγής έχει γίνει μια αρχική εγκατάσταση της PostgreSQL καθώς και ενός αντίστοιχου library(psycopg2).

Ορίσαμε αρχικά διάφορα μοντέλα για την σχεδίαση της βάσης. Δεν ασχοληθήκαμε καθόλου με SQL, καθώς οι διάφορες βιβλιοθήκες του Django REST Framework διαχειρίζονται την βάση από μόνα τους. Γενικά η εφαρμογή μας στο νωτιαίο άκρο μπορεί να χωριστεί σε 3 βασικά μέρη + κάποια άλλα έξτρα, τα οποία θα κατηγοριοποιήσουμε. Τα 3 αυτά μέρη είναι, τα Μοντέλα (οι σχέσεις που ορίζουν την βάση μας), οι Serializers και τα Views. Για τα μοντέλα δεν έχουμε να προσθέσουμε κάτι, ο κώδικας είναι αρκετά ευανάγνωστος από μόνος του.

Οι Serializers είναι αυτοί που είναι υπεύθυνοι για την διαχείριση δεδομένων από τον client, καθώς και για την επικύρωση των δεδομένων. Για αυτό, μπορεί για ένα μοντέλο να υπάρχουν διάφοροι Serializers, ανάλογα με την ενέργεια και τα δεδομένα που χρειαζόμαστε από αυτό. Πχ, το μοντέλο Item που υποδηλώνει το αντικείμενο μιας δημοπρασίας έχει διάφορους Serializers, καθώς είναι διαφορετική η ροή δεδομένων που χρειαζόμαστε όταν θέλουμε να πάρουμε μια λίστα από Items και διαφορετική όταν θέλουμε να δημιουργήσουμε ή ενημερώσουμε ένα Item. Επίσης, βοηθούν στην απόκρυψη δεδομένων, καθώς διαφορετική πρόσβαση στα δεδομένα έχει ένας ιδιοκτήτης και διαφορετική ένας απλώς χρήστης. Μια επιπλέον δουλεία τους, είναι η επικύρωση, η οποία επικυρώνει μεμονομένα κάθε στοιχείο (πχ η ημερομηνία λήξης να είναι αργότερη του σήμερα), αλλά επίσης ελέγχει τις σχέσεις μεταξύ δεδομένων και αν αυτές είναι δεκτές.

Τα viewsets από την άλλη, είναι η δομή απαραίτητη για την επεξεργασία request και δημιουργία endpoint στο νωτιαίο άκρο. Μέσω αυτόν κατοχυρώνεται τι ακριβώς συμβαίνει όταν έρχεται ένα request από τον server. Συνήθως χρησιμοποιούνται μαζί με τους serializers για την σαφή διαμέριση των υποχρεώσεων. Τα viewsets σε αυτή την περίπτωση έχουν ως ρόλο την επιβολή των δικαιωμάτων του χρήστη (πάνω σε ποιά αντικείμενα μπορεί να κάνει αυτη την πράξη ή αν μπορεί καν να την κάνει), το filtering και την διαχείριση διάφορων query parameters. Όμως μπορεί να γίνει και όλη η δουλεία μέσα από αυτούς, κάτι που δεν συνιστάται, αλλά έχουμε χρησιμοποιήσει σε συγκεκριμένες περιπτώσεις, που η διαχείριση ήταν πολύ απλή ή αντιθέτως περίπλοκη, με την έννοια ότι ήταν πολλά μοντέλα μπλεγμένα σε αυτό το request, οπότε η πιο απλή λύση είναι η διαχείρισή τους σε ένα εννιαίο μπλοκ κώδικα.

Πέρα από αυτά τα βασικά, χρησιμοποιήσαμε signals και schedulers για προφανείς λόγους. Αρχικά τα signals χρησιμοποιήθηκαν για την αυτόματη ενημέρωση του μέσου όρου αξιολόγησης ενός χρήστη, ενώ οι schedulers για την αυτόματη ενημέρωση των δημοπρασιών καθώς και την ενημέρωση του αλγόριθμου συστάσεων.

