



Blue Fever, Inc [1]

Smart Video Picker System

A Video Searching System Based on Sentiment Analysis

Dixita Patel

Siyu Xie

08/03/2018

Table of Contents

System Overview	4
Current Situation	4
Alternative Idea.....	4
Key Technologies.....	4
System Architecture.....	5
System Detailed Design.....	6
Front-end Design	6
Back-end Design.....	7
Database Design.....	8
System Implementation.....	9
Files Detail.....	10
Running Server	13
Base Template	13
Home Page.....	14
Sign Up	14
Login & Logout.....	15
Forgot Password	16
Profile.....	17
Search & Results.....	18
Validation and Testing.....	21
Validation	21
Testing	21
Deployment.....	21
Conclusion.....	22
Future Work	22
Front-end Future Improvement.....	22
Back-end Future Improvement.....	22
Database Future Improvement	23
References.....	24

Table of Figures

Figure 1. System Architecture	5
Figure 2. User Interface Layout	6
Figure 3. Emotion/Situation Before Labels	7
Figure 4. Emotion/Situation After Labels.....	7
Figure 5. Model Parameters	7
Figure 6. Top5 Similar Words	8
Figure 7. auth_user Table	8
Figure 8. User Data	9
Figure 9. bffsvps_videoinfo Table.....	9
Figure 10. Videoinfo Data	9
Figure 11. Directory Structure	10
Figure 12. SVPS AutoFiles.....	10
Figure 13. bffsvps Files	11
Figure 14. Migration Files	11
Figure 15. HTML Files.....	12
Figure 16. Bootstrap Templates.....	12
Figure 17. Training Data.....	12
Figure 18. Run Server	13
Figure 19. Server is Running	13
Figure 20. Header & Footer	13
Figure 21. Home Page	14
Figure 22. Sign Up.....	14
Figure 23. Login	15
Figure 24. Logout	15
Figure 25. Forgot Password	16
Figure 26. View Profile	17
Figure 27. Edit Profile	17
Figure 28. Change Password	18
Figure 29. Search Page	18
Figure 30. Sample Case #1	19
Figure 31. Sample Case #2	20

Tables

Table 1. Validation Accuracy	21
---	----

Smart Video Picker System

System Overview

Current Situation

Blue Fever is a tech company that builds relationships with young women over text message to recommend media based on users' moods. The females who chat with the users are called Blue Fever Friends (BFFs). Through chatting with the users, BFF will have an idea about user's current situations or emotions (we call it 'Emotion/Situation Before') as well as the state of mind they want to be in (we call it 'Emotion/Situation After'). Then she searches videos from the Google sheet under various categories of emotions/situations and selects the one that matches both conditions.

However, it turns out that this video searching task is time consuming and has following limitations:

- BFFs cannot search multiple keywords(emotions) in Google sheet at the same time.
- Only the words inside the sheet are searchable, so if she searches anything synonymous to the existing keyword, she will not find the suitable result.

Alternative Idea

To make BFF's video searching task faster and more convenient, we put forward an idea of implementing a Smart Video Picker System. This system will allow BFF to simply get the videos to recommend to the users by searching words, phrases or sentences which represent their emotions or situations.

Natural Language Processing (NLP) algorithm is used by this system to train a sentiment analysis model so that it can interpret the BFF's inputs and assign correspond labels to them. Here 'label' means the category a word belongs to, for instance, 'joyous' belongs to 'Happy' label and 'gloomy' belongs to 'Sad' label. After getting those labels, the system will pair the Emotion Before and Emotion After labels to output relevant videos gathered from database. Thus, instead of spending time scanning whole spreadsheet, the result can be obtained in just one click.

Key Technologies

Programming language: Python

Web Framework: Django

NLP Model: FastText

Database: SQLite

System Architecture

The architecture of the system involves 4 main components:

- I) User
- II) User Interface
- III) Smart Video Picker System
- IV) SQLite Database.

The BFFs can easily interact with the system to find the videos and recommend them to our customers while texting.

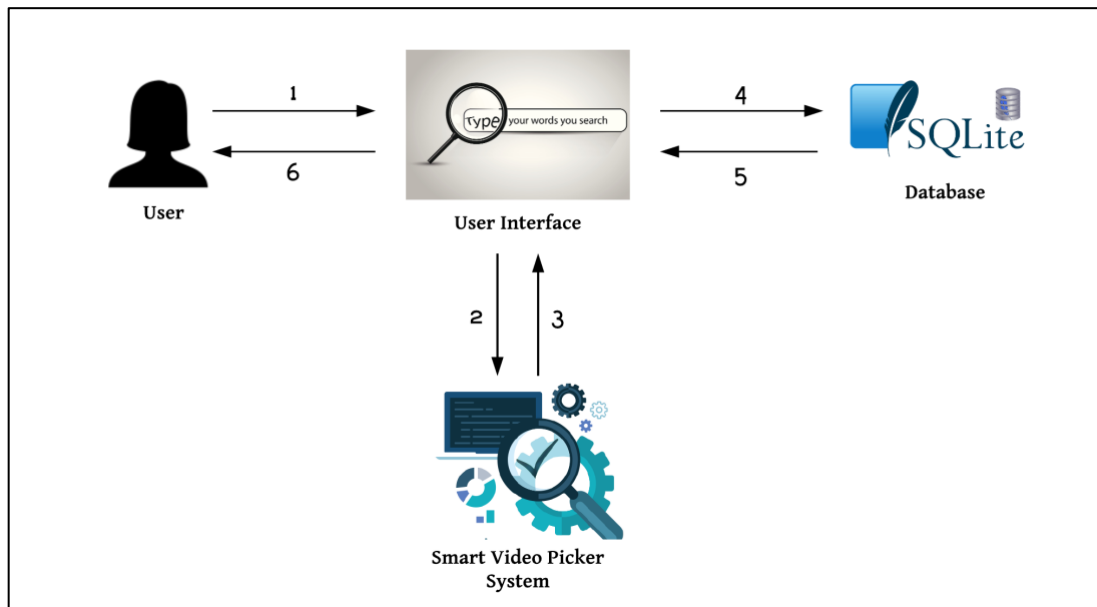


Figure 1. System Architecture

Figure 1 outlines the system, its workflow steps are as follow:

1. While texting, BFFs(user of the system) input the search query which includes Emotion Before and Emotion After conditions.
2. Those two inputs are passed to the Smart Video Picker System, which involves a FastText model trained by customized corpus. It predicts and stores labels of each input
3. A list of video IDs are found based on those labels and then returned using RESTful API¹ in a sorted order of search relevance.
4. The system then maps each video ID with its details from the SQLite database.
5. All relevant information of videos are displayed on the result page.
6. BFFs can pick the most suitable video from the result page and send it to the texter.

¹ RESTful API - **RE**presentational **S**tate **T**ransfer is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

System Detailed Design

We broke down the system design into three parts: front-end design, back-end design and database design.

Front-end Design

Figure 2 represents the general layout of the user interface.

