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)

```
class TestModels(TestCase):
    def setUp(self):
        self.user_student = User.objects.create_user(username='shrikar', password='shrikar123')
        self.user_faculty = User.objects.create_user(username='aakash', password='aakash123')

        self.faculty = faculty_profile.objects.create(
            user=self.user_faculty,
            first_name='aakash',
            middle_name='xyz',
            last_name='patel'
        )

        self.student = student_profile.objects.create(
            user=self.user_student,
            first_name='shrikar',
            middle_name='shaileshbhai',
            last_name='padaliya',
            batch=2022,
            branch='ICT',
            program='B.Tech'
        )

        self.course = Course.objects.create(
            name='Computer Science 101',
            course_code='CS101',
            description='Introduction to Computer Science',
            faculty=self.faculty
        )
           # creating an assignment here
        self.assignment = Assignment.objects.create(
            name='Assignment 1',
            description='First assignment',
            duedate=datetime(2023, 12, 1, 0, 0, 0),
            max_grade=100,
            attachment='aakash_assignment1',
            assignment_course=self.course
        )

        # creating a student submission
        self.submission = Submission.objects.create(
            student=self.student,
            graded=False,
            grade=None,
```

```
            work='shrikar_work1',
            feedback='Good work!',
            assignment=self.assignment
        )

    def test_duplicate_usernames(self):
        # Testing for uniqueness of usernames
        user_duplicate = User(username='shrikar', password='testpassword')
        with self.assertRaises(Exception):
            user_duplicate.save()

    def test_student_enroll_in_course(self):
        # Testing if a student can enroll in a course
        self.course.studentlist.add(self.student)
        self.assertEqual(self.course.studentlist.count(), 1)

    def test_course_with_no_faculty(self):
        # Testing create a course without a faculty

        with self.assertRaises(IntegrityError):
            course_no_faculty = Course.objects.create(
            name='No Faculty Course',
            course_code='NF101',
            description='Course without faculty'
        )

    def test_course_without_students(self):
        # Testing create a course without students
        course_empty = Course.objects.create(
            name='Empty Course',
            course_code='EC101',
            description='Course with no students',
            faculty=self.faculty
        )
        self.assertEqual(course_empty.studentlist.count(), 0)

    def test_student_courses_relationship(self):
        # Testing relationship between student and courses
        self.student.student_courses.add(self.course)
        self.assertEqual(self.student.student_courses.count(), 1)
        self.assertEqual(self.student.student_courses.first().name, 'Computer Science 101')

    def test_faculty_courses_relationship(self):
        # Testing relationship between faculty and courses
        self.faculty.faculty_courses.add(self.course)
        self.assertEqual(self.faculty.faculty_courses.count(), 1)
        self.assertEqual(self.faculty.faculty_courses.first().name, 'Computer Science 101')

    def test_course_students_and_faculty(self):
        # Testing that a course has both students and a faculty
        self.course.studentlist.add(self.student)
        self.assertEqual(self.course.studentlist.count(), 1)
        self.assertEqual(self.course.faculty, self.faculty)

    def test_student_str_method(self):

        self.assertEqual(str(self.student), 'shrikar')