Ο αλγόριθμος συστάσεων υλοποιήθηκε με την τεχνική του matrix factorization. Αναλύει τη συμπεριφορά των χρηστών (επισκέψεις και προσφορές) για να προτείνει αντικείμενα που είναι πιθανό να τους ενδιαφέρουν. Οι επισκέψεις και οι προσφορές έχουν ξεχωριστό βάρος, με τις επισκέψεις να έχουν το ⅓ του βάρους των προσφορών. Μέσω αυτών δημιουργείται ο βασικός πίνακας μεγέθους (πλήθος χρηστών) X (πλήθος αντικειμένων). Στόχος του αλγόριθμου matrix factorization είναι η κατσκευή 2 μικρότερων πινάκων, ο καθένας με μέγεθος (πλήθος χρηστών/αντικειμένων) X (πλήθος "κρυμμένων" χαρακτηριστικών), τους οποίους αποθηκεύουμε ως πίνακες npy και χρησιμοποιούμε για να προβλέψουμε την αξιολόγηση μεταξύ χρήστη και αντικειμένου, πολλαπλασιάζοντας την γραμμή που ανήκει στον χρήστη, με την ανάστροφη γραμμή που ανήκει στον αντικείμενο (δηλαδή στήλη).

Για την εκπαίδευση του μοντέλου χρησιμοποιείται Stohastic Gradient Descent (SGD) με:

- **Λανθάνοντα Χαρακτηριστικά**: 5 (προεπιλογή)
- **Regularization**: λ = 0.001 για αποφυγή overfitting
- **Adaptive Learning Rate**: Μειώνεται σταδιακά κατά την εκπαίδευση
- **Early Stopping**: Σταματά την εκπαίδευση όταν δεν υπάρχει βελτίωση

Δεδομένου της αραιότητας του βασικού πίνακα και της φύσης του προβλήματος μας, που μας ενδιαφέρει μόνο το πόσο καλό είναι ένα αντικείμενο για έναν χρήστη, καθώς δεν μπορούμε να βγάλουμε άκρη για αρνητικές συμπεριφορές μέσα στο πρόγραμμά μας, οι τιμές του πίνακα είναι μη μηδενικές. Η εκπάιδευση γίνεται πάνω σε αυτές, υπολογίζοντας το RMSE (Round Mean Square Error) μόνο για αυτές. Χρησιμοποιούμε gradient clipping για να μη "σκάσουν" τα gradients, με αποτέλεσμα να σπάσει το πρόγραμμα. Η αρχικοποίηση των διανυσμάτων είναι με μη μηδενικές τιμές λίγο μεγαλύτερες του 0.

Για την αποφυγή Overfitting, έχουμε χρησιμοποιήσει τα παρακάτω:

- **L2 Regularization**: Προσθέτει ποινή για μεγάλες τιμές παραμέτρων
- **Train/Validation Split**: 80/20 για παρακολούθηση απόδοσης, η οποία σε κάθε επανάληψη (epoch) αλλάζει τυχαίως, για αυτό και το RMSE δε βελτιώνεται σε κάθε επανάληψη ομαλά, αλλά ανεβοκατεβαίνει (προφανώς κατα μέσο όρο ακολουθεί μια μείωση με την πάροδο των epochs)
- **Early Stopping**: Σταματά όταν η απόδοση σταματά να βελτιώνεται

Αφού λοιπόν φτιαχτούν οι πίνακες users.npy και items.npy, η διαδικασία για την ταξινόμηση των αντικειμένων με βάση τις προτιμήσεις ενός χρήστη είναι η εξής:

Για κάθε χρήστη, το σύστημα:

1. Φορτώνει τα εκπαιδευμένα διανύσματα
2. Υπολογίζει το εσωτερικό γινόμενο μεταξύ του διανύσματος του χρήστη και όλων των ενεργών αντικειμένων
3. Ταξινομεί τα αντικείμενα κατά φθίνουσα σειρά βάσει της προβλεπόμενης αξιολόγησης

4. Επίλογος

Η ανάπτυξη της πλατφόρμας μας αποτέλεσε μια μεγάλη διαδικασία που ξεκίνησε με εκτενή έρευνα πάνω στα πιθανά εργαλεία και τεχνολογίες που είχαμε στη διάθεση μας. Είχαμε ως στόχο την δημιουργία ενός περιβάλλοντος εργασίας που θα μας επέτρεπε να δουλέψουμε αποδοτικά αλλά και θα μας εξασφάλιζε την επεκτασιμότητα της εφαρμογής στο μέλλον.

