

Unit Testing Document.

Overview

This document provides an overview of the unit testing conducted for the `Academix_Portal` Django application. The testing is divided into two major components: `test_models.py` and `test_views.py`. The unit tests were executed using the `python manage.py test` command.

Models Testing (test_models.py)

```
1 class TestModels(TestCase):
2     def setUp(self):
3         self.user_student = User.objects.create_user(username='shrikar', password='shrikar123')
4         self.user_faculty = User.objects.create_user(username='aakash', password='aakash123')
5
6         self.faculty = faculty_profile.objects.create(
7             user=self.user_faculty,
8             first_name='aakash',
9             middle_name='xyz',
10            last_name='patel'
11        )
12
13        self.student = student_profile.objects.create(
14            user=self.user_student,
15            first_name='shrikar',
16            middle_name='shaileshbhai',
17            last_name='padaliya',
18            batch=2022,
19            branch='ICT',
20            program='B.Tech'
21        )
22
23        self.course = Course.objects.create(
24            name='Computer Science 101',
25            course_code='CS101',
26            description='Introduction to Computer Science',
27            faculty=self.faculty
28        )
29        # creating an assignment here
30        self.assignment = Assignment.objects.create(
31            name='Assignment 1',
32            description='First assignment',
33            duedate=datetime(2023, 12, 1, 0, 0, 0),
34            max_grade=100,
35            attachment='aakash_assignment1',
36            assignment_course=self.course
37        )
38
39        # creating a student submission
40        self.submission = Submission.objects.create(
41            student=self.student,
42            graded=False,
43            grade=None,
```

```

44         work='shrikar_work1',
45         feedback='Good work!',
46         assignment=self.assignment
47     )
48
49     def test_duplicate_usernames(self):
50         # Testing for uniqueness of usernames
51         user_duplicate = User(username='shrikar', password='testpassword')
52         with self.assertRaises(Exception):
53             user_duplicate.save()
54
55     def test_student_enroll_in_course(self):
56         # Testing if a student can enroll in a course
57         self.course.studentlist.add(self.student)
58         self.assertEqual(self.course.studentlist.count(), 1)
59
60     def test_course_with_no_faculty(self):
61         # Testing create a course without a faculty
62
63         with self.assertRaises(IntegrityError):
64             course_no_faculty = Course.objects.create(
65                 name='No Faculty Course',
66                 course_code='NF101',
67                 description='Course without faculty'
68             )
69
70     def test_course_without_students(self):
71         # Testing create a course without students
72         course_empty = Course.objects.create(
73             name='Empty Course',
74             course_code='EC101',
75             description='Course with no students',
76             faculty=self.faculty
77         )
78         self.assertEqual(course_empty.studentlist.count(), 0)
79
80     def test_student_courses_relationship(self):
81         # Testing relationship between student and courses
82         self.student.student_courses.add(self.course)
83         self.assertEqual(self.student.student_courses.count(), 1)
84         self.assertEqual(self.student.student_courses.first().name, 'Computer Science 101')
85
86     def test_faculty_courses_relationship(self):
87         # Testing relationship between faculty and courses
88         self.faculty.faculty_courses.add(self.course)
89         self.assertEqual(self.faculty.faculty_courses.count(), 1)
90         self.assertEqual(self.faculty.faculty_courses.first().name, 'Computer Science 101')
91
92     def test_course_students_and_faculty(self):
93         # Testing that a course has both students and a faculty
94         self.course.studentlist.add(self.student)
95         self.assertEqual(self.course.studentlist.count(), 1)
96         self.assertEqual(self.course.faculty, self.faculty)
97
98     def test_student_str_method(self):
99
100         self.assertEqual(str(self.student), 'shrikar')
101

