Trader Interactive Assignment

Overview

A series of webpages that searches for listings that match a potential buyer's input from a web form. Allows a buyer to search for a vehicle. Clicking on a Listing within the search results will display the details about the vehicle and information about the Seller. It will also display reviews for the Seller and allows a potential buyer to email the Seller via a contact form.

Assumptions

The buyer is an anonymous user. Seller reviews can be posted by anonymous users. So this design does not include buyers as registered users of the system.

Technology choices

Here I talk about PHP based technologies because that is the primary language I work with.

Laravel with Eloquent: For ORM and queues for email sending.
Database: A relational DB like Oracle/MySql/PostGreSql for the primary entities.
ElasticSearch: Proposed for better text based searches. More details in later section on designing the Search.

Email Integration

Record outbound emails in DB and drop them in a queue for sending. This is so that “email sent” confirmation to the end user can be provided before waiting for the email sending to be completed.

Search

Rudimentary search with all listing data in a single table

1. Vehicle type, price range searches will work well with this.
2. Searching over keywords is problematic. The queries will require full table scans even if you can find a way to write queries that can get accurate data for the given keywords (I’ve not been able to find a way to search over all the keyword/description columns with various AND OR LIKE combinations to provide reasonable results).

This solution is definitely not acceptable.

Maintain a separate index of keywords to listings. Update this index as listings are added/modified/deleted.

A separate table (dictionary) with keywords (KEYWORD). The dictionary is indexed by the keywords. It is mapped to listing ids via a table (KEYWORD_X_LISTING).
On every listing add/update/delete, update the table KEYWORD_X_LISTING.
Pros: Faster searches w/o table scans.
Cons:

1. Searches on keywords that are not in the dictionary would be ignored.
2. Searching thru description would not work.
3. Phrases like “no damage” would not yield accurate results.

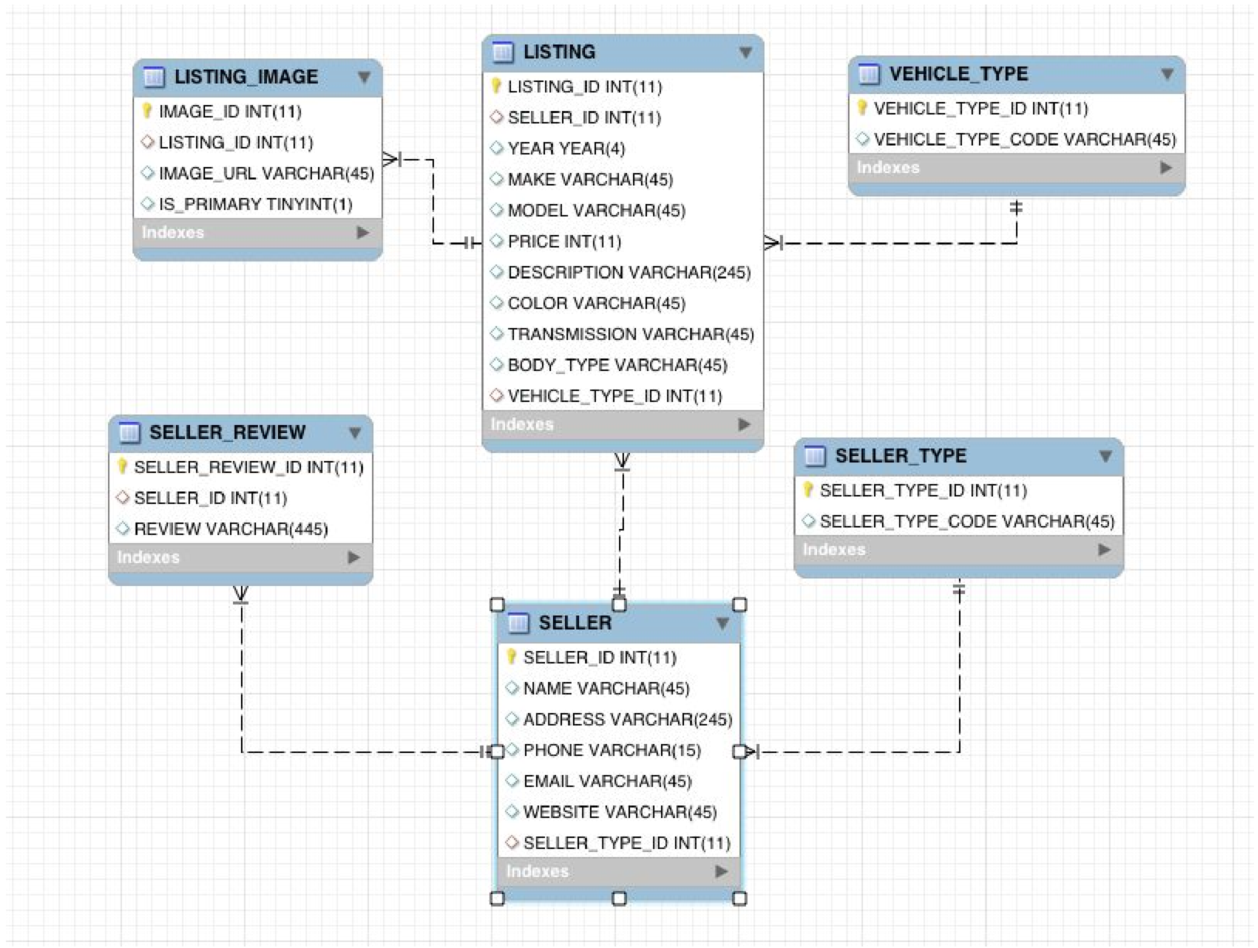
Some options to mitigate some of the cons, would be:
Any search words entered by a buyer that don’t match a keyword would be recorded in the DB. These can be added into the main dictionary (KEYWORD) either after review manually, or via a job that can be configured to add keywords to the main dictionary (KEYWORD) if it has appeared in searches X number of times.

