INFO7374 Algorithmic Digital Marketing

Summary	In this assignment, we implemented all the three implementations. And create two respective apps to fulfill the two modes.
Author	Yifu Liu and Pengfei He
App URL	https://limitless-river-04051.herokuapp.com/

Generate Data

Faker:

After looked at the sample dataset from the algorithm, we used Faker to generate fake data with the same pattern, but for the snacks.

Properties for snacks, which users will use for recommendation:
'Authentic', 'Japanese', 'beautifully', 'tasty', 'organic', 'gluten-free', 'GMO-free', 'all-natural', 'artificial-ingredient-free', 'classic', 'trendy', 'healthy'

Branches for snacks, which users will be recommended:
'Snakku','Love With Food','Candy Club''NatureBox','SnackNation', 'ZenPop'Yummy Bazaar World Sampler','FitSnack','Bokksu','MunchPak','Universal Yums', 'Vegan Cuts Snack Box','TokyoTreat','Try the World Snacks'

fake_data_ncf.py — assignment3

fake_data_ncf.py

FakeData_Generator > fake_data_ncf.py > create_csv_file

```python
def create_csv_file():
    with open('../Fake_Data/snack_ncf_fake.csv', 'w', newline='') as csvfile:
        fieldnames = ['userID', 'itemID', 'rating', 'timestamp', 'products']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        products = (
            'Snakku',
            'Love With Food',
            'Candy Club',
            'NatureBox',
            'SnackNation',
            'ZenPop',
            'Yummy Bazaar World Sampler',
            'FitSnack',
            'Bokksu',
            'MunchPak',
            'Universal Yums',
            'Vegan Cuts Snack Box',
            'TokyoTreat',
            'Try the World Snacks'
        )

        writer.writeheader()
        for i in range(RECORD_COUNT):
            itemIndex = fake.random_int(min=0, max=13)
            writer.writerow(
                {
                    'userID': fake.random_int(min=1, max=100),
                    'itemID': itemIndex,
                    'rating': fake.random_int(min=1, max=10),
                    'timestamp': fake.date_time_between(start_date='-1y', end_date='now', tzinfo=None),
                    'products': products[itemIndex],
                }
            )
```