```

```

102     def test_faculty_str_method(self):
103
104         self.assertEqual(str(self.faculty), 'aakash')
105
106     def test_course_str_method(self):
107
108         self.assertEqual(str(self.course), 'Computer Science 101')
109
110
111     def test_assignment_str_method(self):
112         # Testing the __str__ method of Assignment
113         self.assertEqual(str(self.assignment), 'Assignment 1')
114
115     def test_submission_str_method(self):
116         # Testing the __str__ method of Submission
117         expected_str = f'{self.assignment.assignment_course.course_code} shrikarpadaliya'
118         self.assertEqual(str(self.submission), expected_str)
119
120     def test_submission_defaults(self):
121         # Testing submission default values
122         # Create a new assignment to avoid unique constraint violation
123         new_assignment = Assignment.objects.create(
124             name='Assignment 2',
125             description='Second assignment',
126             duedate=datetime(2023, 12, 2, 0, 0, 0),
127             max_grade=90,
128             attachment='http://example.com/assignment2',
129             assignment_course=self.course
130         )
131         # Use a different student to avoid unique constraint violation
132         new_student = student_profile.objects.create(
133             user=User.objects.create_user(username='mustafa', password='mustafa123'),
134             first_name='mustafa',
135             last_name='lokhandwala',
136             batch=2022,
137             branch='CSE',
138             program='B.Tech'
139         )
140
141         submission_defaults = Submission(student=new_student, assignment=new_assignment)
142         submission_defaults.save()
143         self.assertFalse(submission_defaults.graded)
144         self.assertIsNone(submission_defaults.grade)
145         self.assertIsNone(submission_defaults.feedback)
146         self.assertIsNotNone(submission_defaults.timestamp) # Ensure timestamp is not None
147
148     def test_assignment_due_date(self):
149         # Testing assignment due date
150         self.assertEqual(self.assignment.duedate, datetime(2023, 12, 1, 0, 0, 0))
151
152     def test_submission_timestamp_auto_now_add(self):
153         # Testing that the timestamp in Submission is set automatically on creation
154         new_assignment = Assignment.objects.create(
155             name='Assignment 2',
156             description='Second assignment',
157             duedate=datetime(2023, 12, 2, 0, 0, 0),
158             max_grade=90,
159             attachment='http://example.com/assignment2',

```

```

160         assignment_course=self.course
161     )
162     # Use a different student to avoid unique constraint violation
163     new_student = student_profile.objects.create(
164         user=User.objects.create_user(username='mustafa', password='mustafa123'),
165         first_name='mustafa',
166         last_name='lokhandwala',
167         batch=2022,
168         branch='CSE',
169         program='B.Tech'
170     )
171     submission_timestamp = Submission.objects.create(
172         student=new_student,
173         graded=False,
174         grade=None,
175         work='shrikar_work',
176         feedback='Great job!',
177         assignment=new_assignment
178     )
179     self.assertIsNotNone(submission_timestamp.timestamp)
180
181     def test_submission_grade_update(self):
182         # Testing updating the grade in Submission
183         updated_grade = 95
184         self.submission.grade = updated_grade
185         self.submission.save()
186         self.assertEqual(Submission.objects.get(id=self.submission.id).grade, updated_grade)
187
188     def test_submission_feedback_update(self):
189         # Testing updating the feedback in Submission
190         updated_feedback = 'Excellent work!'
191         self.submission.feedback = updated_feedback
192         self.submission.save()
193         self.assertEqual(Submission.objects.get(id=self.submission.id).feedback, updated_feedback)
194
195     def test_assignment_course_relationship(self):
196         # Testing relationship between Assignment and Course
197         self.assertEqual(self.assignment.assignment_course, self.course)
198
199     def test_submission_assignment_relationship(self):
200         # Testing relationship between Submission and Assignment
201         self.assertEqual(self.submission.assignment, self.assignment)
202
203     def test_assignment_submission_relationship(self):
204         # Testing relationship between Assignment and Submission
205         assignment_submissions = self.assignment.submission_set.all()
206         self.assertEqual(assignment_submissions.count(), 1)
207         self.assertEqual(assignment_submissions.first(), self.submission)
208
209     def test_duplicate_assignment_name(self):
210         # Testing for uniqueness of assignment names within a course
211         Assignment.objects.create(
212             name='Assignment temp',
213             description='First assignment',
214             duedate=datetime(2023, 12, 1, 0, 0, 0),
215             max_grade=100,
216             attachment='aakash_assignment1',
217             assignment_course=self.course