Use a combination of DB and a third-party text search engine like ElasticSearch

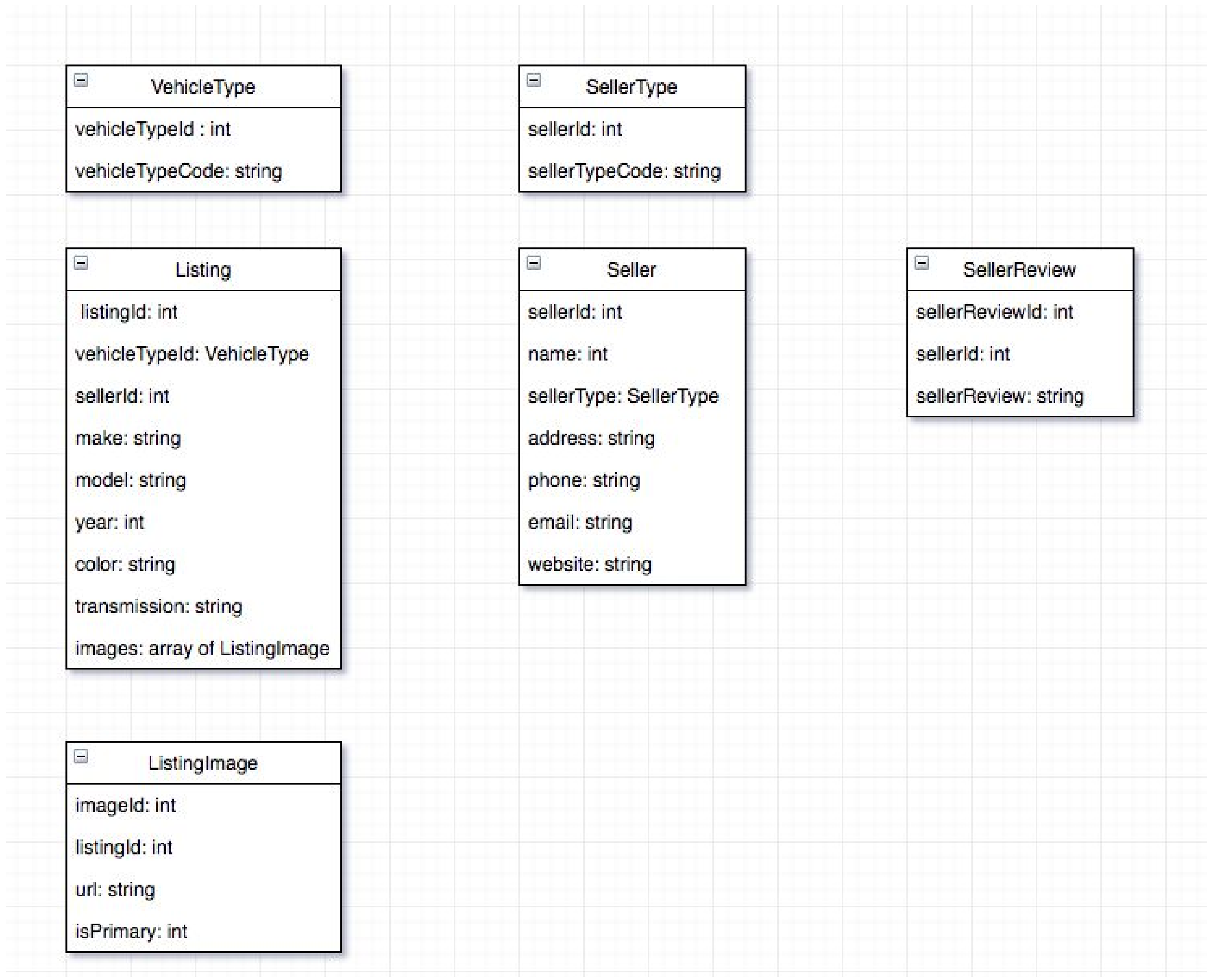
Based on some reading online, ElasticSearch appears to have features like keyword + full text search of indexed records. We could use a DB search for vehicle type and price range intersecting with a keyword/text search of the listing documents in ElasticSearch.