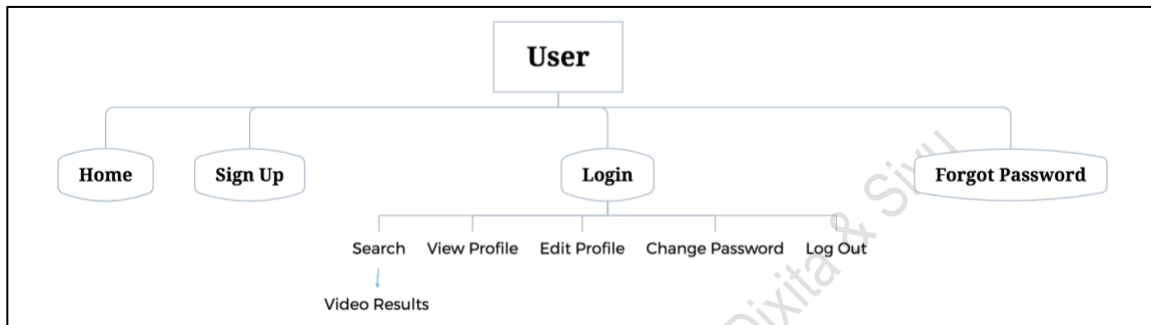


Figure 2. User Interface Layout

When users land into the system, they have 4 options:

- **Home:** It contains the basic introduction about this Smart Video Picker System.
- **Sign Up:** If the users are using the system for the first time, they can create their own profile.
- **Login:** Before searching for videos, users should login first.
- **Forgot Password:** In case users forget their password, they can definitely reset using their email address.

After users successfully log in, they will have 5 more options:

- **Search:** The core part of this system, where users can search for videos based on emotions/situations.
- **View Profile:** Logged-in user can verify their profile details which are pulled out from user database.
- **Edit Profile:** If the users find any mistakes in their profile or want to update their email address, they can easily do so here.
- **Change Password:** The situation in which the users want to change their existing password, they can go with this option.
- **Logout:** The user will be logged out from the system.

When the users input their queries on the search page, they will be shown a list of videos on the next page.

- **Video Results:** This page lists each video in a card form, displaying its Title, Link, Description, Trigger Warning information, Emotion Before and Emotion After labels.

Back-end Design

In every system, the core logic and implementation lies on the back-end. Since Smart Video Picker System involves human emotions as an input, analyzing and assigning each input a label based on its meaning is required.

Labels

As you can see in Figure 3 and Figure 4, we came up with 11 Emotion/Situation Before labels and 7 Emotion/Situation After labels.

B1 [Sad]	B2 [Angry]	B3 [Happy]
B4 [Breakup]	B5 [Bored]	B6 [Confusion]
B7 [Worry]	B8 [Doubt]	B9 [Anxiety]
B10 [Stress]	B11 [Depression]	

Figure 3. Emotion/Situation Before Labels

A1 [Happy]	A2 [Sad]	A3 [Confidence]
A4 [Hope]	A5 [Motivate]	A6 [Relax]
A7 [Thoughtful]		

Figure 4. Emotion/Situation After Labels

Corpus

In order to train a model to analyze user's input data based on its meaning, we need to create our own corpus, which can be called training dataset. It consists of the possible synonyms under each label.

Thus, using these training dataset in the model, the accuracy can be guaranteed.

Algorithm

We use FastText model to train the corpus. FastText is a library created by the Facebook Research Team for efficient learning of word representations and sentence classification [2].

It is an extension to Word2Vec. Instead of feeding individual words into the Neural Network, FastText breaks words into several n-grams (sub-words), where n could range from 1 to the length of the word. For instance, the tri-grams for the word apple is app, ppl, and ple (ignoring the starting and ending of boundaries of words). The word embedding vector for apple will be the sum of all these n-grams. After training the Neural Network, we will have word embeddings for all the n-grams given the training dataset [3]. FastText can be used to obtain vectors for out-of-vocabulary (OOV) words, by summing up vectors for its component char-ngrams, provided at least one of the char-ngrams was present in the training data [4].

So for our model, we tried several combinations of parameters for training, and picked the one with best performance as shown in Figure 5:

```
model=FastText(sentences=labels,min_count=1,sg=1,min_n=2,max_n=6,word_ngrams=1)
```

Figure 5. Model Parameters

The parameters [5] inside FastText can be explained as:

- **sentences** (*iterable of list of str*) - Can be simply a list of lists of tokens. In our case it is the training dataset (corpus).
- **min_count** (*int, optional*) - The model ignores all words with total frequency lower than this.
- **sg** (*{1, 0}, optional*) - Training algorithm: skip-gram if sg=1, otherwise CBOW.
- **min_n** (*int, optional*) - Minimum length of char n-grams to be used for training word representations.
- **max_n** (*int, optional*) - Max length of char ngrams to be used for training word representations. Set max_n to be lesser than min_n to avoid char ngrams being used.
- **word_ngrams** (*{1,0}, optional*) - If 1, uses enriches word vectors with subword(n-grams) information. If 0, this is equivalent to Word2Vec.

```
top_w=model.wv.similar_by_word(word=w,topn=5)
```

Figure 6. Top5 Similar Words

Next, we use **similar_by_word** method [6] to obtain words with high similarities of a given word(input) from training dataset, and then assign a label to it based on similar word's label. As shown in Figure 6.

- **word** (*str*) - Each input word
- **topn** (*{int, False}, optional*) - Number of top-N similar words to return. If topn is False, it returns the vector of similarity scores. In our case, we set topn=5

Once we get labels of the input data, we can connect to the database for extracting necessary information.

Database Design

Two tables are maintained in SQLite database: auth_user and bffsvps_videoinfo.

- auth_user table is to manage user information and give different access control over the database. Its definition can be seen in Figure 7 and the sample data is in Figure 8:

auth_user		CREATE TABLE "auth_user" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT
auth_user	integer	`id` integer NOT NULL PRIMARY KEY AUTOINCREMENT
password	varchar (128)	`password` varchar (128) NOT NULL
last_login	datetime	`last_login` datetime
is_superuser	bool	`is_superuser` bool NOT NULL
username	varchar (150)	`username` varchar (150) NOT NULL UNIQUE
first_name	varchar (30)	`first_name` varchar (30) NOT NULL
email	varchar (254)	`email` varchar (254) NOT NULL
is_staff	bool	`is_staff` bool NOT NULL
is_active	bool	`is_active` bool NOT NULL
date_joined	datetime	`date_joined` datetime NOT NULL
last_name	varchar (150)	`last_name` varchar (150) NOT NULL

Figure 7. auth_user Table

	id	password	last_login	is_superuser	username	first_name	email	is_staff	is_active	date_joined	last_name
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	pbkdf2_sha...	2018-07-20 00...	0	bffs	User1	user1@gmail.com	0	1	2018-06-28 22...	User_Iname
2	2	pbkdf2_sha...	2018-08-02 00...	1	bluefever	Admin	admin@gmail.com	1	1	2018-07-14 20...	Admin_Iname

Figure 8. User Data

As you can see, 'is_staff' attribute is 0 for user named 'bffs' while 1 for 'Admin'. The difference between this two is that the 'Admin' has authority to edit the database while user does not have that power.

