Data preparation.
First of all, I cut out all non English sentences from the file.
I used *langdetect* package for Python.
Unfortunately it there was over 50% reduction in size of the data (48483 → 20123).
As many entries contain unidentifiable language or solely emojis.

Next, I preprocessed all the rows turning each word into its lemma, with help of *nltk*.

Modeling
In order to put the comments in some kind of vector space model, I chose doc2vec.

As other options to consider for this kind of task, I could think of averaged world2vec model or tf idf model. In case of longer comments, it might be an idea to try how BM25 kind of document scoring function might perform.
However, out of time constraints, I did not manage to test them all.

Model training
I decided to train the doc2vec model on the 20123 entries of data I had left.
As I observed the results, it actually might have not been the best idea.
As I imagine, 20123 comments may not be enough to train a model fully reflecting the patterns in whole language (a typical dictionary contains 50-100 k words, so assuming a standard comment has a length of 10 words, therefore whole 'corpus' has around 200k of words, meaning that in optimistic case each word of dictionary is used only 2-4 times). Taking into account that stopwords may make the majority of corpora, non-stopwords may be used ever more rarely.

A one possible solution to this problem, might be transfer learning.
I thought about collecting a already prepared model, trained on some large corpora, and then 'fine tuning' this model, using only our comment dataset.
Nevertheless, I did not have enough time to test out this idea, so this is just a proposal.

The code for model creation is located in *brandbastion.core.similarities.doc2vec create_embeddings.*
I decided to use doc2vec model from *gensim* package.
I chose 300 hidden neuron for doc2vec embeddings, and trained it for 20 epochs with decreasing learning rate.
I think the hyperparameters, learning rate, epochs count might have to be modified, to create a better model.
Once the model is trained, *gensim* offers a function to transform any document (comment) into a vector space model form, with appropriate coefficients.

Data management
For small dataset like this, I decided to use MySQL database, however in real life cases, relational databases are not a good idea. Nevertheless, I attach the sql dump file.
In the database, I decided to keep only the original texts of comments, together with ID tag (which I changed from big number like in IG-comments.txt, to numbers starting from 1).
Coefficients for doc2vec vectors for each document, are stored in the model file, saved in *model* directory.

At the beginning of the program, all entries are transferred into a database table.
Then, during model training, they are taken from the table and fed into the algorithm.
Both of these processes are potential bottle necks if it comes to memory.

In order not to load whole file into memory, I used Python's generators, yielding one row / line at a time. The file may potentially be very very big file, but in our case only several hundreds of megabytes. So may be the row count, which means that doing it naively we might simply run out of RAM.

I added a functionality to add comment to the database.
However this brings up one problem. *Gensim*' doc2vec package offers functionality to calculate similarity between the query document and all documents in the database, out of the box.
That means, that finding similar documents to a query document, can be done very quickly (*brandbastion.core.similarities.doc2vec get_similar_comments*).
However, the similarity between the query comment, and the comments which were not used for training, yet exist in the database, has to be calculated explicitly. *Gensim* package uses cosine similarity function, and so did I. I calculated cosine similarity between query comment, and all comments from database which were not used for model training.
Therefore, when there are some new comments, and the query comment is being put, there is a necessity of merging two kind of results (most similar comments from *gensim's* doc2vec model and calculated explicitly, most similar comments).
However I did not implemented this option, it would be wise to store 300 doc2vec coefficients for every new comment. I can think of two options for that, one is another DB table referring to the comment table, which could contain all the necessary numbers.
At the time of finding most similar comments, program could calculate cosine similarity between query comment vector and all vectors of comments from database.
Now, even though the complexity of such calculation is linear (cosine for every comment in the database), there might be some room for improvements here.
For instance, we could choose one reference point (say (0,0,0,0,0,…,0)), and keep the similarities for vectors of all comments, relative to this point.
Then, when there is a query comment, and if we have a constraint to choose only 10 closest comments, we could use the relative similarity information by applying triangle inequality.
For instance, we know that the distance between two comments A and B is not smaller than the difference of relative distances between comment A - (0,0,…,0), and comment B – (0,0,…,0), which might be helpful in narrowing down the search of similarities.

Now, once in a while, it could be wise to recalculate doc2vec model completely, to take into account all the new comments, that came in after creating previous model.
As this could take a long time, I would do it using separate thread, after completion of which, the models would be replaced.

Architecure
I used Python's *flask* framework, to create simple REST API for this model to work.
It might be simply extended to support other functionalities, as well as it can be connected with some front-end, to make it usable for customers.

The *main.py* starts up a server, which has its code in *brandbastion.web.back.runweb.py*.
Each function in that file has a directive, telling related URL address.
transfer_to_db offers option to put data into the DB.
Recalculate recalculates the doc2vec model so that the new one is build using all available data.
Data can be use to add new comments to database.
Predict allows to get show similar comments to a given one. Now, it only chooses the most similar comments from those which were used to build the existing model. However, to make it work for all available comments (meaning as well for those which were added after model creation), I would have to add transformation into feature vector and explicit cosine similarity calculation (as explained before).