



E-Learning Platform Development with Django and Docker

Kazakh-British Technical University

Murat Tinal

23MD0442

22.12.24

1. Executive Summary

The project creating an e-learning platform with Django and Docker was focused on building a flexible and scalable online learning platform. It used Django for backend development and Docker to create separate containers for the app. The platform includes essential features like user sign-up, course enrollment, managing lessons, quizzes, and payments. The Django Rest Framework (DRF) helped build an API to make the backend and frontend work smoothly together. Docker made sure the platform could be developed, tested, and deployed easily in different environments. A mock dataset was used to simulate real-life data and ensure the platform worked well for users. The main outcomes were the successful use of DRF for APIs, Docker for containers, and an efficient e-learning system. This platform can be expanded in the future with new features like mobile support, better analytics, and more integrations with other services.

2. Table of Contents

E-Learning Platform Development with Django and Docker	1
1. Executive Summary	2
2. Table of Contents	3
3. Introduction	4
4. System Architecture	4
5. Dockerfile	5
6. Docker-Compose	6
7. Docker Networking and Volumes	7
8. Django Models	8
9. Django Views	11
10. Templates	14
11. Django Rest Framework(DRF)	16
12. Conclusion	17
13. Screenshots project	18
14. References	23

3. Introduction

Education is moving online, making learning accessible anytime, anywhere. E-learning connects students, teachers, and resources in one platform, creating an interactive experience. This project aims to build an e-learning platform using Django, a popular web framework. Django's tools handle user management, data, and flexible APIs. Docker ensures easy setup, scalability, and deployment.

Key features include user accounts, courses, lessons, reviews, ratings, progress tracking, and tests. These tools support knowledge sharing and interaction between students and teachers. The project combines technology with practical learning, helping developers explore Django, Docker, and API integration while creating a scalable, user-friendly platform.

4. System Architecture

The system architecture of this project is based on the use of Django, which serves as the main framework for developing a web application, and Docker, which provides containerization and management of the development and deployment environment. The system is based on a structure divided into modules to provide flexibility, scalability, and easier code management.

The application consists of several key components. The course folder stores the main application files such as models, views, routes, and admin settings in Fig.1. These files are responsible for data management, query processing logic, and information display. The “static” directory contains static resources, which can include CSS, JavaScript, and images for the frontend. The main application, called eLearning, contains global project settings such as database configuration and high-level routes. There are also files here “settings.py”, “urls.py” and “wsgi.py”, which are responsible for setting up and launching the project. The “photos/courses” folder stores images related to courses, such as covers and visual elements. The page templates are located in the temps folder and include HTML files for various user interfaces, including registration, authorization, user profile, course descriptions, and other key interface components.

Docker plays an important role in ensuring the stable operation of the system. The “Dockerfile” file defines instructions for creating a container, and “docker-compose.yml” allows you to manage a multi-container architecture by combining a Django application and an SQLite database.

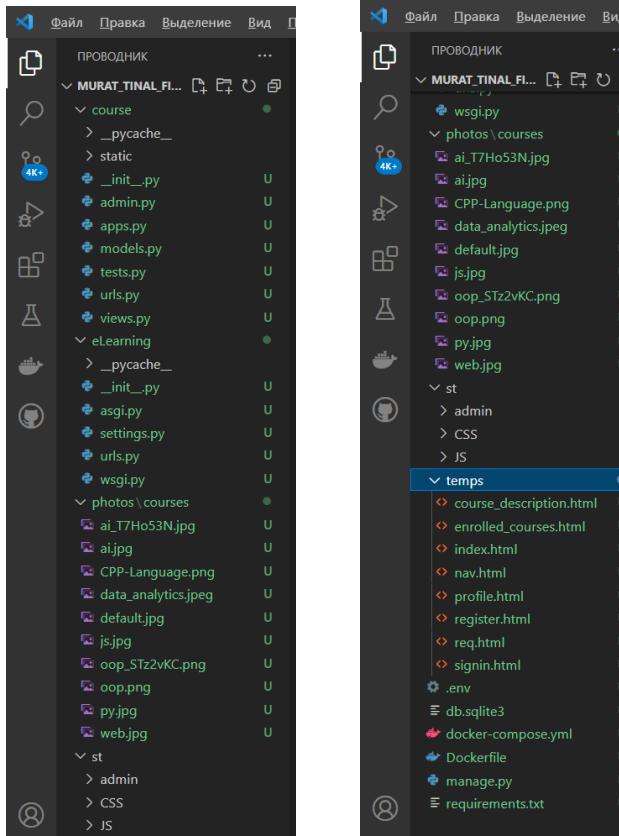


Fig.1

5. Dockerfile

This Dockerfile defines the setup for our Django-based e-learning application using Python 3.12-slim in *Fig.2* as the base image. It starts by installing essential tools like “curl” and building tools needed for compiling dependencies. The file also installs Rust and Cargo, which is required by some Python packages.

```

FROM python:3.12-slim
RUN apt-get update && apt-get install -y \
    curl \
    build-essential
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
ENV PATH="/root/.cargo/bin:${PATH}"
WORKDIR /app
COPY requirements.txt /app/
RUN pip install --no-cache-dir -r requirements.txt
RUN apt-get update && apt-get install -y libpq-dev
COPY . /app/
EXPOSE 8000
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

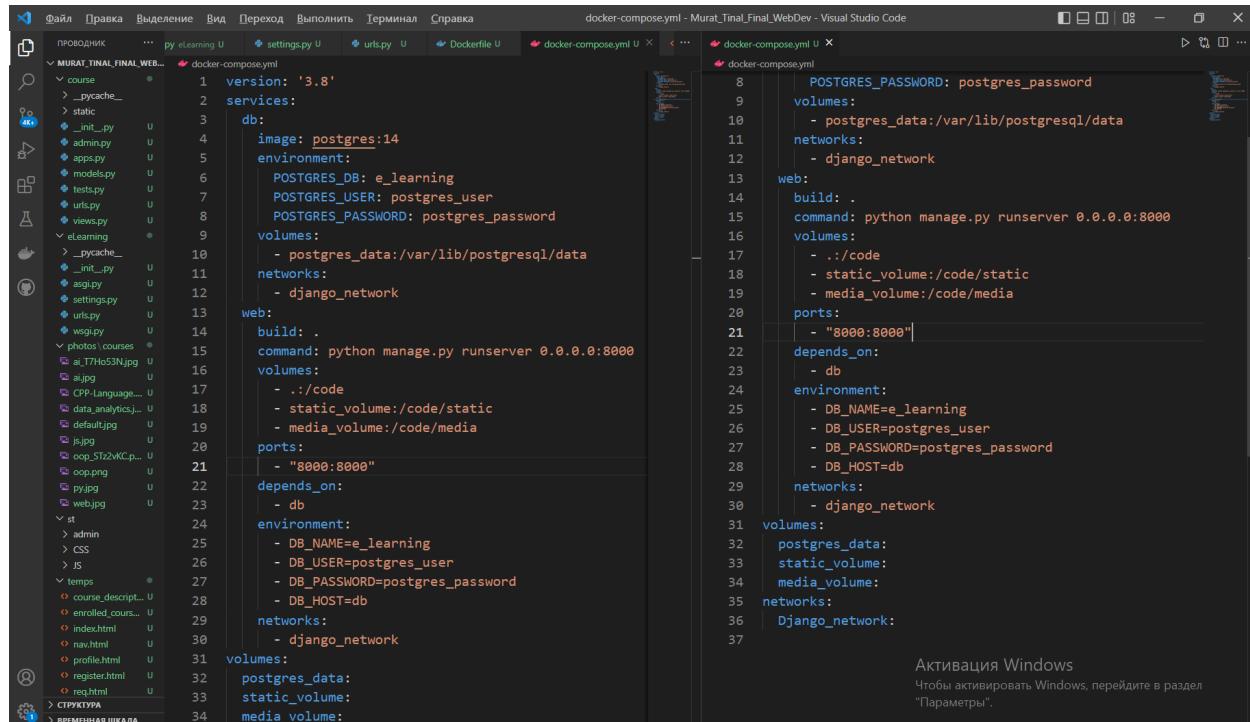
```