- bffsvps_videoinfo table is for storing video details. Its definition is shown in Figure 9 and its sample data is in Figure 10.

bffsvps_videoinfo	CREATE TABLE "bffsvps_videoinfo" (`id` integer NOT NULL
id	integer	`id` integer NOT NULL PRIMARY KEY AUTOINCREMENT
Title	varchar (140)	`Title` varchar (140) NOT NULL
Link	varchar (200)	`Link` varchar (200) NOT NULL
Description	text	`Description` text NOT NULL
Warning	TEXT	`Warning` TEXT NOT NULL
Emotion_Before	TEXT	`Emotion_Before` TEXT NOT NULL
Emotion_After	TEXT	`Emotion_After` TEXT NOT NULL

Figure 9. bffsvps_videoinfo Table

	id	Title	Link	Description	Warning	Emotion_Before	Emotion_After
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Lizzo - Fe...	http://bit.l...	Lizzo sings abo...	No	Angry, Happy, ...	Happy, Confiden...
2	2	Lizzo - Tr...	http://bit.l...	Wild music vide...	Sweari...	Angry, Breaku...	Happy, Confiden...
3	3	Sweetest...	http://bit.l...	Colorful & danci...	No	Happy, Sad, B...	Happy, Relax, H...
4	4	Daily Itine...	http://bit.l...	Impersonations ...	No	Bored, Stress	Happy, Relax
5	5	The Five ...	http://bit.l...	Video focused o...	No	Confusion, Sad...	Happy, Confiden...

Figure 10. Videoinfo Data

After training the model, each input is mapped to **Emotion_Before** and **Emotion_After** labels with its video ids. Then we use this table to display details linked to those ids.

System Implementation

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design [7]. Django follows Model, View and Controller (MVC) architecture which makes the system development fast, secure, scalable and versatile [8].

Smart Video Picker System was implemented on Django and the Figure 11 shows the directory structure of the whole project.

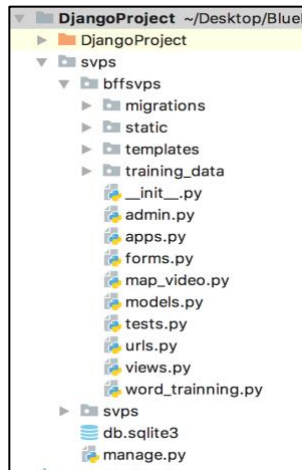


Figure 11. Directory Structure

Files Detail

All these files are properly clustered into different groups by Django.

Project Level

We started a project named svps, Figure 12 shows the files that were created by Django automatically:

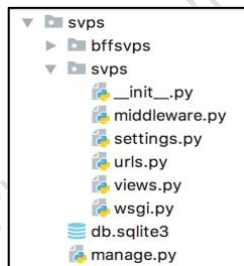


Figure 12. SVPS AutoFiles

The basic information [9] regarding above files are:

- The outer **svps/** root directory is the container for this project.
- **manage.py**: A command-line utility that lets you interact with this Django project in various ways.
- The inner **svps/** directory is the actual Python package for this project. Its name is the one that you'll need to use to import anything inside it (e.g. **svps.urls**).
- **svps/__init__.py**: An empty file that tells Python that this directory should be considered a Python package.
- **svps/settings.py**: Settings/configuration for this Django project
- **svps/urls.py**: The URL declarations for this Django project; a “table of contents” of your Django-powered site.
- **svps/wsgi.py**: An entry-point for WSGI-compatible web servers to serve your project.

Application Level

Inside the project, we created an app named bffsvps, Django generated this basic directory structure and files.

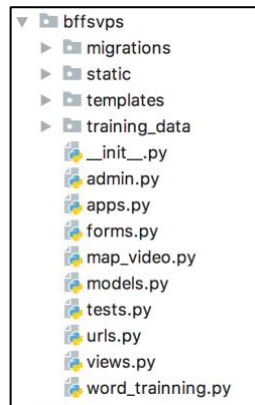


Figure 13. bffsvps Files

Figure 13 shows the files contained in our bffsvps app. They are:

- **__init__.py:** An empty file that tells Python that this directory should be considered a Python package.
- **admin.py:** One of the most powerful parts of Django is the automatic admin interface, which is controlled by admin python file. It reads metadata from your models to provide a quick, model-centric interface where trusted users can manage content on your site [10].
- **apps.py:** Set app's name and other configurations here.
- **forms.py:** Django provides a rich framework to facilitate the creation of forms and the manipulation of form data, which were written in this file.
- **models.py:** Django provides an abstraction layer (the “models”) for structuring and manipulating the data of your Web application [11]. All database and tables are defined here.
- The **migrations/** directory: Every time you create or edit tables in the models.py, migration is mandatory. The migrations folder stores the relevant files which are automatically generated while you make migrations, as shown in Figure 14:

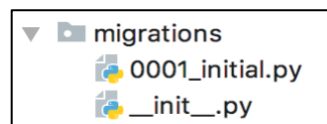


Figure 14. Migration Files

- **tests.py:** It is for debug purpose.
- **urls.py:** To design URLs for an application, we created a Python module called a URLconf and stored here. Like a table of contents for the app, it contains a simple mapping between URL patterns and views [11].

- **views.py:** Django has the concept of “views” to encapsulate the logic responsible for processing a user’s request and for returning the response [11].
- **word_training.py:** A python file where we generated a FastText model and trained our corpus to get the labels of each input.
- **map_video.py:** We mapped labels with video ids from database, and returned details of each selected video here.
- **templates/ directory:** The template layer provides a designer-friendly syntax for rendering the information to be presented to the user [11]. All HTML templates that were designed by us are shown in Figure 15:

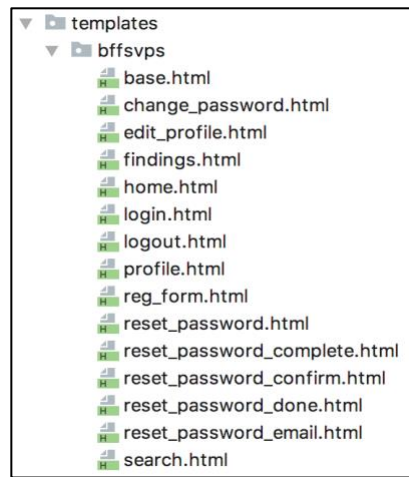


Figure 15. HTML Files

- **static/ directory:** We used Bootstrap templates, which are stored in static folder (Figure 16).