```

```

218         )
219
220     with self.assertRaises(IntegrityError):
221         Assignment.objects.create(
222             name='Assignment temp',
223             description='Duplicate assignment name',
224             duedate=datetime(2023, 12, 2, 0, 0, 0),
225             max_grade=90,
226             attachment='aakash_assignment1',
227             assignment_course=self.course
228         )
229
230     def test_submission_student_assignment_uniqueness(self):
231         # Testing for uniqueness of submissions by student for a specific assignment
232         assignment = Assignment.objects.create(
233             name='Assignment 2',
234             description='Second assignment',
235             duedate=datetime(2023, 12, 2, 0, 0, 0),
236             max_grade=90,
237             attachment='aakash_assignment2',
238             assignment_course=self.course
239         )
240         Submission.objects.create(
241             student=self.student,
242             graded=False,
243             grade=None,
244             work='shrikar_work2',
245             feedback='Good work!',
246             assignment=assignment
247         )
248         with self.assertRaises(IntegrityError):
249             Submission.objects.create(
250                 student=self.student,
251                 graded=False,
252                 grade=None,
253                 work='shrikar_work2',
254                 feedback='Another submission',
255                 assignment=assignment
256             )
257
258     def test_assignment_course_foreign_key_constraint(self):
259         # Testing integrity constraint for assignment_course foreign key
260         with self.assertRaises(IntegrityError):
261             Assignment.objects.create(
262                 name='Assignment 3',
263                 description='Third assignment',
264                 duedate=datetime(2023, 12, 3, 0, 0, 0),
265                 max_grade=80,
266                 attachment='aakash_assignment3',
267                 assignment_course=None
268             )
269
270     def test_submission_assignment_foreign_key_constraint(self):
271         # Testing integrity constraint for submission assignment foreign key
272         with self.assertRaises(IntegrityError):
273             Submission.objects.create(
274                 student=self.student,
275                 graded=False,

```

```

276         grade=None,
277         work='http://example.com/submission3',
278         feedback='Yet another submission',
279         assignment=None
280     )
281
282     def test_query_creation(self):
283         # Testing creation of a query
284         query_instance = query.objects.create(
285             course=self.course,
286             user=self.student,
287             qry='Doubt about the lecture content'
288         )
289         self.assertEqual(str(query_instance), 'CS101 shrikar padaliya')
290         self.assertIsNotNone(query_instance.timestamp)
291         self.assertEqual(query_instance.qry, 'Doubt about the lecture content')
292         self.assertIsNone(query_instance.reply)
293
294     def test_query_reply(self):
295         # Testing replying to a query
296         query_instance = query.objects.create(
297             course=self.course,
298             user=self.student,
299             qry='Doubt about the lecture content'
300         )
301         reply_text = 'This is a reply.'
302         query_instance.reply = reply_text
303         query_instance.save()
304         self.assertEqual(query_instance.reply, reply_text)
305
306     def test_query_defaults(self):
307         # Testing default values of a query
308         query_instance = query.objects.create(
309             course=self.course,
310             user=self.student,
311             qry='Doubt about the lecture content'
312         )
313         self.assertIsNone(query_instance.reply)
314
315     def test_query_course_foreign_key_constraint(self):
316         # Testing integrity constraint for query course foreign key
317         with self.assertRaises(IntegrityError):
318             query.objects.create(
319                 course=None,
320                 user=self.student,
321                 qry='Doubt about the lecture content'
322             )
323
324     def test_query_user_foreign_key_constraint(self):
325         # Testing integrity constraint for query user foreign key
326         with self.assertRaises(IntegrityError):
327             query.objects.create(
328                 course=self.course,
329                 user=None,
330                 qry='Doubt about the lecture content'
331             )

```

Test Coverage

- **Total Tests:** 30
- **Execution Time:** 10.698s

Summary

The `test_models.py` file focuses on testing the models of the `Academix_Portal` application. Key models tested include `student_profile`, `faculty_profile`, `Course`, `Assignment`, `Submission`, and `query`. The tests cover various functionalities such as model relationships, string representations, default values, and integrity constraints.