Fig.2

The application files are copied into the Docker container, and dependencies listed in “`requirements.txt`” are installed using “`pip`”. The PostgreSQL development package “`libpq-dev`” is installed to ensure that the app can connect to the PostgreSQL database.

The application is set to run on port 8000, and the last line defines the command to start the Django development server inside the container, making it accessible from any address. This configuration allows the application to run smoothly inside Docker, with all dependencies packaged together.

6. Docker-Compose



The screenshot shows the Visual Studio Code interface with two tabs open: `docker-compose.yml` and `docker-compose.yml`. The left pane displays the project structure for `MURAT_FINAL_WEB`, which includes a `py_elearning` folder containing Python files like `__init__.py`, `settings.py`, `urls.py`, and `Dockerfile`, along with static files such as `ai_7THo53N.jpg`, `ai.jpg`, and `CPP-Language...`. The right pane shows the `docker-compose.yml` file content:

```

version: '3.8'
services:
  db:
    image: postgres:14
    environment:
      POSTGRES_DB: e_learning
      POSTGRES_USER: postgres_user
      POSTGRES_PASSWORD: postgres_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - django_network
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - ./code
      - static_volume:/code/static
      - media_volume:/code/media
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      - DB_NAME=e_learning
      - DB_USER=postgres_user
      - DB_PASSWORD=postgres_password
      - DB_HOST=db
    networks:
      - django_network
    volumes:
      - postgres_data:
          static_volume:
          media_volume:
        networks:
          - Django_network

```

Fig.3

In Fig.3, I created a “`docker-compose.yml`” file to set up the task management application with a PostgreSQL database. The file defines two services: “`db`” for PostgreSQL and “`web`” for the Django application.

The “`db`” service uses the “`postgres:14`” image. I set environment variables to configure the database:

- 1) “`POSTGRES_DB`”: Specifies the name of the database, which is “`e_learning`”.
- 2) “`POSTGRES_USER`”: Sets the username for accessing the database, defined as “`postgres_user`”.
- 3) “`POSTGRES_PASSWORD`”: Sets the password for the database user, which is “`postgres_password`”.

Additionally, the database service uses a volume named “`postgres_data`” to ensure data is stored persistently at “`/var/lib/postgresql/data`”. This means that even if the container restarts or is removed, the database data remains intact.

The “`web`” service builds the Django application from the current directory using the Dockerfile. I specified the command to run the Django development server. The web service also

mounts volumes for the code “.”, static files “static_volume”, and media files “media_volume” to facilitate real-time changes during development.

7. Docker Networking and Volumes

The image shows a terminal window with three stacked code snippets representing Docker Compose files:

- Top snippet:** Shows environment variables and network definitions. It includes:
 - `environment:` with entries: `- DB_NAME=e_learning`, `- DB_USER=postgres_user`, `- DB_PASSWORD=postgres_password`, `- DB_HOST=db`.
 - `networks:` with entry: `- django_network`.
- Middle snippet:** Shows volume definitions. It includes:
 - `volumes:` with entry: `- postgres_data:/var/lib/postgresql/data`.
 - `networks:` with entry: `- django_network`.
- Bottom snippet:** Shows a `web:` service definition which extends the middle snippet's configurations.

Fig.4

Docker volumes in *Fig.4*, “postgres_data”, “static_volume”, and “media_volume”, help keep data safe even if a container stops or is recreated. For example, “postgres_data” holds the database, so no data is lost if the database container restarts.

To keep data safe when containers restart, we use volumes. For example, the `postgres_data` volume stores the database data, so it's not lost if the PostgreSQL container stops or gets recreated. This volume is linked to the `/var/lib/postgresql/data` folder, where PostgreSQL keeps its data. The `static_volume` and `media_volume` work the same way, storing static files and media files that users upload. These volumes are outside of the container, which means the data stays safe even if the container is recreated. The `static_volume` is connected to the `/code/static` folder, and the `media_volume` is connected to `/code/media`, making sure the app can always access these files.

Networks help the containers talk to each other. The `django_network` allows the web service and database containers to communicate easily, which is important for everything to work properly and keep the app running smoothly.

8. Django Models

The "SystemAdmin" class in *Fig.5* represents the system administrator who manages the platform. It includes fields such as "name", "number", "email" and "address" to store personal information about the administrator. The "username" and "password" fields are unique for each administrator, which ensures secure access. The "`__str__`" method is used to display the administrator's name when accessing this object. This class helps to manage the administrator's

data and ensures that only authorized administrators can log in. This is a simple model designed to store important information about the top managers of the system.

The screenshot shows a code editor with the title "models.py - Murat_Tinal_Final_WebDev - Vis". The file contains the following Python code:

```

1  Правка Выделение Вид Переход Выполнить Терминал Справка
2  ПРОВОДНИК ... __init__.py course U admin.py U models.py U tests.py U __init__.py eLearning U settings.py U urls.py
3  MURAT_TINAL_FINAL_WEB... course > models.py ...
4      3
5      4 class SystemAdmin(models.Model):
6          5     name = models.CharField(max_length=30, default="System Admin")
7          6     number = models.IntegerField(max_length=10)
8          7     email = models.EmailField(max_length=50)
9          8     address = models.TextField(max_length=50)
10         9     username = models.CharField(max_length=20, unique=True)
11        10     password = models.CharField(max_length=16)
12        11     def __str__(self):
13        12         return self.name
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