Figure 16. Bootstrap Templates

- **training_data/ directory:** The dataset we used for training the model can be found under training_data directory. (Figure 17)

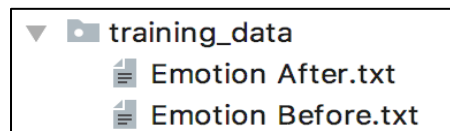


Figure 17. Training Data

Running Server

To run application locally, we need to run the server first using below command:

python manage.py runserver

Application can be launched on the browser via <http://127.0.0.1:8000/> Figure 18 and 19 show the status of the server.

```
Terminal
+ (DjangoProject) Siyus-MacBook-Pro:DjangoProject siyu_xie$ cd svps
x (DjangoProject) Siyus-MacBook-Pro:svps siyu_xie$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
August 02, 2018 - 18:14:16
Django version 2.0.6, using settings 'svps.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figure 18. Run Server

```
[02/Aug/2018 18:14:31] "GET / HTTP/1.1" 302 0
[02/Aug/2018 18:14:31] "GET /bffsvps/login/ HTTP/1.1" 302 0
[02/Aug/2018 18:14:31] "GET /bffsvps/search/ HTTP/1.1" 200 3217
[02/Aug/2018 18:14:31] "GET /static/img/bluefever.png HTTP/1.1" 304 0
[02/Aug/2018 18:15:28] "GET /bffsvps/logout/ HTTP/1.1" 200 2501
[02/Aug/2018 18:15:34] "GET /bffsvps/home/ HTTP/1.1" 200 2334
[02/Aug/2018 18:16:28] "GET /bffsvps/register/ HTTP/1.1" 200 3781
[02/Aug/2018 18:19:17] "POST /bffsvps/register/ HTTP/1.1" 200 3922
[02/Aug/2018 18:19:55] "POST /bffsvps/register/ HTTP/1.1" 302 0
[02/Aug/2018 18:19:55] "GET /bffsvps/login/ HTTP/1.1" 200 2805
[02/Aug/2018 18:20:26] "GET / HTTP/1.1" 302 0
```

Figure 19. Server is Running

Base Template

The 'base.html' file has the logic for header and footer. This file is extended on each HTML page of User Interface using Jinja2. See Figure 20.