On every listing add/update/delete, we could add/update/delete listing document from ElasticSearch via their REST API.
Pros: ElasticSearch is open source and available on AWS as a service. The text/keyword search problem has been solved elegantly allowing phrases and relevance ordering.
Cons: Price and overhead of making calls to an external system on every add/update/delete and search with keywords.

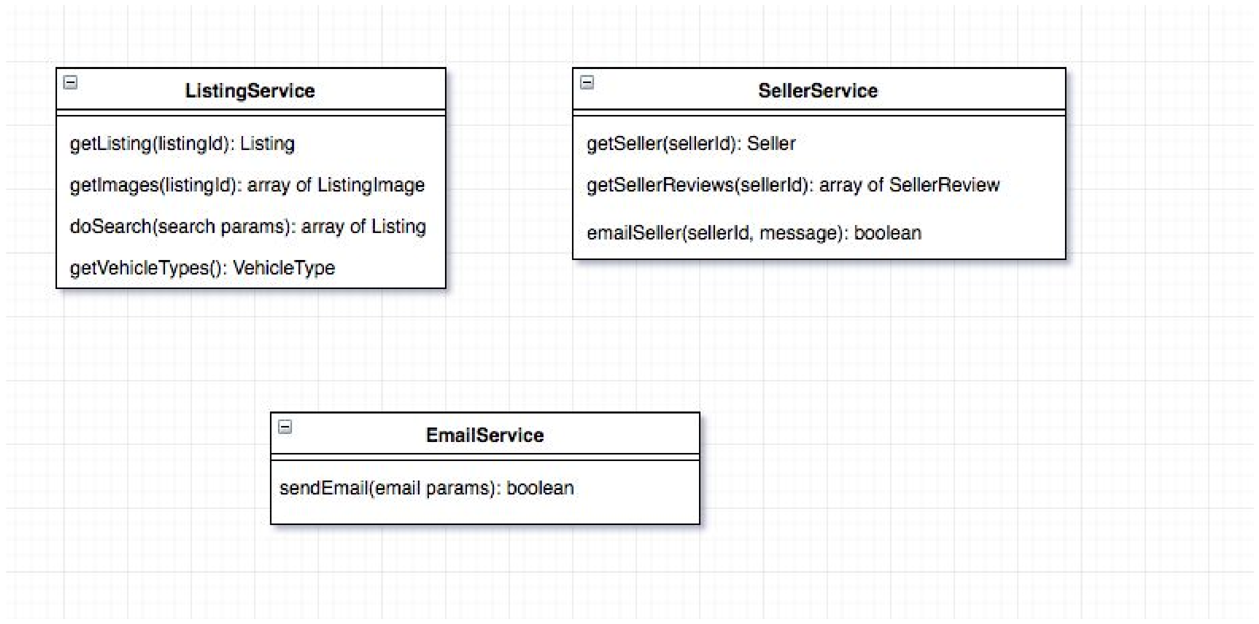
Data Model



Domain classes



Services



ListingService
getListing(listingId): Listing
getImages(listingId): array of ListingImage
doSearch(search params): array of Listing
getVehicleTypes(): VehicleType
SellerService
getSeller(sellerId): Seller
getSellerReviews(sellerId): array of SellerReview
emailSeller(sellerId, message): boolean
EmailService
sendEmail(email params): boolean