```

Fig.5

The screenshot shows a code editor with the following Python code:

```

15 15 class Course(models.Model):
16 16     category_choices = [ ('marketing','marketing'),
17 17         ('programming','Programming'),('web','Web Development'),('app','App Development'),
18 18         ('game','Game Development'),[('database','Database Management'),('network','Network Security'),
19 19         ('cloud','Cloud Computing'),('iot','Internet of Things'),('blockchain','Blockchain Technology'),
20 20         ('cyber','Cyber Security'),('vr','Virtual Reality'),('ar','Augmented Reality'),('mr','Mixed Reality'),
21 21         ('robotics','Robotics'),('bigdata','Big Data'),('datascience','Data Science'),('ai','Artificial Intelligence'),
22 22         ('ml','Machine Learning'),('dl','Deep Learning'),('cv','Computer Vision'),('nlp','Natural Language Processing')]
23
24 23     course_id = models.AutoField(primary_key=True)
25 24     course_name = models.CharField(max_length=100)
26 25     category = models.CharField(max_length=20,choices=category_choices)
27 26     description = models.TextField(max_length=500,blank=True)
28 27     is_active = models.BooleanField(default=True)
29 28     image_url = models.ImageField(upload_to='courses/',default='courses/default.jpg',blank=True)
30
31 30     def save(self, *args, **kwargs):
32 31         if self.pk:
33 32             try:
34 33                 old_course = Course.objects.get(pk=self.pk)
35 34                 if old_course.image_url != self.image_url:
36 35                     if old_course.image_url.name != 'courses/default.jpg':
37 36                         # Delete the old image file
38 37                         os.remove(old_course.image_url.path)
39 38             except Course.DoesNotExist:
40 39                 pass
41 40             super().save(*args, **kwargs)
42 41         def __str__(self):
43 42             return self.course_name
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
187
188
189
189
190
191
192
193
194
195
196
197
197
198
199
199
200
201
202
203
204
205
206
207
207
208
209
209
210
211
212
213
214
215
215
216
217
217
218
219
219
220
221
222
222
223
223
224
224
225
225
226
226
227
227
228
228
229
229
230
230
231
231
232
232
233
233
234
234
235
235
236
236
237
237
238
238
239
239
240
240
241
241
242
242
243
243
244
244
245
245
246
246
247
247
248
248
249
249
250
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1
```

deleting old course images when updating, making it easier to manage files. This class effectively organizes and manages course-related data.

```

class Student(models.Model):
    name = models.CharField(max_length=30, default="Student")
    number = models.IntegerField(max_length=10, blank=True, null=True)
    email = models.EmailField(max_length=50, blank=True)
    address = models.TextField(max_length=50, blank=True)
    username = models.CharField(max_length=20, unique=True, default="student")
    password = models.CharField(max_length=16, blank=True)
    enrolled_courses = models.ManyToManyField(Course, related_name='Student', blank=True)
    def __str__(self):
        return self.name

```

Fig.7

The "Student" class in **Fig.7** represents students using the platform. It includes fields such as "name", "number", "email address" and "email address" for personal information. The Username and Password fields ensure that each student has unique login credentials. Students can enroll in multiple courses using the "registered courses" field, which creates a link between students and the courses they attend. The "`__str__`" method shows the student's name when referring to the object. This class is necessary to manage information about students and their interaction with the courses available on the platform.

```

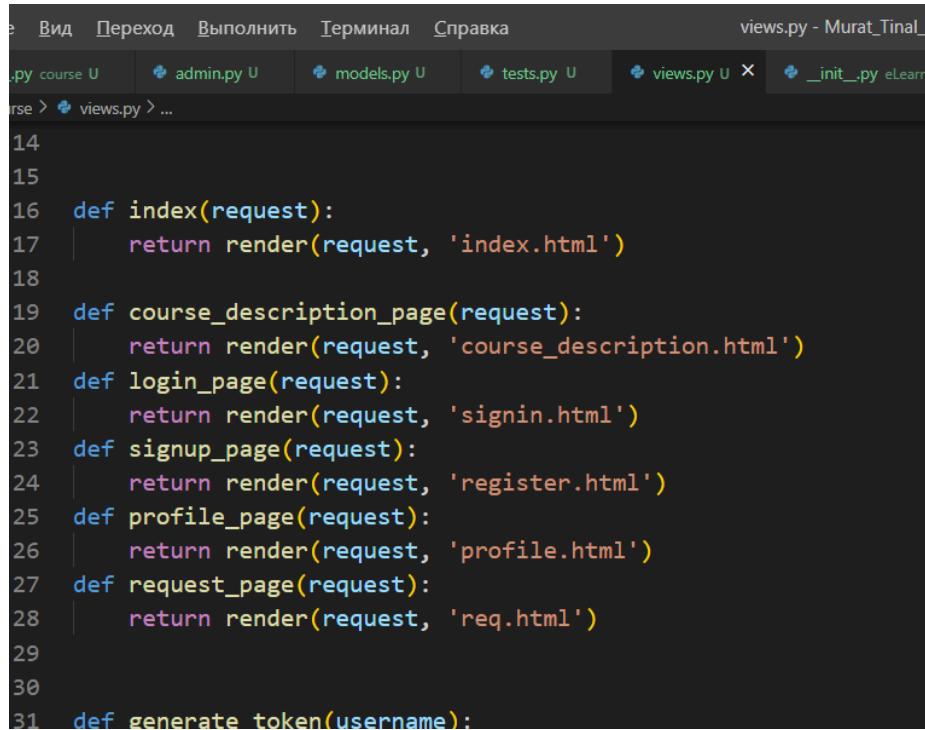
6
7 class CourseRequest(models.Model):
8     request_id = models.AutoField(primary_key=True)
9     course = models.ForeignKey(Course, on_delete=models.CASCADE, related_name='course_requests')
0     student = models.ForeignKey(Student, on_delete=models.CASCADE, related_name='course_requests')
1     request_date = models.DateField(auto_now=True)
2     reason = models.TextField(max_length=500, blank=True)
3     status = models.CharField(max_length=20, default="pending")
4
5     class Meta:
6         unique_together = ('course', 'student')
7     def __str__(self):
8         return f"{self.student.name} request for {self.course.course_name}"
9

```

Fig.8

The CourseRequest class tracks students' requests to enroll in certain courses. It has fields such as "request_id" for unique identification, "course" and "student" to link the request to the corresponding objects, as well as "request_date" to indicate the date when the request was made. The "reason" field allows students to explain why they want to take the course, and the "status" field indicates whether the request is under review, approved, or rejected. The "unique together" restriction ensures that a student cannot request the same course multiple times. This activity simplifies the process of managing course registration requests.

9. Django Views



The screenshot shows a code editor window with a dark theme. At the top, there is a menu bar with Russian labels: 'Вид' (View), 'Переход' (Transition), 'Выполнить' (Execute), 'Терминал' (Terminal), and 'Справка' (Help). Below the menu, there is a tab bar with several files: 'course.py' (marked as 'U'), 'admin.py' (marked as 'U'), 'models.py' (marked as 'U'), 'tests.py' (marked as 'U'), 'views.py' (marked as 'U' and highlighted in blue), and '_init_.py' (marked as 'eLearn'). The main area of the editor contains Python code for a Django application's views module. The code defines several functions:

```
14
15
16 def index(request):
17     return render(request, 'index.html')
18
19 def course_description_page(request):
20     return render(request, 'course_description.html')
21 def login_page(request):
22     return render(request, 'signin.html')
23 def signup_page(request):
24     return render(request, 'register.html')
25 def profile_page(request):
26     return render(request, 'profile.html')
27 def request_page(request):
28     return render(request, 'req.html')
29
30
31 def generate_token(username):
```

Fig.9

The function `index(request)` in **Fig.9** displays the main page of the application. When the user accesses this route, Django calls `render()` to process the request and returns an HTML `index.html`. The main page serves as a starting point where users can view general information about the educational platform or follow links to other pages. This feature is the basis of the user interface, providing easy access to the main functions of the application.

`course_description_page` is responsible for displaying the course description page. She uses a template `course_description.html`, which most likely contains textual and visual information about a particular course.

`login_page` loads the login page to the platform using a template `signin.html`. On this page, the user can enter their credentials (username and password).