```
base.html {% extends "bffsvps/base.html" %}
```

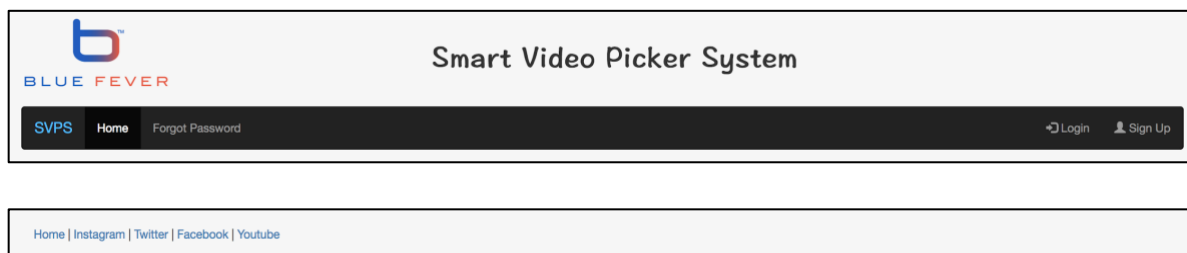


Figure 20. Header & Footer

Home Page

As soon as you land into the system, you will be able to see the homepage with a simple introduction regarding this system. 'Home.html' file has the logic behind it. See Figure 21.

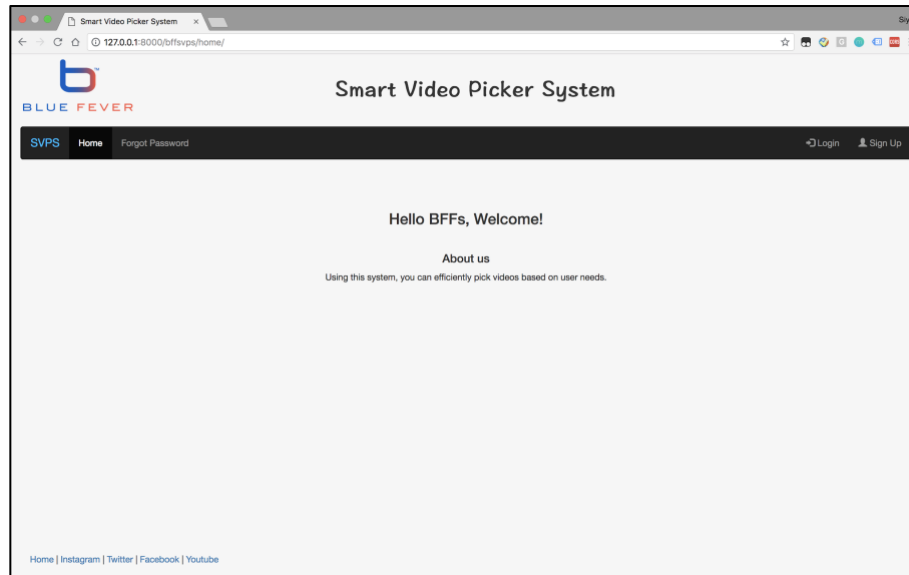
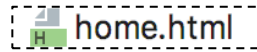


Figure 21. Home Page

Sign Up

Users need to create their profile before using the system. 'reg_form.html' file has the form handling code to link database. All these data are stored in auth_user table. See Figure 22.

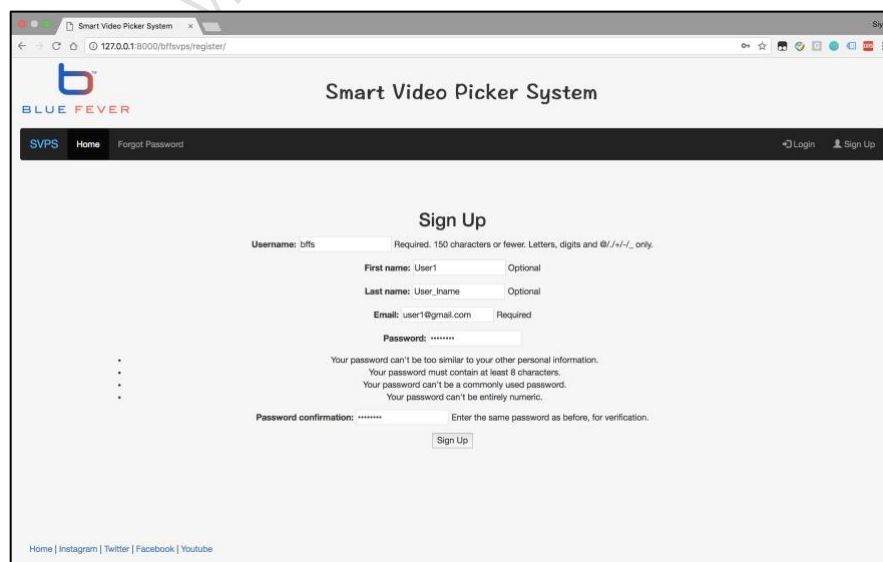
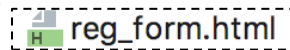


Figure 22. Sign Up

Login & Logout

Once users have their credentials, they can log into the system and start searching for videos. When they logs out, they are back to the homepage. See Figure 23 and Figure 24.

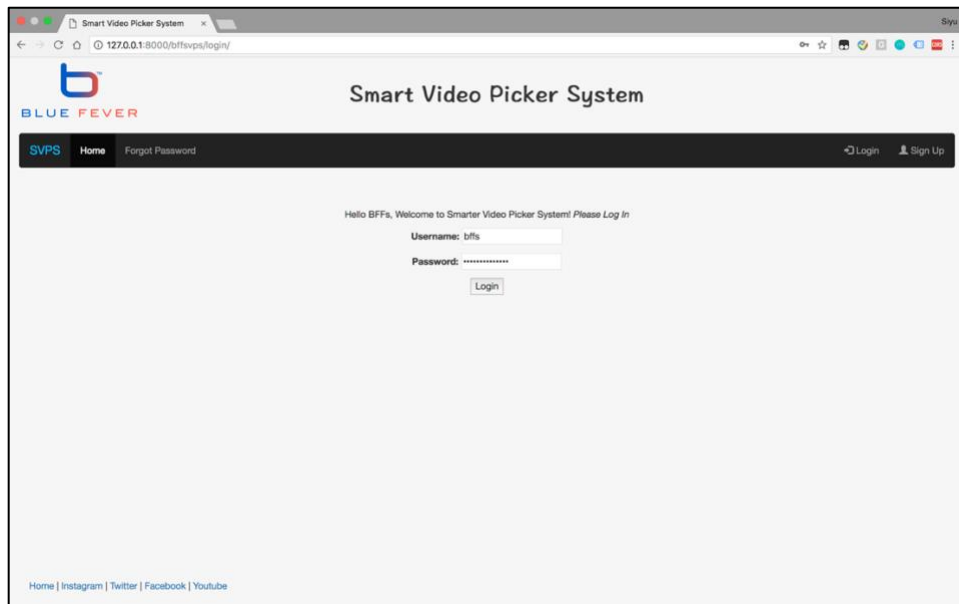
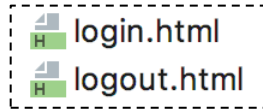


Figure 23. Login

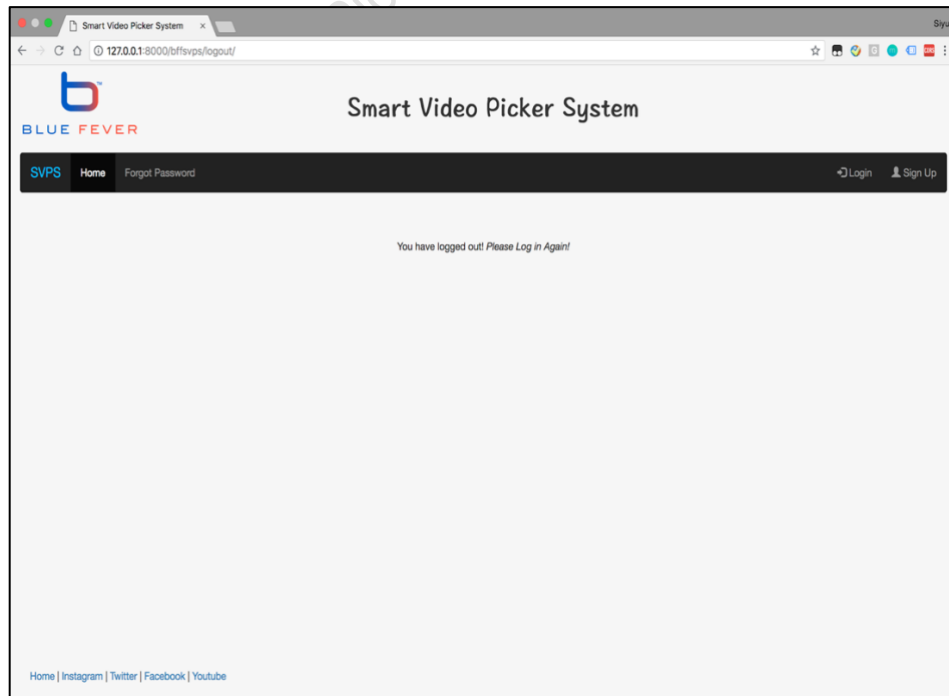


Figure 24. Logout

Forgot Password

When an existing user is returning to the system but forgot the password, he/she can use this option. See Figure 25.

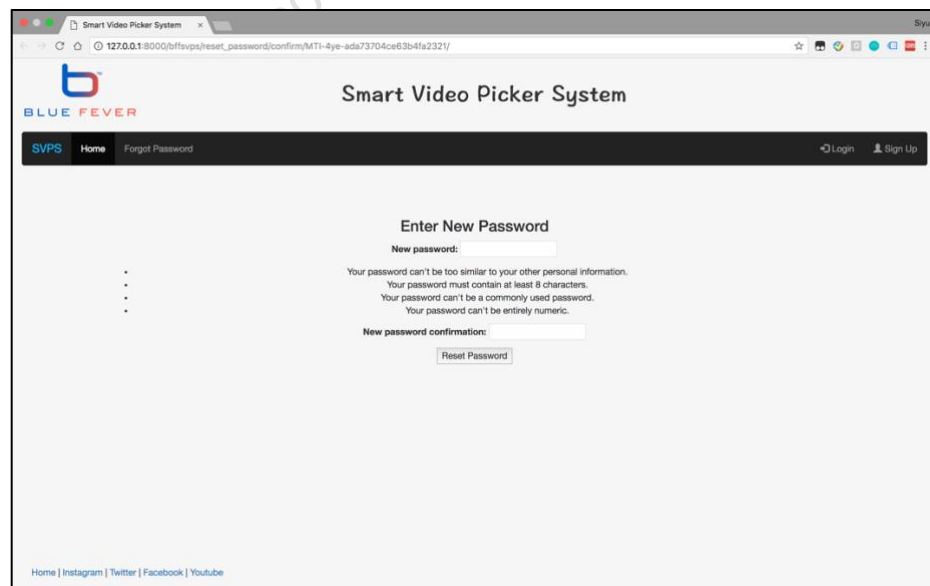
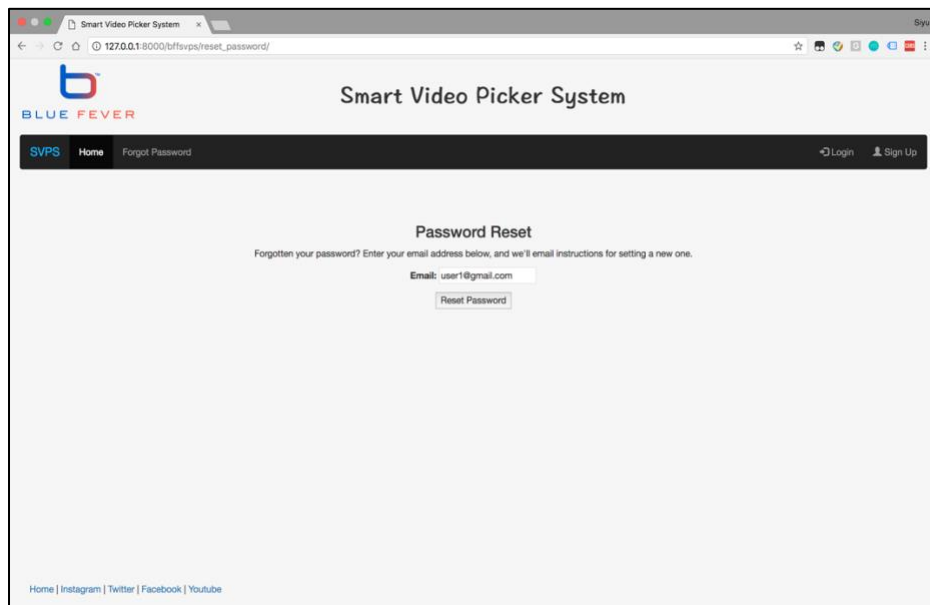
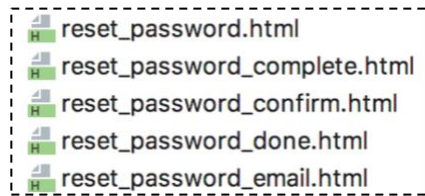



Figure 25. Forgot Password

Profile

When a user logs into the system, he/she can have three options regarding his/her profile: view, edit or change password. 'profile.html' is used to pull users data from database (Figure 26), 'edit_profile.html' pulls data and allows the user to make changes (Figure 27) and 'change_password.html' has the logic to change password (Figure 28).

 profile.html

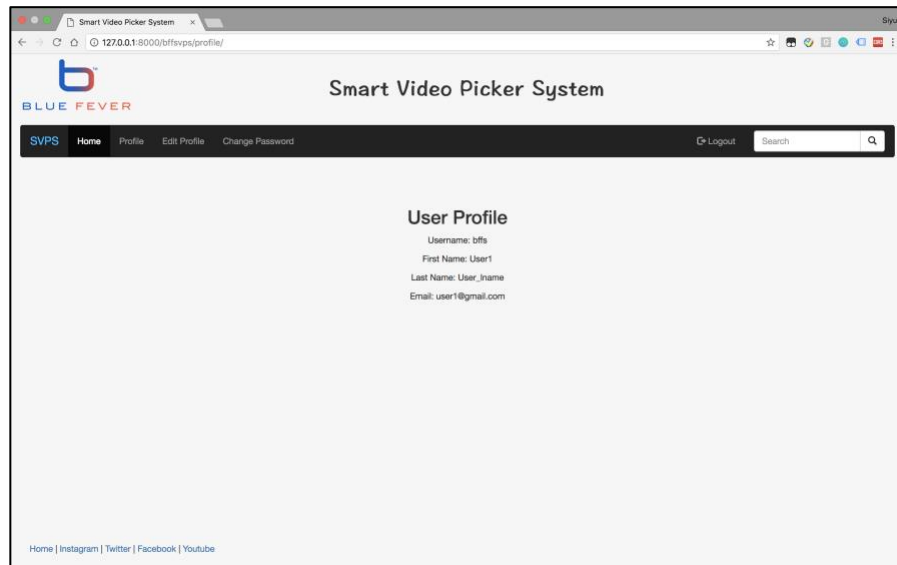



Figure 26. View Profile

 edit_profile.html

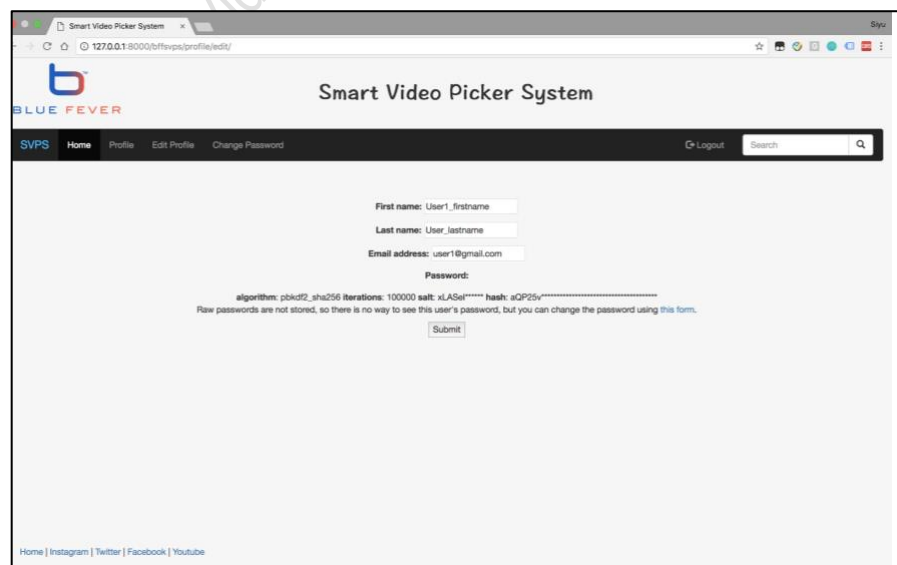


Figure 27. Edit Profile

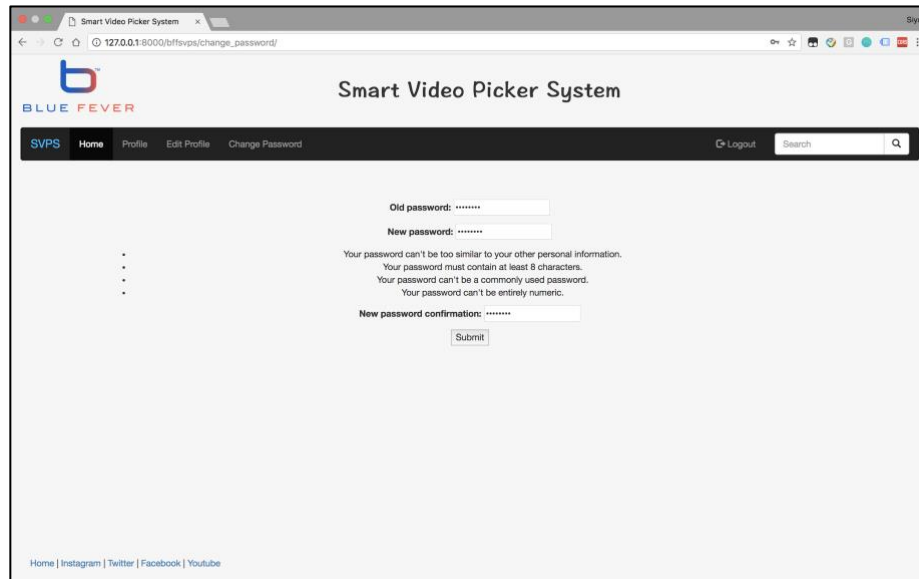
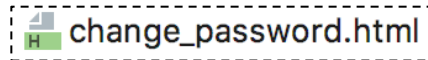


Figure 28. Change Password

Search & Results

The core part of the system is this search page. As can be seen in the Figure 29 below, there are two input box named 'Emotion Before' and 'Emotion After' to take the user's queries. When users input their queries and hit 'Find Video' button, it redirects the search page to the result page('findings.html'), where the video detailed list is displayed.

