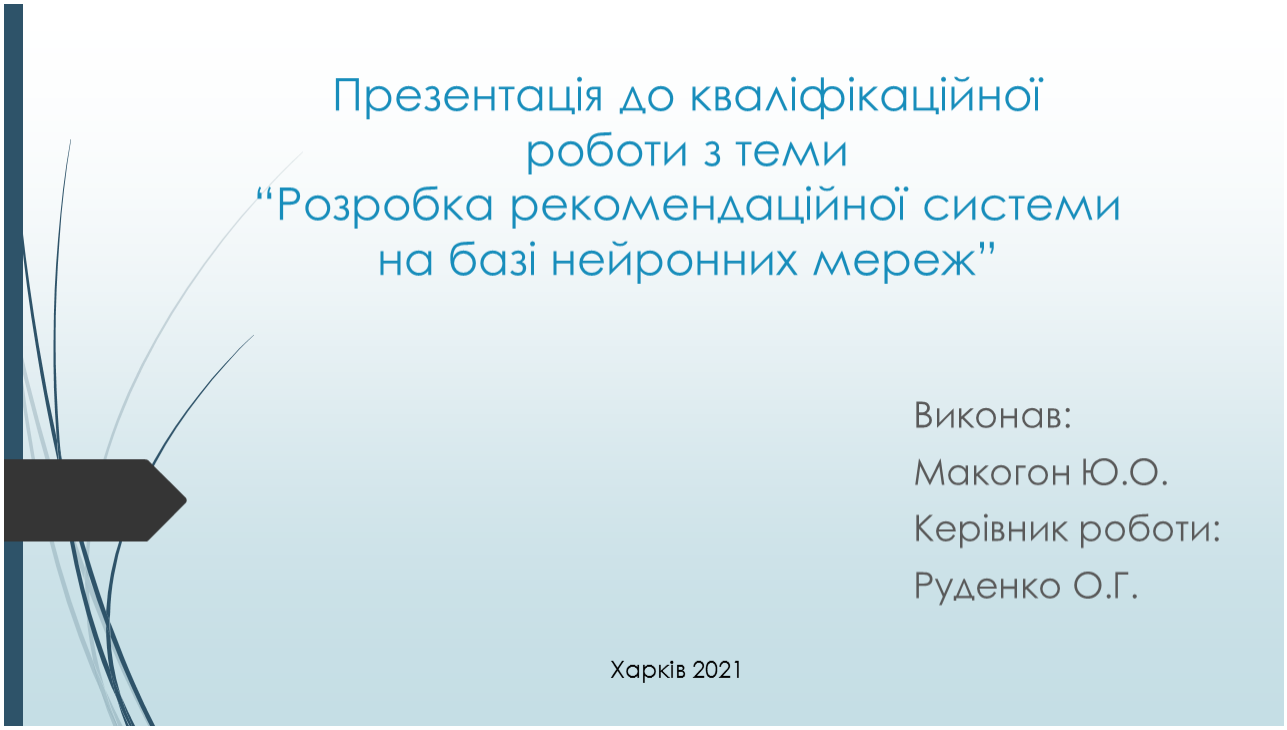
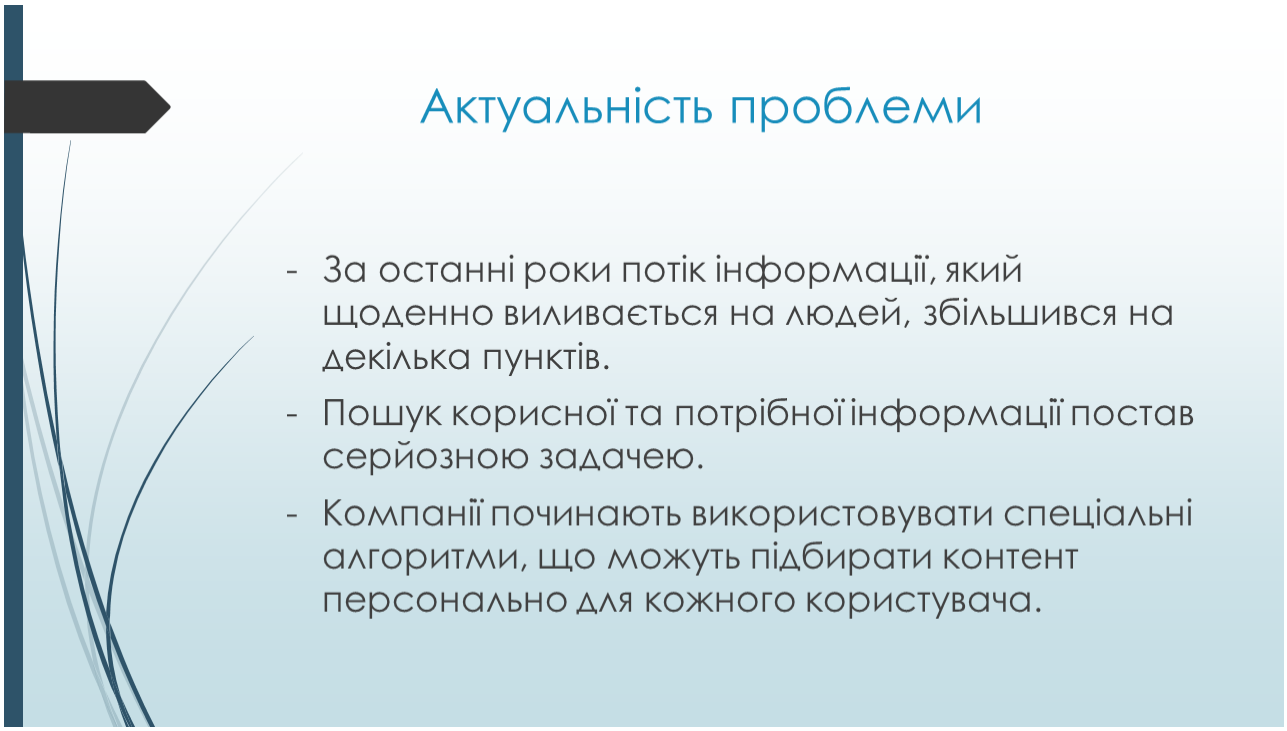
ДОДАТОК А. ГРАФІЧНА ЧАСТИНА