`signup_page` is responsible for displaying the registration page by displaying a template `register.html`. It allows new users to create an account by entering fields such as name, email address, and password.

```

def login_view(request):
    if request.method=='POST':
        username = request.POST['username']
        password = request.POST['password']
        print(username,password)
        user = Student.objects.filter(username=username,password=password)
        if user.exists():
            token = generate_token(username)
            return JsonResponse({"status": "success", "token": token})
        else:
            return JsonResponse({"status": "your password or username is incorrect"})

```

Fig.10

login_view verifies the credentials sent via the POST request, and if they are correct, generates a JWT token. If the user is found, a token is created, which is returned to the client in JSON format. If the user does not exist, an error message is returned. The function demonstrates the basic authentication implementation. This can be improved by adding limits on the number of login attempts or temporarily locking the account after several failures.

```

def logout_view(request):
    if request.method=='POST':
        logout(request)
        return JsonResponse({"status": "success"})


def enrolled_courses_page(request):
    return render(request, 'enrolled_courses.html')


def all_courses(request):
    courses = Course.objects.filter(is_active=True)
    course_data = []
    for course in courses:
        course_dict = {
            "course_id": course.pk,
            "course_name": course.course_name,
            "category": course.category,
            "description": course.description,
            'is_active': course.is_active,
            "image_url": course.image_url.url
        }
        course_data.append(course_dict)

    for course in course_data:
        course["image_url"] = request.build_absolute_uri(course["image_url"])

    return JsonResponse({"courses": course_data})

```

Fig.11

```

2
3 def student_info(request):
4     if request.method=='POST':
5         auth_header = request.headers.get('Authorization')
6         token = auth_header.split(" ")[1]
7         try:
8             payload = jwt.decode(token, settings.SECRET_KEY, algorithms=["HS256"])
9             username = payload["username"]
10            print(username)
11            student = Student.objects.get(username=username)
12            course=get_enrolled_courses(username)
13            serialized_student = serializers.serialize('json', [student])
14            student=json.loads(serialized_student)
15            student_data=student[0]['fields']
16            return JsonResponse({"student": student_data, "enrolled_courses": course})
17        except Exception as e:
18            return JsonResponse({"status": "student not found"})
19

```

Fig.12

The student_info function uses a JWT to authenticate a user and retrieve their information and enrolled courses. It returns the student's details and courses in JSON format shown **Fig.12**.

```

def signup(request):
    if request.method=='POST':
        name=request.POST['name']
        number=request.POST['number']
        email=request.POST['email']
        address=request.POST['address']
        username=request.POST['username']
        password=request.POST['password']
        student = Student(name=name,number=number,email=email,address=address,username=username,password=password)
        try:
            student_exist = Student.objects.filter(username=username)
            if student_exist:
                return JsonResponse({"status": "student already exist"})
            else:
                student.save()
                token=generate_token(username)
                return JsonResponse({"status": "success", "token": token})
        except Exception as e:
            return JsonResponse({"status": e})

```

Fig.13

The signup function handles new user registration by saving the student's details in the database. It checks for username duplicates and generates a JWT upon successful registration shown in **Fig.13**.

```

def get_course(request):
    if request.method=='GET':
        course_id = request.GET['course_id']

        print(course_id)
        course = Course.objects.get(course_id=course_id)
        course_data = {
            "course_id": course.pk,
            "course_name": course.course_name,
            "category": course.category,
            "description": course.description,
            "image_url": course.image_url.url
        }
        course_data["image_url"] = request.build_absolute_uri(course_data["image_url"])

    return JsonResponse({"course": course_data})

```

Fig.14

The `get_course` function in **Fig.14** is used to get data about a specific course by its ID. It only processes GET requests by extracting the `course_id` parameter from the request. After receiving the ID, the function searches for the corresponding `Course` object in the database using the `get()` method of the `Course` model. If a course is found, data from its fields is collected in the `course_data` dictionary, which includes key attributes such as the course ID, its name, category, description, and image URL. A special feature of the function is the conversion of the relative image path to an absolute URL via the `request.build_absolute_uri()`. This is important for the correct display of images on client devices.

10. Templates

\temps\index.html: The first section, entitled "About App" in **Fig.15**, provides a brief description of the e-learning platform. It displays the name of the project, and text explaining the purpose of developing the platform, including the use of Django, Docker, and other technologies. The entire block is designed using embedded CSS styles.

The second section is devoted to courses. It has a heading "Courses" and a container that is initially empty but is designed to dynamically add information about courses. A JavaScript file JS/script.js using the tag `{%static %}`.

The third section contains an important note for users: only registered users can subscribe to the courses. A list and stylized text for explanations have been added to it.

```

<section id="about">
  <div class="container">
    
    <h1 style="color: #chocolate">E-Learning Platform(by Murat Tinal)</h1>
    <div class="section-content">
      <p style="background-color: #orange;">The goal of this project is to develop a robust e-learning platform using
    </div>
  </div>
</section>

<section id="courses">
  <div class="container">
    <h1 style="color: #orange;">Courses</h1>
    <div id="course-container" class="w-100">
      <!-- Courses will be dynamically added here -->
      <script src="{% static 'JS/script.js' %}"></script>
    </div>
  </div>
</section>

<section id="subscribe">
  <div class="container">
    <h1></h1>
    <div class="section-content">
      <ul style="list-style-type: none; padding-left: 0;">
        <li><span style="font-size: 20px;"></span></li>
        <li><span style="font-size: 20px;"></span></li>
      </ul>
      <p style="color: #red;">Note that you can't request a course if you are not signed in.</p>
    </div>
  </div>
</section>
{% endblock %}

```

Активация
Чтобы активировать
"Параметры".