Noteworthy Tests

1. `test_duplicate_usernames`
 - Ensures uniqueness of usernames by attempting to create a duplicate user and expecting an exception.
2. `test_student_enroll_in_course`
 - Verifies if a student can successfully enroll in a course.
3. `test_assignment_due_date`
 - Checks if the due date of an assignment is correctly set.
4. `test_submission_grade_update`
 - Tests updating the grade in a submission and confirms the update.
5. `test_query_creation`
 - Validates the creation of a query instance with appropriate attributes.

Output

```
1 Found 30 test(s).
2 Creating test database for alias 'default'...
3 System check identified no issues (0 silenced).
4 .....
5 -----
6 Ran 30 tests in 10.698s
7
8 OK
9 Destroying test database for alias 'default'...
```

Conclusion

All 30 tests in `test_models.py` passed successfully, indicating that the models and database formation is correct.

Views Testing (test_views.py)

```
1 class TestViews(TestCase):
2
3     def setUp(self):
4         self.factory = RequestFactory()
5         self.client = Client()
6         self.user_student = User.objects.create_user(email='shrikar@daiict.ac.in', username='shrikar', password='password')
7         self.user_faculty = User.objects.create_user(email='aakash@daiict.ac.in', username='aakash', password='password')
8         self.user = User.objects.create_user(username='testuser', password='password')
9
10        self.faculty = faculty_profile.objects.create(
11            user=self.user_faculty,
12            first_name='aakash',
13            middle_name='xyz',
14            last_name='patel'
15        )
16
17        self.student = student_profile.objects.create(
18            user=self.user_student,
19            first_name='shrikar',
20            middle_name='shaileshbhai',
21            last_name='padaliya',
22            batch=2022,
23            branch='ICT',
24            program='B.Tech'
25        )
26
27        self.course = Course.objects.create(
28            name='Computer Science 101',
29            course_code='CS101',
30            description='Introduction to Computer Science',
31            faculty=self.faculty
32        )
33
34        self.assignment = Assignment.objects.create(
35            name='Assignment 1',
36            description='First assignment',
37            duedate=datetime(2023, 12, 1, 0, 0, 0),
38            max_grade=100,
39            attachment='aakash_assignment1',
40            assignment_course=self.course
41        )
42
43
44        self.my_course_url = reverse('mycourse')
45        self.otp_ver = reverse('otp_ver')
46        self.view_assignments_url = reverse('view_assignments', args=['CS101'])
47        self.profile_url = reverse('students_profile')
48        self.view_announcements_url = reverse('announcements', args=['CS101'])
49        self.view_materials_url = reverse('materials', args=['CS101'])
50        self.view_query_url = reverse('view_query', args=['CS101'])
51        self.view_feedback_url = reverse('feedback', args=['CS101'])
52        self.view_student_list_url = reverse('student_list', args=['CS101'])
53        self.add_assignment_url = reverse('add_assignment', args=['CS101'])
```



```

54     self.add_announcement_url = reverse('addannouncement', args=['CS101'])
55     self.add_submission_url = reverse('add_submission', args=['CS101', 'Assignment 1'])
56
57 # testing view enrolled courses page
58 def test_my_course_GET(self):
59     login = self.client.login(username='shrikar', password='shrikar123')
60     response = self.client.get(self.my_course_url)
61     self.assertEqual(response.status_code, 200)
62     self.assertTemplateUsed(response, 'my_course_student.html')
63     self.assertTemplateUsed(response, 'base.html')
64
65 # tests for view assignment page faculty side
66 def test_view_assignments_faculty_GET(self):
67     login = self.client.login(username='aakash', password='aakash123')
68     response = self.client.get(self.view_assignments_url)
69     self.assertEqual(response.status_code, 200)
70     self.assertTemplateUsed(response, 'view_assignments_faculty.html')
71
72 #tests for view assignment page student side
73 def test_view_assignments_GET(self):
74     login = self.client.login(username='shrikar', password='shrikar123')
75     response = self.client.get(self.view_assignments_url)
76     self.assertEqual(response.status_code, 200)
77     self.assertTemplateUsed(response, 'view_assignments.html')
78     self.assertTemplateUsed(response, 'navbar.html')
79
80 #tests for view profile page
81 def test_view_profile_GET(self):
82     login = self.client.login(username='shrikar', password='shrikar123')
83     response = self.client.get(self.profile_url)
84     self.assertEqual(response.status_code, 200)
85     self.assertTemplateUsed(response, 'student_profile.html')
86
87 #tests for view announcements page
88 def test_view_announcements_GET(self):
89     login = self.client.login(username='shrikar', password='shrikar123')
90     response = self.client.get(self.view_announcements_url)
91     self.assertEqual(response.status_code, 302)
92
93 #tests for view materials page
94 def test_view_materials_GET(self):
95     login = self.client.login(username='shrikar', password='shrikar123')
96     response = self.client.get(self.view_materials_url)
97     self.assertEqual(response.status_code, 200)
98     self.assertTemplateUsed(response, 'materials.html')
99
100 #tests for view query page
101 def test_view_query_GET(self):
102     login = self.client.login(username='shrikar', password='shrikar123')
103     response = self.client.get(self.view_query_url)
104     self.assertEqual(response.status_code, 200)
105     self.assertTemplateUsed(response, 'view_query.html')
106
107 #tests for view feedback page student side
108 def test_view_feedback_GET(self):
109     login = self.client.login(username='shrikar', password='shrikar123')
110     response = self.client.get(self.view_feedback_url)
111     self.assertEqual(response.status_code, 200)