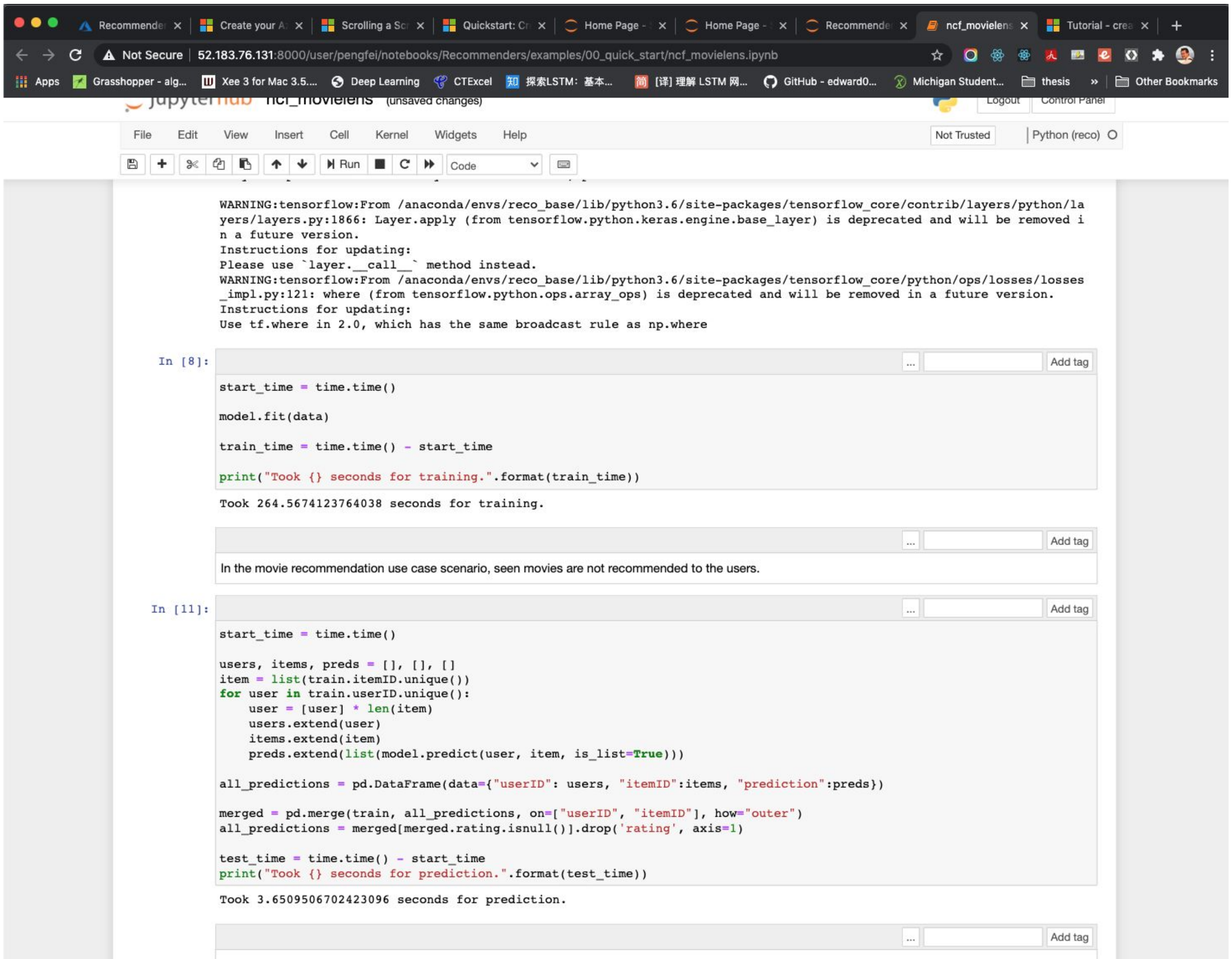
```python
def create_train_file():
    with open('../Fake_Data/snack_npa_train.csv', 'w', newline='') as csvfile:
        # in order to use the same Embedding binary data used by CNN model, we set the
        # fieldname to still be 'candidateNews' and 'clickedNews'.Even through it is better
        # to be named as 'candiateProperty' and 'clickedProperty' for snacks.
        fieldnames = ['id', 'ImpressionID',
                      'User', 'CandidateNews', 'ClickedNews']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        writer.writeheader()
        for i in range(RECORD_COUNT_TRAIN):
            candidateNews = []
            clickedNews = []
            for j in range(CANDIDATE_SIZE):
                candidateNews.append(fake.random_int(
                    min=1, max=(len(propertyMapping) - 1)))
                clickedNews.append(fake.random_int(
                    min=1, max=(len(propertyMapping) - 1)))
            writer.writerow(
                {
                    'id': i,
                    'ImpressionID': fake.random_int(min=1, max=100),
                    'User': fake.random_int(min=1, max=1000),
                    'CandidateNews': candidateNews,
                    'ClickedNews': clickedNews,
                }
            )
```

Apply algorithms on Azure DSVM

We implemented both algorithms on Azure DSVM for better performance and easier configuration.