Fig.15

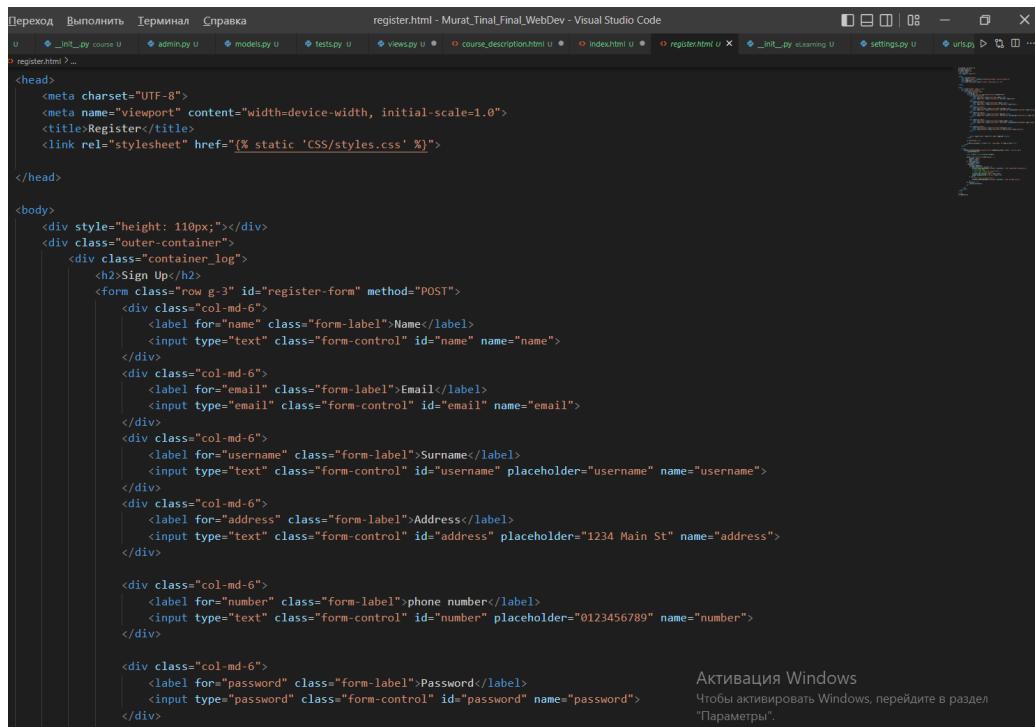
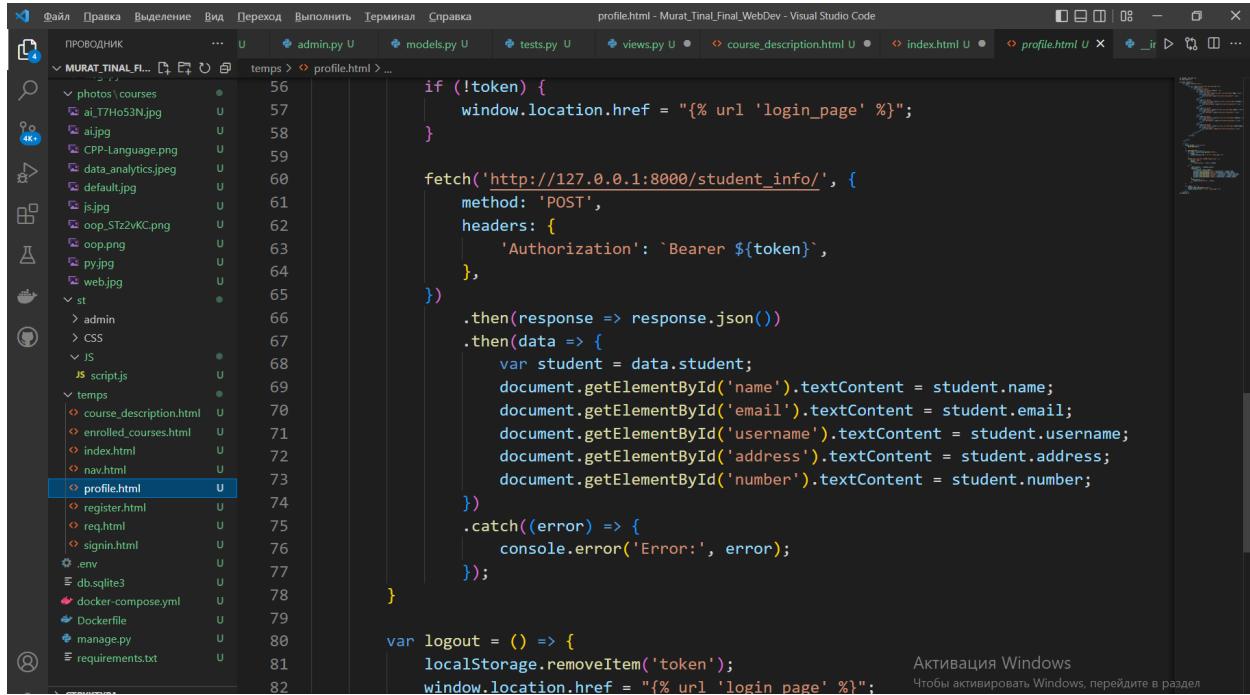


Fig.16

The form sends data using the POST method. However, instead of sending it to the server as standard, Js is used to prevent the page from reloading. A submission event is processed, data is collected using form data, and then sent to the server using the fetch method.

If registration is successful, the user is shown a success message, the token is saved in localStorage, and the user is redirected to the main page (the URL is set using { % url index % }). If the user already exists, an error message is displayed.



```

Файл Правка Выделение Вид Переход Выполнить Терминал Справка profile.html - Murat_Tinal_Final_WebDev - Visual Studio Code
ПРОВОДНИК temps > profile.html > ...
MURAT_TINAL_FINAL_WebDev
photos/courses
ai_17Ho53N.jpg
ai.jpg
CPP-Language.png
data_analytics.jpeg
default.jpg
js.jpg
oop_ST2vKC.png
oop.png
py.jpg
web.jpg
st
> admin
> CSS
JS
script.js
temps
course_description.html
enrolled_courses.html
index.html
nav.html
profile.html
register.html
req.html
signin.html
.env
db.sqlite3
docker-compose.yml
Dockerfile
manage.py
requirements.txt
if (!token) {
    window.location.href = "{% url 'login_page' %}";
}

fetch('http://127.0.0.1:8000/student_info/', {
    method: 'POST',
    headers: {
        'Authorization': `Bearer ${token}`,
    },
})
.then(response => response.json())
.then(data => {
    var student = data.student;
    document.getElementById('name').textContent = student.name;
    document.getElementById('email').textContent = student.email;
    document.getElementById('username').textContent = student.username;
    document.getElementById('address').textContent = student.address;
    document.getElementById('number').textContent = student.number;
})
.catch(error => {
    console.error('Error:', error);
});

var logout = () => {
    localStorage.removeItem('token');
    window.location.href = "{% url 'login_page' %}";
}

```

Fig.17

11. Django Rest Framework (DRF)

In **Fig.18** sets routes for the Django web application, and using DRF involves processing API requests using routes that allow interaction with application data. DRF provides convenient integration to create a RESTful API where I worked with resources such as courses and user requests.

Functions `get_course_request` in **Fig.19** implemented using DRF to process API requests. For example, instead of simple views, I used the View API to return data in JSON format, which is often required for client-server interaction. DRF helps manage data serialization, query processing, and the addition of authentication for the API.

```
urlpatterns = [
    path('', views.index, name='index'),
    path('course_description_page/', views.course_description_page, name='course_description_page'),
    path('login_page/', views.login_page, name='login_page'),
    path('signup_page/', views.signup_page, name='signup_page'),
    path('profile_page/', views.profile_page, name='profile_page'),
    path('request_page/', views.request_page, name='request_page'),
    path('enrolled_courses_page/', views.enrolled_courses_page, name='enrolled_courses_page'),  

    path('all_courses/', views.all_courses, name='all_courses'),
    path('student_info/', views.student_info, name='student_info'),
    path('login/', views.login_view, name='login'),
    path('signup/', views.signup, name='signup'),
    path('get_course_request/', views.get_course_request, name='get_course_request'),
    path('creat_course_request/', views.creat_course_request, name='creat_course_request'),
    path('get_course/', views.get_course, name='get_course'),  
]
```

Fig.18

```
def get_course_request(request):
    course_requests = CourseRequest.objects.all().filter(student_id=1)
    return JsonResponse({"course_requests": list(course_requests.values())})
```