```

```
112         self.assertTemplateUsed(response, 'add_feedback.html')
113
114     #tests for view feedback page faculty side
115     def test_view_feedback_faculty_GET(self):
116         login = self.client.login(username='aakash', password='aakash123')
117         response = self.client.get(self.view_feedback_url)
118         self.assertEqual(response.status_code, 200)
119         self.assertTemplateUsed(response, 'feedback_faculty.html')
120
121     #tests for view enrolled student list
122     def test_view_student_list_GET(self):
123         login = self.client.login(username='shrikar', password='shrikar123')
124         response = self.client.get(self.view_student_list_url)
125         self.assertEqual(response.status_code, 200)
126         self.assertTemplateUsed(response, 'student_list.html')
127
128     # tests for adding assignment
129     def test_add_assignment_POST(self):
130         login = self.client.login(username='aakash', password='aakash123')
131         data = {
132             'name' : 'Assignment 2',
133             'max_grade': '100',
134             'description' : 'Second assignment',
135             'duedate' : datetime(2023, 12, 1, 0, 0, 0),
136             'attachment' : 'https://github.com',
137             'assignment_course' : self.course}
138
139         response = self.client.post(self.add_assignment_url, data)
140         self.assertEqual(response.status_code, 302)
141
142     #tests for adding announcement
143     def test_add_announcement_POST(self):
144         login = self.client.login(username='aakash', password='aakash123')
145         data = {
146             'user' : self.user_faculty,
147             'course' : self.course,
148             'title': 'Lecture file',
149             'description' : 'PFA materials',
150             'timestamp' : datetime(2023, 12, 1, 0, 0, 0)}
151
152
153         response = self.client.post(self.add_announcement_url, data)
154         self.assertEqual(response.status_code, 302)
155
156     #tests for adding submission
157     def test_add_submission_POST(self):
158         login = self.client.login(username='shrikar', password='shrikar123')
159         data = {
160             'student' : self.student,
161             'work' : 'https://github.com',
162             'title': 'Lecture file',
163             'assignment' : self.assignment,
164             'timestamp' : datetime(2023, 12, 1, 0, 0, 0)}
165
166
167         response = self.client.post(self.add_submission_url, data)
168         self.assertEqual(response.status_code, 302)
169
```

```
170 def test_get_registerPage(self):
171     response = self.client.get(reverse('login_func', args=['student']))
172
173     self.assertEqual(response.status_code, 200)
174     self.assertTemplateUsed(response, 'login_page_student.html')
175
176
177 def test_user_registration(self):
178     data = {
179         'email': 'test@daiict.ac.in',
180         'password': 'abcd1234',
181         'first_name': 'test1',
182         'middle_name': 'test2',
183         'last_name': 'test3',
184         'batch': '1234',
185         'branch': 'test4',
186         'program': 'test5'
187     }
188     response = self.client.post(reverse('otp_ver'), data=data)
189
190     self.assertEqual(response.status_code, 200)
191     self.assertTemplateUsed(response, 'register.html')
192
193 def test_register_existing_email(self):
194     User.objects.create_user(username='test@example.com', password='Password123')
195     data = {
196         'email': 'test@daiict.ac.in',
197         'password': 'abcd1234',
198         'first_name': 'test1',
199         'middle_name': 'test2',
200         'last_name': 'test3',
201         'batch': '1234',
202         'branch': 'test4',
203         'program': 'test5'
204     }
205
206     response = self.client.post(reverse('otp_ver'), data=data)
207
208     self.assertEqual(response.status_code, 200)
209     self.assertTemplateUsed(response, 'register.html')
210
211 def test_log_out(self):
212     self.client.login(username='shrikar@daiict.ac.in', password='shrikar123')
213     response = self.client.get(reverse('log_out'))
214
215     self.assertEqual(response.status_code, 302)
216     self.assertRedirects(response, reverse('HomePage'))
217
218
219 def test_log_out_evenif_logged_Out(self):
220     response = self.client.get(reverse('log_out'))
221
222     self.assertEqual(response.status_code, 302)
223     self.assertRedirects(response, reverse('HomePage'))
224
225 def test_redirect_if_authenticated(self):
226     request = self.factory.get('/otp_ver')
227     request.user = self.user
```

```
228         response = views.verifyRegistration(request)
229         self.assertEqual(response.status_code, 302)
230         self.assertEqual(response.url, '/mycourse')
```

