Commerce Procesboek

12 december

**Lecture sql**

Sql herkende ik nog van het vak datarepresentaties, in dit vak hebben we een aantal opdrachten gemaakt over dit onderwerp. Grotendeels was dit wel al weggezakt in mijn geheugen dus de lecture was wel nodig/interessant. Tijdens deze lecture probeer ik gelijk mee te schrijven om alle queries en arguments zo goed mogelijk te onthouden. De vraag die in het begin van de lecture bij me opkomt is hoe we deze tables gaan implementeren met Django in de websites. De uitgebreide SQL syntax blijkt voor het commerce-project minder nodig te zijn om te kennen, daar Django dit voor ons doet door middel van modellen. In models.py, in de map van onze app, kunnen we classes aanmaken voor elke tabel. Deze tabellen kunnen we dan met elkaar verbinden met Foreign keys en zo kunnen we een heel systeem van modellen van data maken. Na het maken van de modellen is het nodig de migrations aan te maken in python. Hiermee wordt de python code omgezet in een sql file, wat we daarna kunnen aanpassen wanneer we python manage.py shell gebruiken. Nu kunnen we met behulp van python onze databases aanpassen, updaten, dingen deleten etc. Met de Django admin app kunnen we hierna op een nog makkelijkere manier data toevoegen aan onze modellen. Verder leren we ook nog hoe we gebruikers aanmaken en login/logout functionaliteiten toevoegen aan de applicatie.

**13 december**

Na het doorlezen van de specificaties, en het documenteren van een vaag/geschetst plan heb ik een stuk meer inzicht in hoe de applicatie er ongeveer uit komt te zien en hoe hij hoort te werken. Het eerste waar ik mee begon was nadenken over hoe de verschillende modules op elkaar aansluiten, en welke velden elke module nodig heeft. De module die al aangemaakt is, User, is gemaakt met als input Abstractuser. Ik moest even op de Django site kijken hoe deze precies werkt. Waar ik achter kwam is dat dit het default user model is van django, maar dat je deze op de manier hoe hij er nu staat altijd later nog kan aanpassen als dat nodig is. Ook moet het model nog toegevoegd worden in de admin.py.

Wat ik nu heb gedaan is in models.py de modellen aangemaakt met de fields waarvan ik denk dat ik ze nodig heb, en de Foreignkeys waarmee ik denk dat het systeem uiteindelijk goed kan werken. Hierna heb ik mijn aanpassingen gepusht naar github en ben ik gaan kijken naar het onderdeel Create Listing. Hiervoor moest ik een aantal forms maken in de createnew.html, en met de post method deze informatie sturen naar een functie in views.py waar de informatie omgezet werd naar data die door django naar sql data gezet kan worden. Telkens als ik dit probeerde kreeg ik een error, maar na een half uur proberen toch maar even in de lecture gekeken. Blijkbaar was ik vergeten mijn veranderingen in modules.py te migraten….. zo kan je blijven proberen natuurlijk. Hierna lukte het wel om een list aan te maken en via de admin kon ik deze zien en ook weer verwijderen.

Toen kon ik beginnen aan het deel Active Listings Page. Hiervoor moet ik eerst weten welke listings er active zijn. Om dit te weten moet ik een field aanmaken voor de list class. De default mode voor deze class zet ik op True, omdat alleen de owner pas op de listing page eventueel de listing kan deactivaten, en hier kom ik pas later in de opdracht aan toe. Hierna ging ik de index pagina aanpassen zodat de listings goed weergeven worden. Wat trouwens hierbij mijn idee is is dat ik eerst alle funcionaliteiten in orde wil hebben voordat ik me druk ga maken om hoe het eruit ziet op de website. Ik filter in views.py welke listings active zijn, en deze stuur ik vervolgens index.html in. Ik vond het erg lastig hoe ik de image erbij kon krijgen, en ook wist ik niet of de image wel in het goede formaat zou komen. W3schools is een enorm handige website om html gerelateerde vragen in opzoeken. Hier kwam ik erachter dat ik met de src functie in een img gewoon de normale url kan invullen, en zo een image kan tonen. Ik wou ook niet dat mijn auction te groot werd door bijvoorbeeld de description, dus maakte ik een class in styles.css aan die ik hiervoor kon gebruiken.