Fig.19

```
models.py U tests.py U views.py U course_description.html U index.html U urls.py course U __init__.py eLearning  
eLearning > urls.py > ...
1 from django.contrib import admin
2 from django.urls import path, include
3 from django.conf.urls.static import static
4 from django.conf import settings
5
6 urlpatterns = [
7     path("admin/", admin.site.urls),
8     path("", include("courses.urls")),
9
10 ]
11
12 if settings.DEBUG:
13     urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Fig.20

```

2
3 def student_info(request):
4     if request.method=='POST':
5         auth_header = request.headers.get('Authorization')
6         token = auth_header.split(" ")[1]
7         try:
8             payload = jwt.decode(token, settings.SECRET_KEY, algorithms=["HS256"])
9             username = payload["username"]
10            print(username)
11            student = Student.objects.get(username=username)
12            course=get_enrolled_courses(username)
13            serialized_student = serializers.serialize('json', [student])
14            student=json.loads(serialized_student)
15            student_data=student[0]['fields']
16            return JsonResponse({"student": student_data, "enrolled_courses": course})
17        except Exception as e:
18            return JsonResponse({"status": "student not found"})
19

```

Fig.21

Fig.21 represents a handler for getting information about a student and his courses through a request processed using DRF. Here you can see the principle of interaction with authentication tokens, which is often used in APIs created using the Django Rest Framework. The incoming request checks the POST method and uses the Authorization header to extract the JWT token. DRF is often used to work with tokens due to the built-in authentication tools.

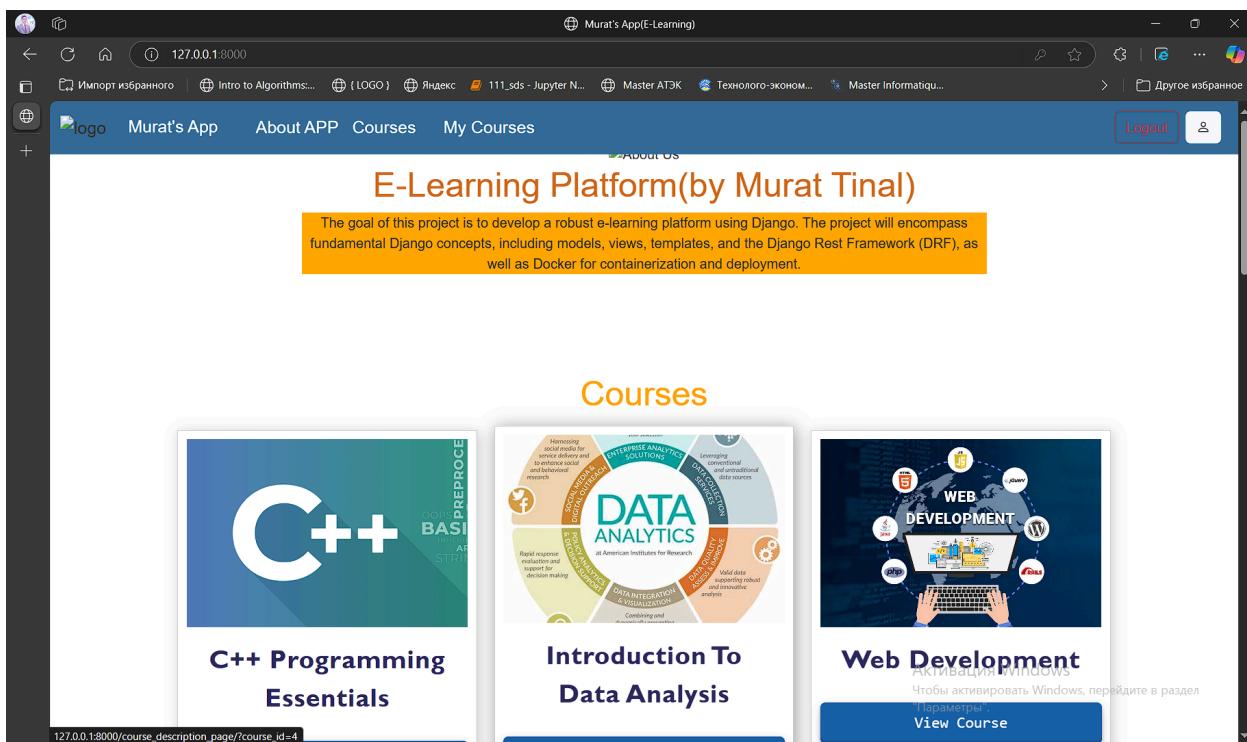
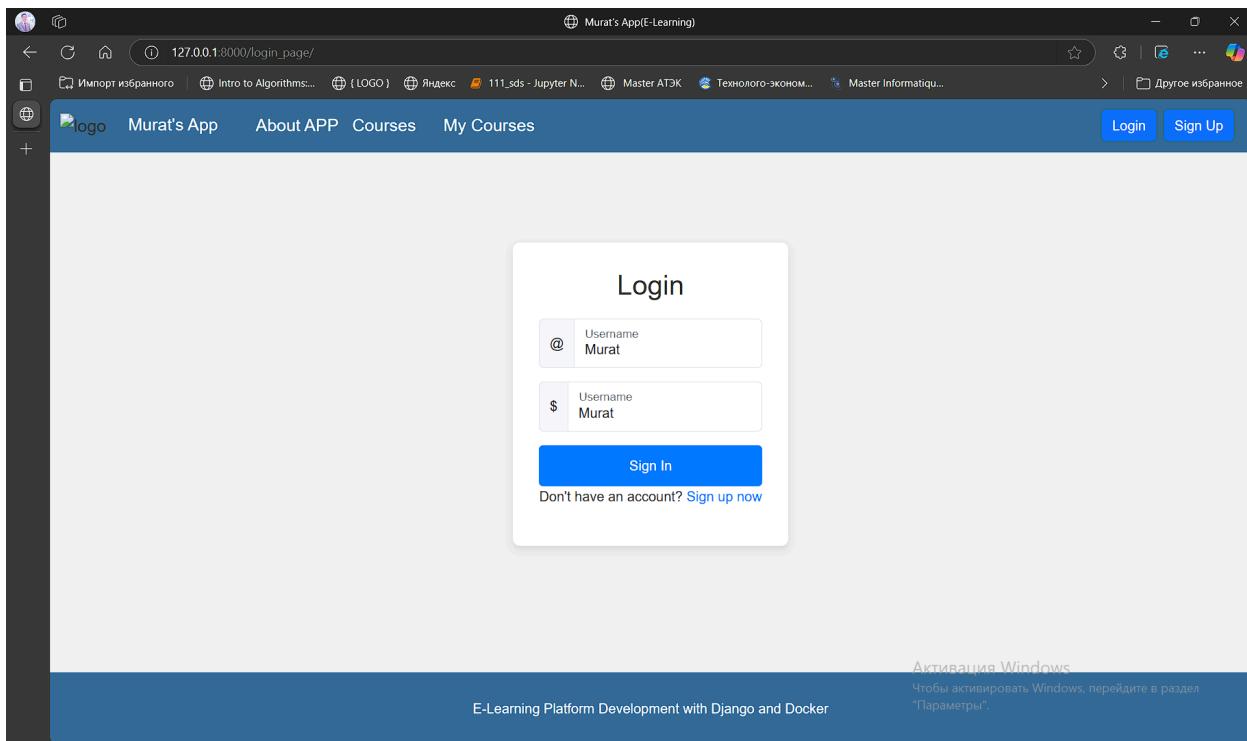
The student's object is then extracted from the database, and the data is serialized to turn it into a JSON format. DRF provides a powerful serialization engine that could be improved here by using a ModelSerializer instead of manual serialization. The function returns student data and a list of his courses in JSON format, which is ideal for the REST API. In case of an error, the "student not found" status is returned, which can also be standardized using DRF.

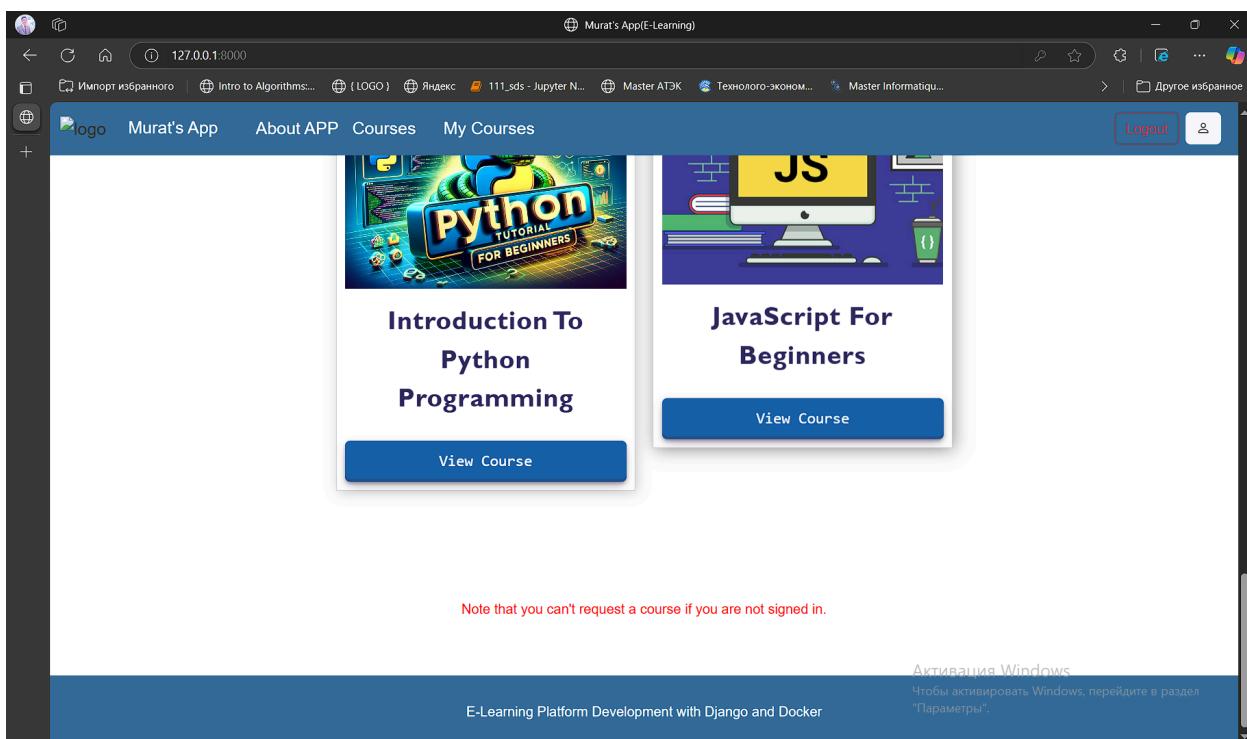
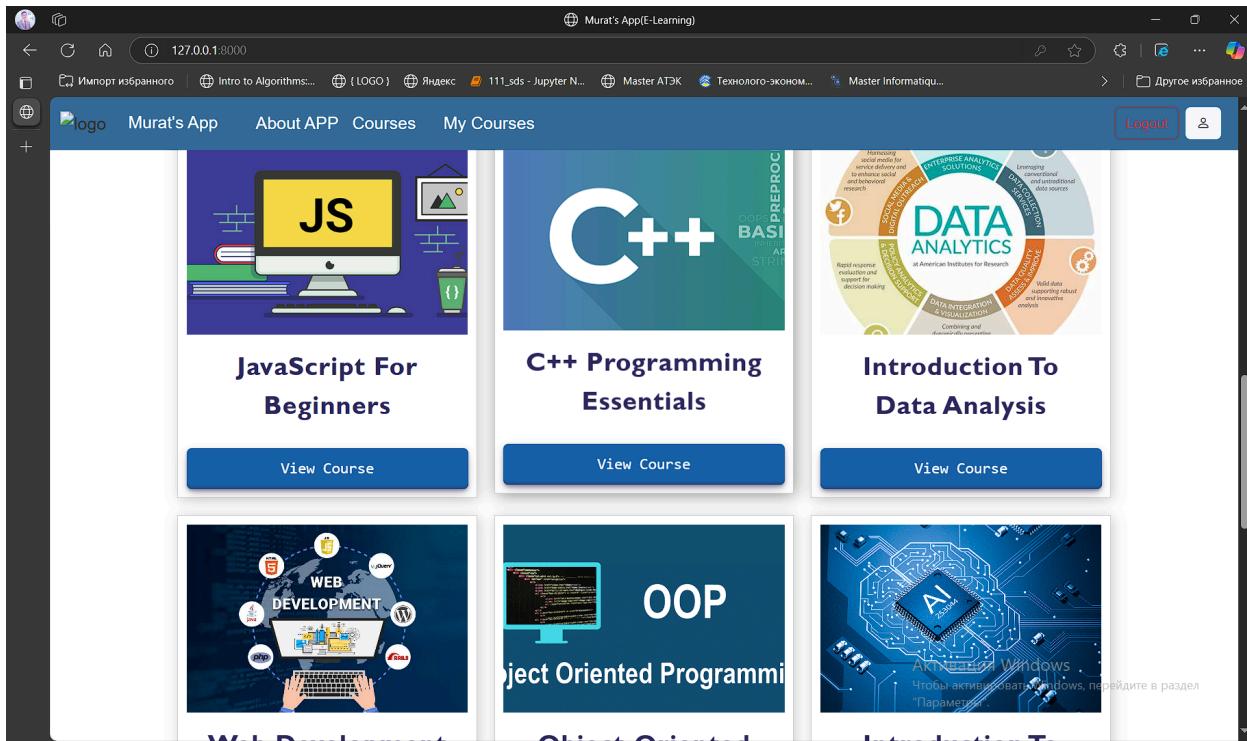
12. Conclusion