Test Coverage

- **Total Tests:** 19
- **Execution Time:** 16.157s

Summary

The `test_views.py` file focuses on testing the views of the `Academix_Portal` application. Views tested include pages for enrolled courses, assignments, profiles, announcements, materials, queries, and feedback. The tests cover various scenarios such as successful page rendering, handling of post requests, and redirections.

Noteworthy Tests

1. `test_my_course_GET`
 - Verifies the successful rendering of the enrolled courses page for students.
2. `test_view_assignments_faculty_GET`
 - Ensures the assignments view for faculty is rendered correctly.
3. `test_add_assignment_POST`
 - Tests the addition of an assignment by sending a POST request and expecting a successful response.
4. `test_view_feedback_GET`
 - Validates the rendering of the feedback page for students.
5. `test_user_registration`
 - Verifies the user registration process, checking if the registration page is rendered after submitting valid registration data.

Output

```
1 Found 19 test(s).
2 Creating test database for alias 'default'...
3 System check identified no issues (0 silenced).
4 .....
5 -----
6 Ran 19 tests in 16.157s
7
8 OK
9 Destroying test database for alias 'default'...
```

Conclusion

All 19 tests in `test_views.py` passed successfully, indicating the proper functionality of the views in the `Academix_Portal` application.

Overall Conclusion

The unit testing for the `Academix_Portal` application demonstrates the correctness and reliability of both the data models and views. All 49 tests passed without any failures, affirming the application's readiness for further development and deployment.