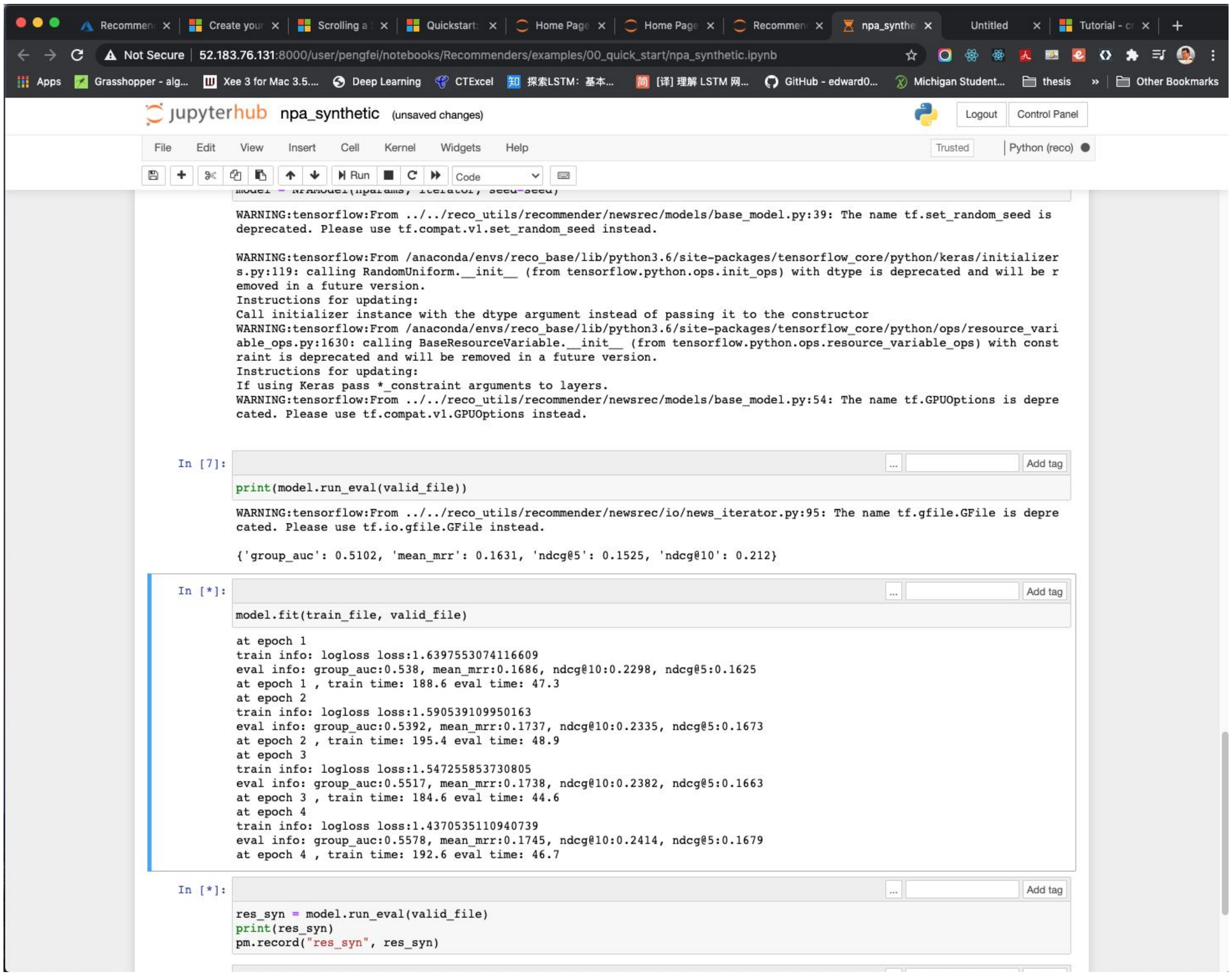
NCF:

The example usage of NCF is for movies, which can be easily applied to snacks, which can both be judged by rating. With the data of the same pattern, NCF algorithm will provide the predicted rating of the snacks. Thus, for our recommendation system, we can provide the user with the top predicted snacks.