In conclusion, this project demonstrated the effectiveness of combining Django, Docker, and DRF in developing a full-fledged e-learning platform. The containerization provided a stable and scalable environment, while Django's powerful features enabled smooth development of the backend. DRAFT allowed for the creation of robust APIs, enhancing system interaction with the front end. Challenges such as Docker networking and data persistence were overcome through careful configuration. Overall, the platform is scalable, maintainable, and can be easily extended with new features. Future improvements may include enhanced security, user interface upgrades, and further optimization for large-scale usage.

Here you can see my screenshots of my project (GitHub):

https://github.com/kiraakz/Murat_Tinal_Final_WebDev



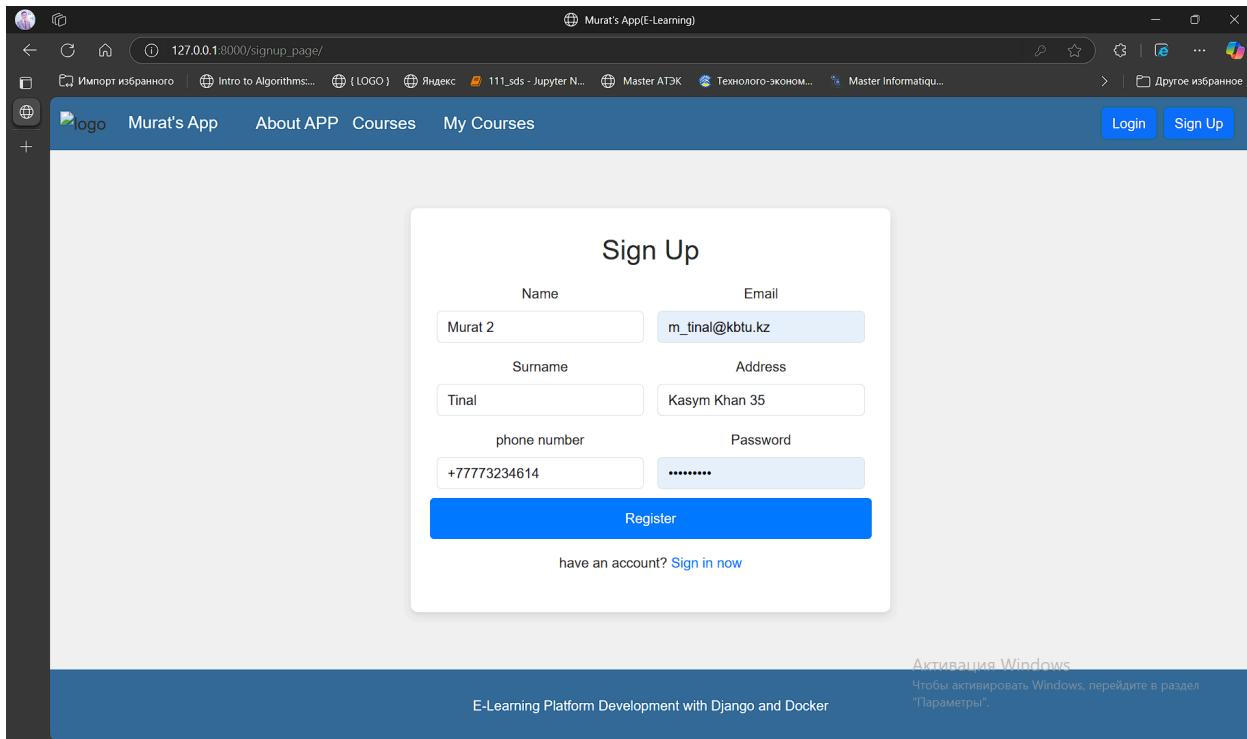


The screenshot shows a web browser window with the URL `127.0.0.1:8000/course_description_page/?course_id=2`. The page title is "Murat's App(E-Learning)". The main content area displays a course card for "C++ Programming Essentials". The card features a blue gradient background with the text "PROGRAMMING LANGUAGE" at the top left and "C++" in large white letters in the center. Below the card, the course title "C++ Programming Essentials" is displayed in bold black text. A description follows: "Description: Learn the fundamentals of C++, a powerful and widely-used programming language." A "Content:" section lists topics such as C++ Syntax and Structure, Data Types and Variables, Control Flow (loops, conditionals), Functions and Scope, Pointers and Memory Management, Classes and Objects, Inheritance and Polymorphism, and Standard Template Library (STL). A note states: "This course will cover the core concepts of C++ programming, enabling learners to write efficient and effective code in C++." At the bottom of the card is a red "Request Course" button. To the right of the card, there is a sidebar with the text "Активация Windows" and a note: "Чтобы активировать Windows, перейдите в раздел \"Параметры\"." The browser's address bar and various tabs are visible at the top.

The screenshot shows a web browser window with the URL `127.0.0.1:8000/request_page/?course_id=2`. The page title is "Murat's App(E-Learning)". The main content area displays a form titled "Please fill out the form to request the course". Inside the form, there is a section labeled "Reason for requesting the course" containing a text input field with the placeholder text "I need this course to improve my c++ skill". Below the input field is a dark blue "Submit" button. At the bottom of the page, there is a footer with the text "E-Learning Platform Development with Django and Docker". To the right of the footer, there is a sidebar with the text "Активация Windows" and a note: "Чтобы активировать Windows, перейдите в раздел \"Параметры\"." The browser's address bar and various tabs are visible at the top.