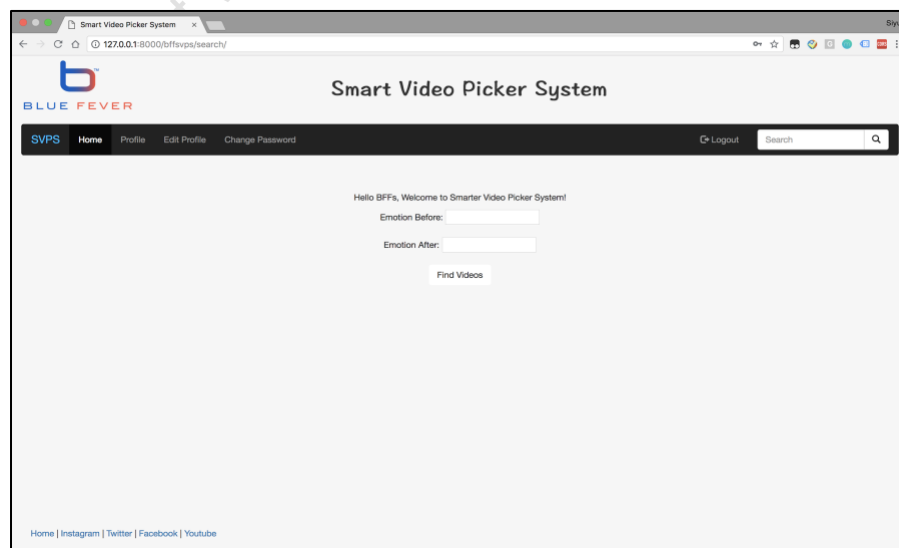
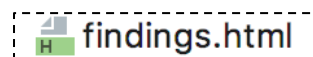
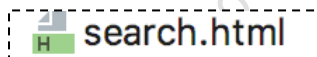


Figure 29. Search Page

Smart Video Picker System

Below we present 2 sample cases which shows two possible results while searching.

- Sample Case #1:

When BFF's input Emotion/Situation Before and After, if we have the video which satisfies both input, then we will output the exact matching videos, pointing out the top count of the search results.

See the sample input & output on Figure 30.

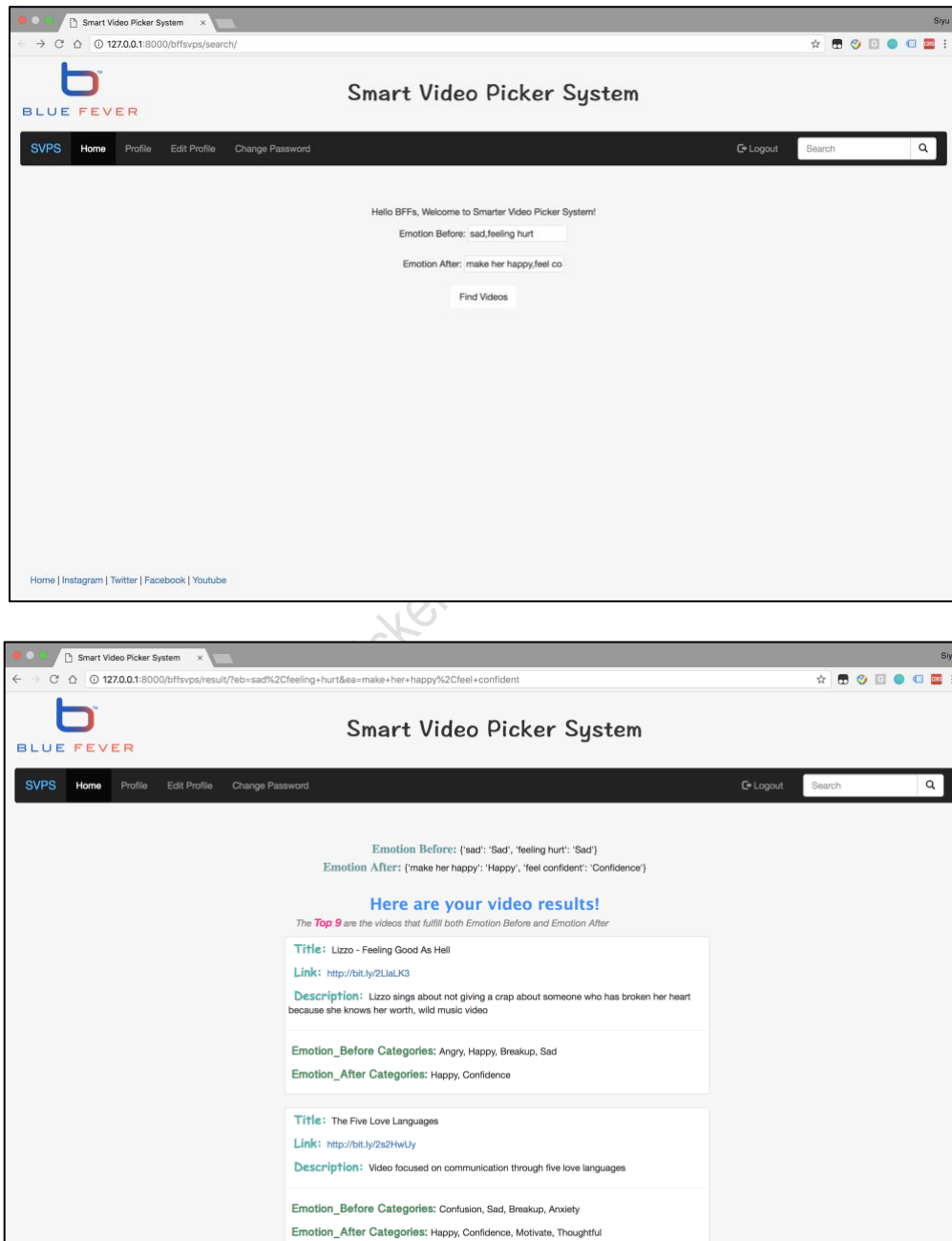


Figure 30. Sample Case #1

Smart Video Picker System

- Search Sample #2:

If we don't have any video that fulfills both Before and After input from BFFs (the user), we will output partial matching results where videos that satisfy Emotion/Situation After come first.

See the sample input & output on Figure 31.

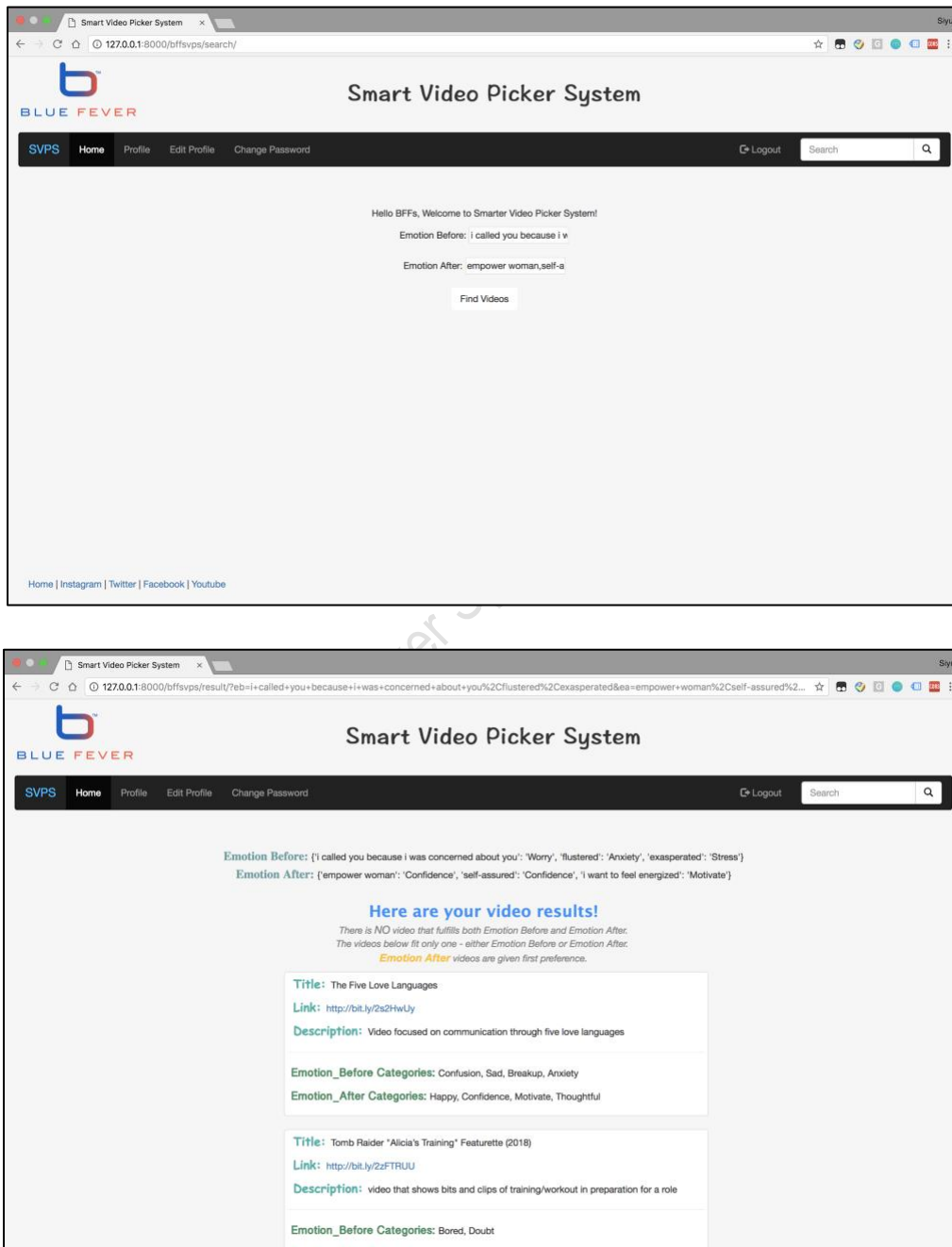


Figure 31. Sample Case #2

Validation and Testing

The most important task before deploying a system is to validate the algorithm's accuracy and to perform User Acceptance Test (UAT).

Validation

We validated Smart Video Picker System by :

Step 1: Modifying the corpus - using stemming and lemmatization to make word consistent.

Step 2: Considering different conditions - setting appropriate threshold values to decide the label of each input as accurate as possible.

Table 1 shows the record of the accuracy that we obtained from each step. The accuracy is defined as equation (1).

Table 1. Validation Accuracy

Validation	Base	Step 1	Step 2
Accuracy	49.25%	64.18%	90.01%

$$Accuracy = \frac{\# \text{ correct labels of input data}}{\# \text{ total labels of input}} \quad (1)$$

Testing

User acceptance testing (UAT) is the last phase of the software testing process. During UAT, actual software users test the software to make sure it can handle required tasks in real-world scenarios, according to specifications.

In our case, BFFs are the actual users who tested the system. The testing helped to check how well the system behaved and get feedback for improvement.

The testing phase is ongoing.

Deployment

The easiest and best way to deploy Django project is using Heroku. Heroku is a cloud application platform, it is basically a Platform-as-a-Service (PaaS). It supports several programming languages, including Python.