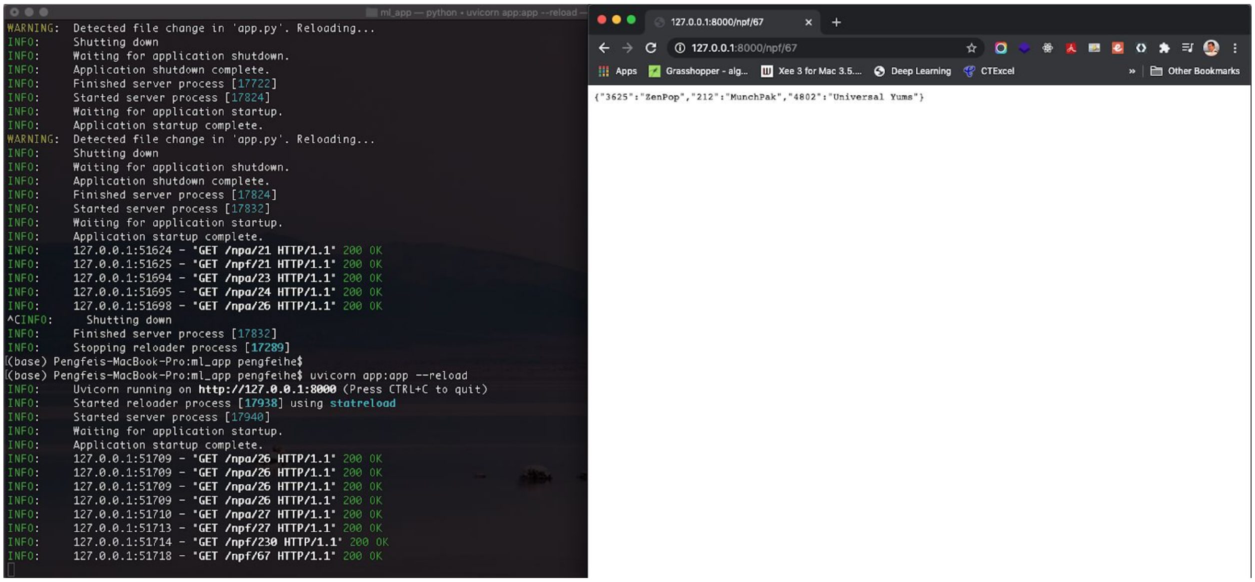
NPA:

The NPA algorithm is a little tricky here. This algorithm is originally used for recommendation News titles, based on the News the user used to click, which seems not that related with snacks. But we find out that we can treat the properties of the snacks as the “News title” for that snack. For example, if the user take a lot of properties like 'organic' and ‘healthy’, then the recommended “News title” will be like “all-nature”. In this way, we won’t need to change the algorithm too much.