The screenshot shows a web browser window for 'Murat's App(E-Learning)' at the URL 127.0.0.1:8000/enrolled_courses_page/. The page has a blue header with navigation links: 'Murat's App', 'About APP', 'Courses', 'My Courses', 'Logout', and a user icon. The main content area is titled 'Murat's Courses' and lists three courses: 'C++ Programming Essentials', 'Web Development', and 'Introduction to Deep Learning'. Each course entry includes a 'View Course' button. The background features vertical orange bars on the left and right sides. A watermark for 'E-Learning Platform Development with Django and Docker' is visible at the bottom.

The screenshot shows a web browser window for 'Murat's App(E-Learning)' at the URL 127.0.0.1:8000/profile_page/. The page has a blue header with navigation links: 'Murat's App', 'About APP', 'Courses', 'My Courses', 'Logout', and a user icon. The main content area is titled 'Profile' and displays the user's information: Name: Murat, Surname: Tinal, Email: m_tinal@kbtu.kz, Address: Kasym Khan 33, and Phone Number: 77773234613. The background features vertical orange bars on the left and right sides. A watermark for 'E-Learning Platform Development with Django and Docker' is visible at the bottom.



15. References

- <https://buttercms.com/blog/build-an-online-learning-platform-with-django/>
- <https://www.educative.io/projects/creating-an-e-learning-website-using-django>
- <https://python.plainenglish.io/how-i-built-an-online-learning-platform-with-django-in-30-days-udemy-clone-a384ba507cf7>
- <https://github.com/ahmedEid1/E-Learning-Platform>
- <https://github.com/ashish-makes/django-lms>
- <https://stackskills.com/p/django-projects-e-learning-portal>
- <https://medium.com/@emmanuelsibanda/building-an-e-learning-school-minimum-viable-product-on-django-cf81dc8280c6>
- <https://github.com/ShivamRohillaa/E-learning-Django->
- <https://www.youtube.com/watch?v=hEMkB5wngSc&list=PLTV1jAY3nKHMotVkJWRcqH4EGPQqliQZ23>
- <https://www.youtube.com/watch?v=sREYiEsooew&list=PLeQ4zRISjJn96nZJAIFVxjE6k9LEexyKU>
- <https://www.youtube.com/watch?v=3z0XtYwly4w&list=PLxQ9RfumYrxa86SoJkszzrCURi2Rw74eg>
- <https://www.youtube.com/watch?v=aK9Y-0AsbA4&list=PLSJxovilIyDGkHNqlrPSU2kXu1aophIkG>
- <https://www.djangoproject-rest-framework.org/>
- <https://django.fun/articles/tutorials/kratko-o-django-rest-framework/>
-