Dit deel van de opdracht verleidde me toch wat om aan de slag te gaan met bootstrap en de layout, maar hier ben ik toch niet helemaal in opgegaan omdat het eerst afmaken van dat alles werkt toch belangrijker is. Op dit moment zijn de active auctions te zien op de index page zoals ik wil.

**14 December:**

De volgende stap is het maken van de pagina voor de listings zelf. Deze pagina moet natuurlijk als input krijgen welke listing moet worden weergegeven. Ik moest er best even over nadenken hoe ik dit ging doen, maar in de lecture deden ze iets als {{ url ‘listing’ ….. }} waar op de stippellijn iets moet komen dat als argument aan de url gegeven wordt. Ik vroeg aan een medestudent of ik hiervoor handmatig iets moest aanmaken als een id voor listings of iets dergelijks, omdat je hier bepaalde relaties tussen de modules gebruikt. Hij zei me dat dit automatisch wordt aangemaakt dus ben ik gelijk gaan expirimenteren met deze methodes. Zoals in de lecture flight\_id doorgestuurd werd probeer ik dat hier dus te doen met listing\_id. Natuurlijk moest ik ook een path aanmaken die dit id bevat. Wat ik deed om te controleren of ik echt zomaar een listing.id door kon sturen zonder dat ik hiervoor een field aangemaakt heb binnen de module, heb ik op de index page {{ listing.id }} geprint. Door eerst alleen de listing.title te printen op de pagina waar ik naar doorverwezen werd kon ik controleren of deze link goed werkte.

Ik besloot om een extra model aan te maken waar een bepaalde listing is gekoppeld aan een user, om op deze manier een watchlist samen te stellen. Ik kwam erachter dat het me op deze manier niet ging lukken om een watchlist te maken, dus besloot ik even terug te kijken in de lecture of ik misschien op een andere manier een watchlist kon maken. Blijkbaar lijkt de watchlist functie best een beetje op het passenger on flights gedeelte van de lecture waar ze manytomany relaties gebruiken. Uiteindelijk is het me gelukt om twee verschillende functies in views.py aan te maken die ervoor zorgen dat een item toegevoegd en verwijderd kan worden. Eerst probeerde ik dit in een functie te stoppen, maar dit werkte niet zoals ik wou.

Volgende stap wat het bieden implementeren. Hiervoor maakte ik een django form aan die het huidig hoogste bod doorgeeft. Ik kwam er hier achter dat ik de startprice in het listing model moet veranderen naar een Foreign key van de type bid. Uiteindelijk was deze eerste optie niet wat ik gebruikt heb om alles werkend te krijgen. Ik maakte wel een Django form aan, en vervolgens maakte ik een functie aan genaamd placebid die een beetje dezelfde indeling heeft als de createlisting functie. Het lastige gedeelte van dit deel vond ik hoe ik bepaalde delen van de data kon bereiken in python. Bijvoorbeeld hoe ik het bod met de hoogste price kon filteren uit alle boden. Hier had ik even moeite mee maar op de Django website stond wel nuttige informatie over hoe je in Django je SQL data kan bewerken/ophalen. Wat nu nog niet lukt bij mij is hoe ik het hoogste huidige bod op de indexpagina kan krijgen bij de active listings. Hier zit nog wel wat uitdaging.

Het gedeelte om de listing te closen was nu aan de beurt. Tussen het vorige onderdeel en deze heb ik trouwens overal user.is\_authenticated toegevoegd en in views.py login\_required, om ervoor te zorgen dat ingelogde gebruikers meer toegang hebben dan gebruikers die niet ingelogd zijn. De form op de pagina van de listing die je can closen moet er natuurlijk alleen staan voor de owner van de listing. De form heb ik op dezelfde manier gemaakt als de button voor de wishlist.