Fast API

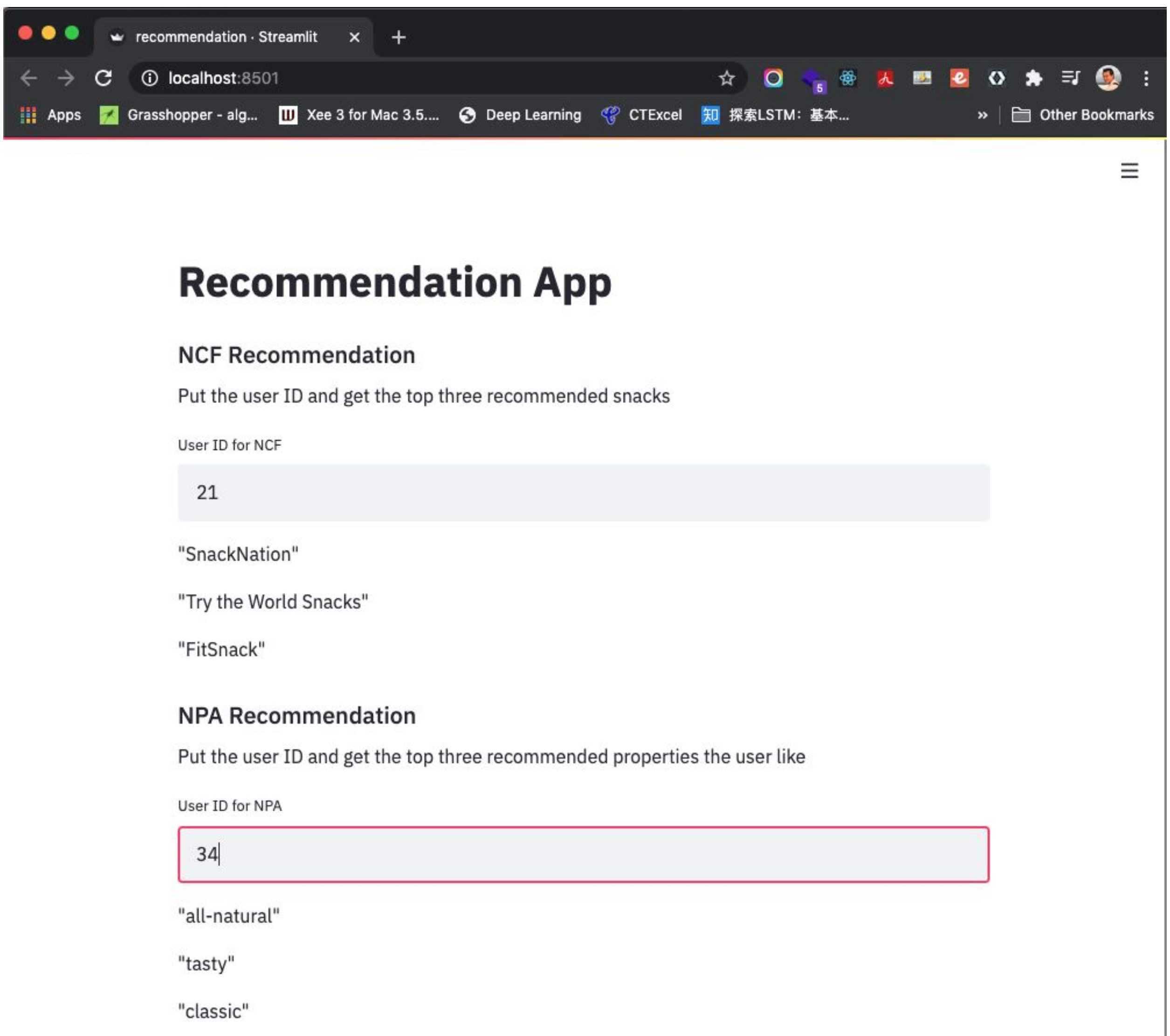
We provided two ‘get’ methods to get the recommended snack brand or snack properties for a certain userID.




```python
@app.get('/ncf/{userID}')
async def get_npf(userID):
    df = npf.loc[npf['userID'] == int(userID)]
    df_sorted = df.sort_values('rating', ascending=False)
    df_head = df_sorted.head(3)
    df_product = df_head['products'].drop_duplicates()
    return df_product


@app.get('/npa/{userID}')
async def get_npa(userID):
    df = npa.loc[npa['User'] == int(userID)]
    df_sorted = df.sort_values('ImpressionID', ascending=False)