Постановка задачі

Створення онлайн-сервісу для оцінювання та обговорення кінофільмів, який використовує систему рекомендацій.

- Реєстрація та авторизація.
- Клієнтська частина у форматі SPA.
- Можливість оцінювання і коментування кінострічок.
- Алгоритм рекомендацій фільмів.

Використовувані технології

Сервер:

- Django
- Python
- Django rest framework

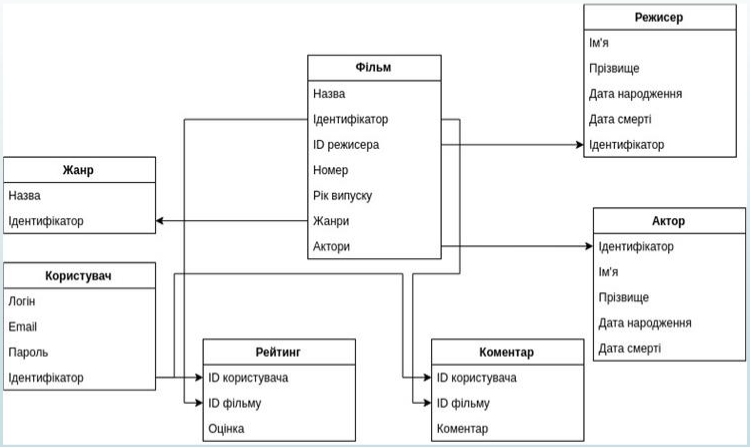
Клієнт:

- React.js
- Redux.js
- Immutable.js
- Redux-saga

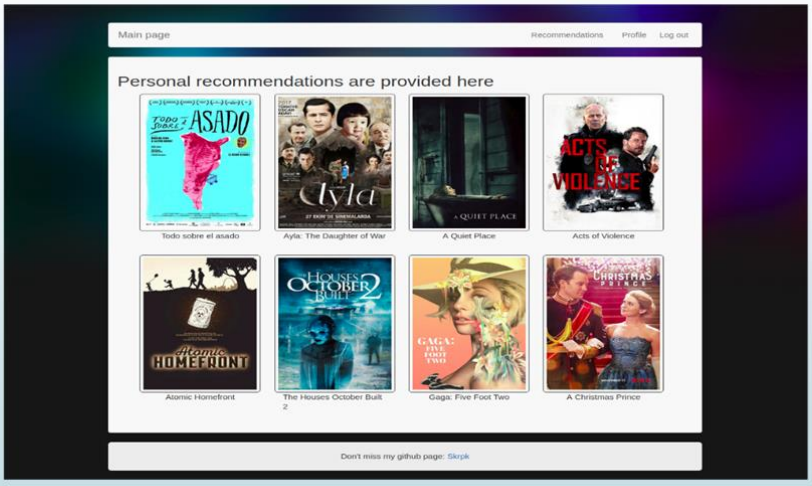
Алгоритм рекомендацій:

- TensorFlow
- NumPy
- Pandas

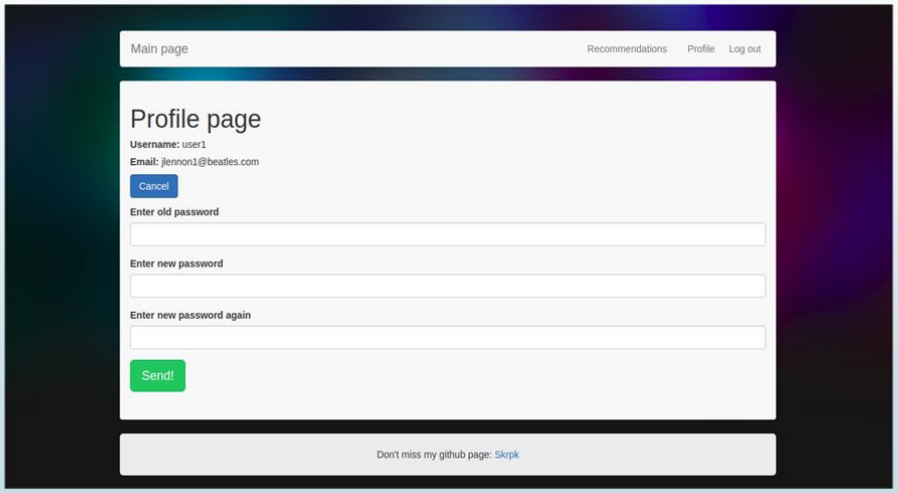
База даних



Інтерфейс програми



Вікно користувача



Висновки

У даній роботі були вивчені основні методи і алгоритми рекомендаційної системи. На їх основі реалізований прототип веб-сайту для побудови профілю та вибору кінофільму.

В ході даної роботи були вирішені наступні завдання:

- Досліджено способи и алгоритми рекомендаційних систем, їх недоліки та особливості;
- Досліджено особливості використання нейронних мереж;
- Розроблено алгоритм рекомендацій;
- Розроблено веб-сайт для використання алгоритму;
- Здобуто навики використання Django, Python, React.js, TensorFlow.

ДОДАТОК Б. КОД ПРОГРАМИ

Б.1 settings.py. Конфігурація серверу

```
INSTALLED_APPS = [
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'rest_framework.authtoken',
  'django.contrib.staticfiles',
  'api.apps.ApiConfig',
  'catalog.apps.CatalogConfig'
]
MIDDLEWARE = [
  'django.middleware.security.SecurityMiddleware',
  'django.contrib.sessions.middleware.SessionMiddleware',
  'django.middleware.common.CommonMiddleware',
  'django.middleware.csrf.CsrfViewMiddleware',
  'django.contrib.auth.middleware.AuthenticationMiddleware',
  'django.contrib.messages.middleware.MessageMiddleware',
  'django.middleware.clickjacking.XFrameOptionsMiddleware',
  'diploma.middleware.dev_cors_middleware'
]
DATABASES = {
  'default': {
      'ENGINE': 'django.db.backends.postgresql_psycopg2',
      'NAME': 'kinoman',
      'USER': 'kinomanuser',
      'PASSWORD': 'avecezar1996',
      'HOST': 'localhost',
      'PORT': '',
  }
}
```

Б.2 urls.py. Головний роутинг програми

```
urlpatterns = [
  path('admin/', admin.site.urls),
  path('api/', include('api.urls')),
] + static(settings.STATIC_URL,
document_root=settings.STATIC_ROOT)
```

Б.3 Моделі для бази даних

```
     class Genre(models.Model):
       name = models.CharField(max_length=200, help_text="Enter a
film genre (e.g. Science Fiction, Drama etc.)")
       def __str__(self):
           return self.name
     class Film(models.Model):
       title = models.CharField(max_length=200)
       index=models.IntegerField(null=True)
       movie_id = models.CharField(max_length=8, unique=True,
primary_key=True)
       year = models.IntegerField(null=True)
       genres = models.ManyToManyField(Genre)
       def __str__(self):
           return self.title
       def get_absolute_url(self):
           return reverse('book-detail',
args=[str(self.movie_id)])
     class Director(models.Model):
       first_name = models.CharField(max_length=100)
       last_name = models.CharField(max_length=100)
       date_of_birth = models.DateField(null=True, blank=True)
       date_of_death = models.DateField('Died', null=True,
blank=True)
       def get_absolute_url(self):
           return reverse('director-detail', args=[str(self.id)])
     class Rating(models.Model):
       user_id = models.CharField(max_length=16)
       movie_id = models.CharField(max_length=8)
       rating = models.DecimalField(decimal_places=2,
max_digits=4)
       rating_timestamp = models.DateTimeField()
       type = models.CharField(max_length=8, default='explicit')
```