Once the testing phase is completed, the system will be deployed.

Conclusion

The implementation of Smart Video Picker System originated from the need of selecting video faster. It eased the video searching process based on the situation and emotion of the user. The system involved human emotions which requires Natural Language Processing to make appropriate decision.

To accomplish this goal, we first collected all possible Emotion/Situation Before and Emotion/Situation After labels with the help of BFFs. Then we used FastText, a model that can process the input of words, phrases or sentences to predict their corresponding labels. Finally we output video details from database by mapping their ids to the found labels.

Django web framework handles security issues, built-in admin access, stable release and well organized document for each step. By using it, we developed simple and interactive User Interface with all the required functionalities.

While building the system, we trained the algorithm separately to ensure the acceptable accuracy and efficiency. Afterwards, we started integrating it with the User Interface using Django, but since Django was new for us, we followed a useful video tutorial [12] to learn its basic structure and logic. We ran into some troubles when we attempted to extract and push inputs from the search page to our algorithm. After trying various methods, it was the RESTful API that gave the desired results and helped us achieve the task.

In version 1.0, we only considered the most occurring labels from the conversation with our customers. There are still scopes for future development of this system, which will be implemented in version 2.0 or higher.

Future Work

This report describes Version 1.0 of the Smart Video Picker System. There are many improvement scopes on both front-end and back-end side in the future.

Front-end Future Improvement

- Add more search conditions/options, e.g. reasons of current emotions, topics, genres, etc.
- Create a copy button for copying a video link more easily.
- For user's (BFFs) convenience, add template message of each video and display it on the result page.

Back-end Future Improvement