    encoded = df_sorted['CandidateNews'].values[0][1:-1].split(',')
    decode = []
    for e in encoded:
        index = int(e.strip())
        decode.append(propertyMapping[index])
    return decode

if __name__ == '__main__':
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

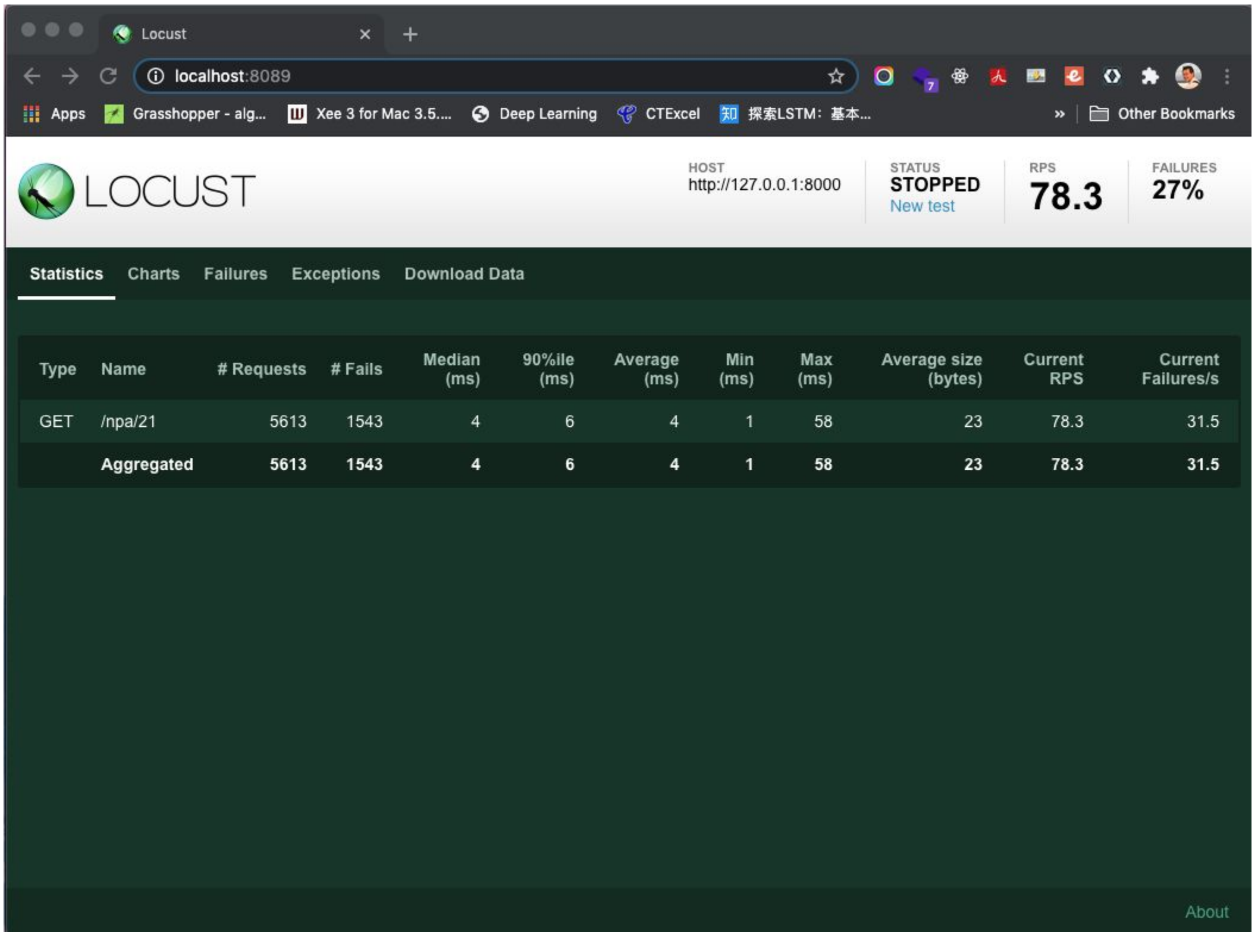
Steamlit App

The UI is built by streamlit and deployed to Heroku:https://limitless-river-04051.herokuapp.com/
Users can:
1 Input a user ID and get the recommended snacks.
2 Input a userID and get the properties of snacks the user like

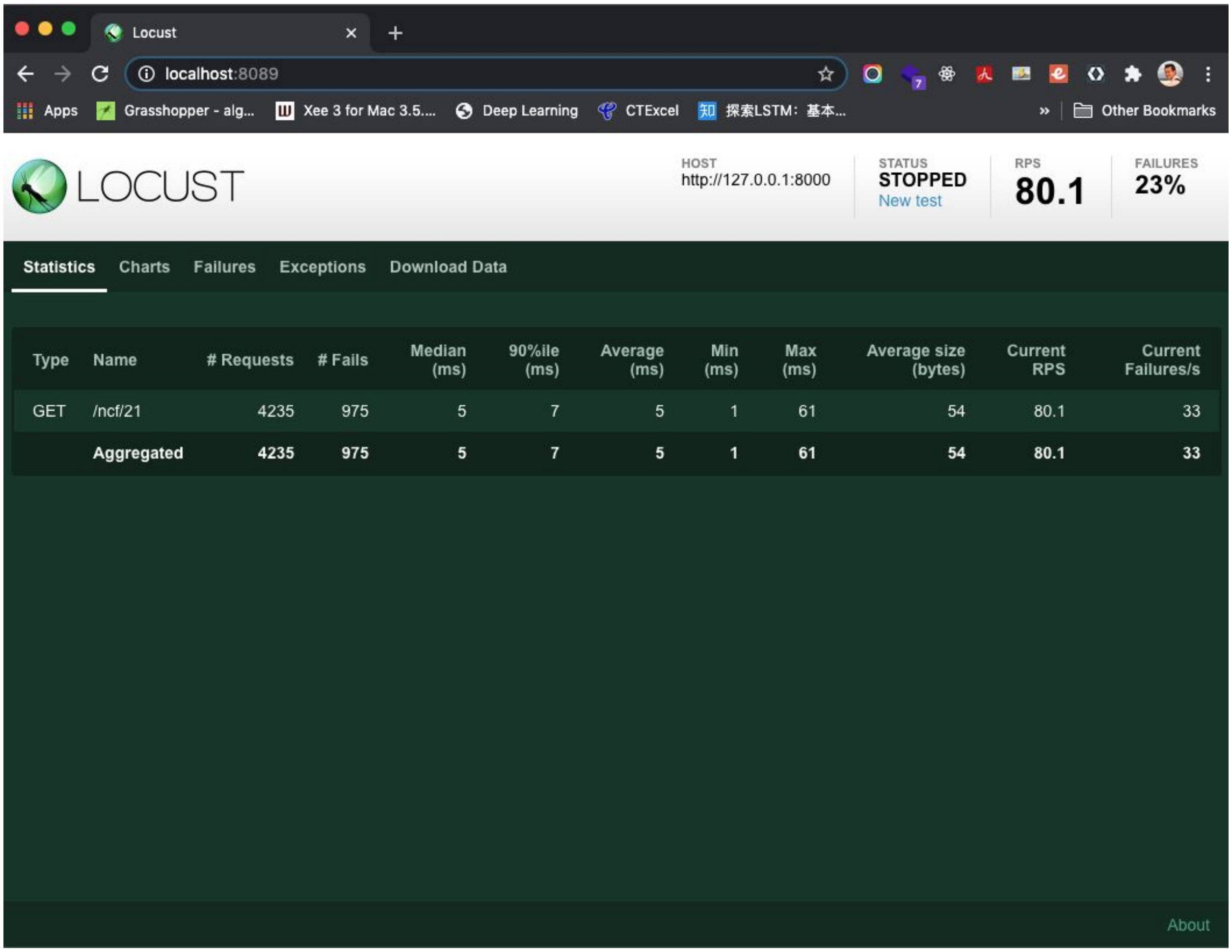


Load Testing using Locust

We used locust for the load testing. We tested the two FastAPI on the local machine. They are both tested on 800 users, 10 user / second, each user will wait 5-15s. The result is as follows:



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/npa/21	5613	1543	4	6	4	1	58	23	78.3	31.5
	Aggregated	**5613**	**1543**	**4**	**6**	**4**	**1**	**58**	**23**	**78.3**	**31.5**



We can find out that they perform almost the same, but NAP has a higher failure rate. Which is to our expectation. Because not like NCF, which only provides the rating, NAP will provide the recommended properties, which is more complex.