Б.4 Контроллер для видачі рекомендацій

```
     import pickle
     model = pickle.load(open("model.p", "rb"))
     class getRecommedations(generics.ListAPIView):
       serializer_class = FilmPreviewSerializer
       permission_classes = (permissions.IsAuthenticated,)
       def list(self, request, id):
           films_count = Film.objects.count()
           films = Film.objects.all()
           recommended_films_indexes_array = model.predict(id,
np.arange(films_count))
           result =
Film.objects.filter(index__in=np.argsort(recommended_films_indexes
_array)[:8])
           serializer = self.get_serializer(result, many=True)
```

```
            return Response(serializer.data)
        def get(self, request, *args, **kwargs):
            id = kwargs['id'].split(sep="=")[1]
            return self.list(request, id)
```

Б.5 Розробка алгоритму рекомендацій

```
import numpy as np
import pandas as pd
import tensorflow as tf
     ratings = list(Rating.objects.all())
       films = list(Film.objects.all())
       users = list(User.objects.all())
       interactions_matrix = []
       # a = Rating.objects.filter(user_id=1, movie_id='0068646')
       # print(int(a[0].rating))
       for user in users:
           rating_of_user = []
           for film in films:
               rating = Rating.objects.filter(user_id=user.id,
movie_id=film.movie_id)
               if rating:
                   rating_of_user.append(int(rating[0].rating))
               else:
                   rating_of_user.append(0)
           interactions_matrix.append(rating_of_user)
       interactions_matrix = coo_matrix(interactions_matrix)
     num_input = len(films)
     num_hidden_1 = 10
     num_hidden_2 = 5
     X = tf.placeholder(tf.float64, [None, num_input])
     weights = {
       'encoder_h1': tf.Variable(tf.random_normal([num_input,
num_hidden_1], dtype=tf.float64)),
       'encoder_h2': tf.Variable(tf.random_normal([num_hidden_1,
num_hidden_2], dtype=tf.float64)),
       'decoder_h1': tf.Variable(tf.random_normal([num_hidden_2,
num_hidden_1], dtype=tf.float64)),
       'decoder_h2': tf.Variable(tf.random_normal([num_hidden_1,
num_input], dtype=tf.float64)),
     }
     biases = {
       'encoder_b1': tf.Variable(tf.random_normal([num_hidden_1],
dtype=tf.float64)),
       'encoder_b2': tf.Variable(tf.random_normal([num_hidden_2],
dtype=tf.float64)),
       'decoder_b1': tf.Variable(tf.random_normal([num_hidden_1],
dtype=tf.float64)),
       'decoder_b2': tf.Variable(tf.random_normal([num_input],
dtype=tf.float64)),
```

```
    }
    # Building the encoder
    def encoder(x):
      # Encoder Hidden layer with sigmoid activation #1
      layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x,
weights['encoder_h1']), biases['encoder_b1']))
      # Encoder Hidden layer with sigmoid activation #2
      layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1,
weights['encoder_h2']), biases['encoder_b2']))
      return layer_2
    # Building the decoder
    def decoder(x):
      # Decoder Hidden layer with sigmoid activation #1
      layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x,
weights['decoder_h1']), biases['decoder_b1']))
      # Decoder Hidden layer with sigmoid activation #2
      layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1,
weights['decoder_h2']), biases['decoder_b2']))
      return layer_2
    # Construct model
    encoder_op = encoder(X)
    decoder_op = decoder(encoder_op)
    # Prediction
    y_pred = decoder_op
    # Targets are the input data.
    y_true = X

# Define loss and optimizer, minimize the squared error
    loss = tf.losses.mean_squared_error(y_true, y_pred)
    optimizer = tf.train.RMSPropOptimizer(0.03).minimize(loss)
    predictions = pd.DataFrame()
    # Define evaluation metrics
    eval_x = tf.placeholder(tf.int32, )
    eval_y = tf.placeholder(tf.int32, )
    pre, pre_op = tf.metrics.precision(labels=eval_x,
predictions=eval_y)
    # Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
local_init = tf.local_variables_initializer()

    for i in range(epochs):
              avg_cost = 0
              for batch in interactions_matrix:
                  _, l = session.run([optimizer, loss],
feed_dict={X: batch})
                  avg_cost += l
              avg_cost /= num_batches
    interactions_matrix = np.concatenate(interactions_matrix,
axis=0)
    preds = session.run(decoder_op, feed_dict={X:
interactions_matrix})
    print(len(preds.shape))
    pickle.dump(preds, open("model.p", "wb"))
```