Σημαντικό είναι να αναφέρουμε ότι χρησιμοποιήσαμε και Github που μας επέτρεψε την ασύγχρονη δουλειά πάνω στην εργασία, την αποθήκευση προηγούμενων εκδόσεων και την οργάνωση το κώδικα μας. Ακόμα και η συγχώνευση των αλλαγών μέσω Pull Requests αποτέλεσε σημαντικό εργαλείο για τη συνεργασία μας καθώς μας επέτρεπε να μαθαίνουμε σε τι σημείο έχει φτάσει ο συνάδελφος μας σε κάθε φάση.

Μια μεγάλη ερώτηση στο Front End ήταν η διαχείριση των tokens και της κατάστασης του χρήστη. Η επιλογή κεντρικοποιημένης διαχείρισης έγινε σύμφωνα με τα industry standard ώστε η εφαρμογή να ακολουθεί βέλτιστες πρακτικές και να είναι εύκολη στη συντήρηση και επέκταση.

Επιπλέον, η χρήση custom CSS δημιούργησε το ρίσκο έλλειψης συνέπειας στη συνολική φιλοσοφία της αισθητικής της πλατφόρμας μας. Προς αποφυγή αυτού αποφασίσαμε σε κάποια προκαθορισμένα χρώματα και βασικούς κανόνες σχεδιασμού. Παράλληλα διατηρήσαμε συνεχή επικοινωνία μεταξύ μας για κάθε απόφαση που αφορούσε το UI και τη συμπεριφορά της εφαρμογής, ώστε το αποτέλεσμα να είναι μια ομοιόμορφη και επαγγελματική εμπειρία χρήσης.

Στο νωτιαίο άκρο της εφαρμογής λόγω του Django χρειάστηκε να έχουμε μια σωστή αποτύπωση των σχέσεων των αντικειμένων ώστε να παραμένουν οργανωμένα και κατανοητά. Η χρήση διαφορετικών serializers ανα use case μας επιτρέπει να ελέγχουμε ποια δεδομένα στέλνονται για ένα μεγαλύτερο βαθμό ασφάλειας.

Πάνω σε αυτά χρειάστηκε και η κατάλληλη υλοποίηση των permissions, και τη διαχείριση των ρόλων. Παραδείγματος ένας απλώς χρήστης βλέπει μόνο τα απαραίτητα στοιχεία μιας δημοπρασίας, ενώ ο δημιουργός της ή ένας admin έχει πρόσβαση σε παραπάνω πληροφορίες.

Εκτέλεση

Το project μπορεί ενδεικτικά να εκτελεστεί ακολουθώντας τα παρακάτω βήματα σε μηχάνημα Linux. Σε Windows είναι διαφορετική η δημιουργία του python virtual environment(venv). Να σημειωθεί πως δεν έχουμε περάσει τη βάση μέσα στα αρχεία (λόγω μεγέθους), οπότε διάφορες λειτουργικότητες μπορεί να μη λειτουργούν/είναι ακριβείς.

1. Μέσα στο project_auction_2000141_19000109 καλέστε το αρχείο setup.sh (δώστε του δικαιώματα εκτέλεσης με chmod +x αν δεν έχει ήδη).
2. Ανοίξτε 2 διαφορετικά terminal, στο ένα θα τρέχει το backend και στο άλλο το frontend.
3. Για το backend:
    a. source venv/bin/activate
    b. cd auction/
    c. python manage.py runserver
4. Για το frontend:
    a. cd frontend/
    b. npm run dev
    c. Μέσα στην κονσόλα που ανήγει, πατήστε: o, και μετά enter