- The keywords linked with each video can be added into the database so that when we get the list of possible video as the results, only the videos that have those keywords can be displayed.

- Narrow down current labels to sublabels as well as collecting more possible labels.
- Add training data into our corpus based on new labels to further improve accuracy of the video search results.

Database Future Improvement

- As the number of videos increases, we can move from SQLite database to larger database such as MySQL or MongoDB to handle traffic efficiently.

Smart Video Picker System by Dixita & Siyu

References

1. Blue Fever, Inc [Online]. Available: <https://www.bluefever.com/>
2. NSS. (2017, July). Text Classification & Word Representations using FastText (An NLP library by Facebook) [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/07/word-representations-text-classification-using-fasttext-nlp-facebook/>
3. Steeve Huang. (2018, February). Word2Vec and FastText Word Embedding with Gensim [Online]. Available: <https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c>
4. Wode.AI. Using FastText via Gensim [Online]. Available: http://dev.wode.ai/hub/repo/gensim/docs/notebooks/FastText_Tutorial.ipynb
5. Gensim. models.fasttext – FastText model [Online]. Available: <https://radimrehurek.com/gensim/models/fasttext.html>
6. Gensim. models.keyedvectors – Store and query word vectors[Online]. Available: https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedvectors.WordEmbeddingsKeyedVectors.similar_by_word
7. Django. Django makes it easier to build better Web apps more quickly and with less code [Online]. Available: <https://www.djangoproject.com/>
8. The django Book. The Model-View-Controller Design Pattern [Online]. Available: <https://djangobook.com/model-view-controller-design-pattern/>
9. Django Documentation. Writing your first Django app, part 1[Online]. Available: <https://docs.djangoproject.com/en/2.1/intro/tutorial01/>
10. Django Documentation. Intro to Django [Online]. Available: <https://www.djangoproject.com/start/>
11. Django. Django documentation [Online]. Available: <https://docs.djangoproject.com/en/2.0/>
12. Max Goodridge. (2016, November) Introduction to Django Web Framework for Python [Online]. Available: <https://www.youtube.com/watch?v=zPVLrvpzOOU&list=PLw02n0FEB3E3VSHjyYMcFadtQORv1lSsj&index=1>