```

```
    def test_faculty_str_method(self):

        self.assertEqual(str(self.faculty), 'aakash')

    def test_course_str_method(self):

        self.assertEqual(str(self.course), 'Computer Science 101')


    def test_assignment_str_method(self):
        # Testing the __str__ method of Assignment
        self.assertEqual(str(self.assignment), 'Assignment 1')

    def test_submission_str_method(self):
        # Testing the __str__ method of Submission
        expected_str = f'{self.assignment.assignment_course.course_code} shrikarpadaliya'
        self.assertEqual(str(self.submission), expected_str)

    def test_submission_defaults(self):
        # Testing submission default values
        # Create a new assignment to avoid unique constraint violation
        new_assignment = Assignment.objects.create(
            name='Assignment 2',
            description='Second assignment',
            duedate=datetime(2023, 12, 2, 0, 0, 0),
            max_grade=90,
            attachment='http://example.com/assignment2',
            assignment_course=self.course
        )
        # Use a different student to avoid unique constraint violation
        new_student = student_profile.objects.create(
            user=User.objects.create_user(username='mustafa', password='mustafa123'),
            first_name='mustafa',
            last_name='lokhandwala',
            batch=2022,
            branch='CSE',
            program='B.Tech'
        )

        submission_defaults = Submission(student=new_student, assignment=new_assignment)
        submission_defaults.save()
        self.assertFalse(submission_defaults.graded)
        self.assertIsNone(submission_defaults.grade)
        self.assertIsNone(submission_defaults.feedback)
        self.assertIsNotNone(submission_defaults.timestamp)  # Ensure timestamp is not None

    def test_assignment_due_date(self):
        # Testing assignment due date
        self.assertEqual(self.assignment.duedate, datetime(2023, 12, 1, 0, 0, 0))

    def test_submission_timestamp_auto_now_add(self):
        # Testing that the timestamp in Submission is set automatically on creation
        new_assignment = Assignment.objects.create(
            name='Assignment 2',
            description='Second assignment',
            duedate=datetime(2023, 12, 2, 0, 0, 0),
            max_grade=90,
            attachment='http://example.com/assignment2',
```

```
            assignment_course=self.course
        )
        # Use a different student to avoid unique constraint violation
        new_student = student_profile.objects.create(
            user=User.objects.create_user(username='mustafa', password='mustafa123'),
            first_name='mustafa',
            last_name='lokhandwala',
            batch=2022,
            branch='CSE',
            program='B.Tech'
        )
        submission_timestamp = Submission.objects.create(
            student=new_student,
            graded=False,
            grade=None,
            work='shrikar_work',
            feedback='Great job!',
            assignment=new_assignment
        )
        self.assertIsNotNone(submission_timestamp.timestamp)

    def test_submission_grade_update(self):
        # Testing updating the grade in Submission
        updated_grade = 95
        self.submission.grade = updated_grade
        self.submission.save()
        self.assertEqual(Submission.objects.get(id=self.submission.id).grade, updated_grade)

    def test_submission_feedback_update(self):
        # Testing updating the feedback in Submission
        updated_feedback = 'Excellent work!'
        self.submission.feedback = updated_feedback
        self.submission.save()
        self.assertEqual(Submission.objects.get(id=self.submission.id).feedback, updated_feedback)

    def test_assignment_course_relationship(self):
        # Testing relationship between Assignment and Course
        self.assertEqual(self.assignment.assignment_course, self.course)

    def test_submission_assignment_relationship(self):
        # Testing relationship between Submission and Assignment
        self.assertEqual(self.submission.assignment, self.assignment)

    def test_assignment_submission_relationship(self):
        # Testing relationship between Assignment and Submission
        assignment_submissions = self.assignment.submission_set.all()
        self.assertEqual(assignment_submissions.count(), 1)
        self.assertEqual(assignment_submissions.first(), self.submission)

    def test_duplicate_assignment_name(self):
        # Testing for uniqueness of assignment names within a course
        Assignment.objects.create(
            name='Assignment temp',
            description='First assignment',
            duedate=datetime(2023, 12, 1, 0, 0, 0),
            max_grade=100,
            attachment='aakash_assignment1',
            assignment_course=self.course
```

```
        )

        with self.assertRaises(IntegrityError):
            Assignment.objects.create(
                name='Assignment temp',
                description='Duplicate assignment name',
                duedate=datetime(2023, 12, 2, 0, 0, 0),
                max_grade=90,
                attachment='aakash_assignment1',
                assignment_course=self.course
            )

    def test_submission_student_assignment_uniqueness(self):
        # Testing for uniqueness of submissions by student for a specific assignment
        assignment = Assignment.objects.create(
            name='Assignment 2',
            description='Second assignment',
            duedate=datetime(2023, 12, 2, 0, 0, 0),
            max_grade=90,
            attachment='aakash_assignment2',
            assignment_course=self.course
        )
        Submission.objects.create(
            student=self.student,
            graded=False,
            grade=None,
            work='shrikar_work2',
            feedback='Good work!',
            assignment=assignment
        )
        with self.assertRaises(IntegrityError):
            Submission.objects.create(
                student=self.student,
                graded=False,
                grade=None,
                work='shrikar_work2',
                feedback='Another submission',
                assignment=assignment
            )

    def test_assignment_course_foreign_key_constraint(self):
        # Testing integrity constraint for assignment_course foreign key
        with self.assertRaises(IntegrityError):
            Assignment.objects.create(
                name='Assignment 3',
                description='Third assignment',
                duedate=datetime(2023, 12, 3, 0, 0, 0),
                max_grade=80,
                attachment='aakash_assignment3',
                assignment_course=None
            )

    def test_submission_assignment_foreign_key_constraint(self):
        # Testing integrity constraint for submission assignment foreign key
        with self.assertRaises(IntegrityError):
            Submission.objects.create(
                student=self.student,
                graded=False,
```

```
                grade=None,
                work='http://example.com/submission3',
                feedback='Yet another submission',
                assignment=None
            )

    def test_query_creation(self):
        # Testing creation of a query
        query_instance = query.objects.create(
            course=self.course,
            user=self.student,
            qry='Doubt about the lecture content'
        )
        self.assertEqual(str(query_instance), 'CS101 shrikar padaliya')
        self.assertIsNotNone(query_instance.timestamp)
        self.assertEqual(query_instance.qry, 'Doubt about the lecture content')
        self.assertIsNone(query_instance.reply)

    def test_query_reply(self):
        # Testing replying to a query
        query_instance = query.objects.create(
            course=self.course,
            user=self.student,
            qry='Doubt about the lecture content'
        )
        reply_text = 'This is a reply.'
        query_instance.reply = reply_text
        query_instance.save()
        self.assertEqual(query_instance.reply, reply_text)

    def test_query_defaults(self):
        # Testing default values of a query
        query_instance = query.objects.create(
            course=self.course,
            user=self.student,
            qry='Doubt about the lecture content'
        )
        self.assertIsNone(query_instance.reply)

    def test_query_course_foreign_key_constraint(self):
        # Testing integrity constraint for query course foreign key
        with self.assertRaises(IntegrityError):
            query.objects.create(
                course=None,
                user=self.student,
                qry='Doubt about the lecture content'
            )

    def test_query_user_foreign_key_constraint(self):
        # Testing integrity constraint for query user foreign key
        with self.assertRaises(IntegrityError):
            query.objects.create(
                course=self.course,
                user=None,
                qry='Doubt about the lecture content'
            )
```

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

```
Found 30 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..........
----------------------------------------------------------------------
Ran 30 tests in 10.698s

OK
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)

```
class TestViews(TestCase):

    def setUp(self):
        self.factory = RequestFactory()
        self.client = Client()
        self.user_student = User.objects.create_user(email='shrikar@daiict.ac.in', username='shrikar', password
        self.user_faculty = User.objects.create_user(email='aakash@daiict.ac.in', username='aakash', password='
        self.user = User.objects.create_user(username='testuser', password='password')

        self.faculty = faculty_profile.objects.create(
            user=self.user_faculty,
            first_name='aakash',
            middle_name='xyz',
            last_name='patel'
        )

        self.student = student_profile.objects.create(
            user=self.user_student,
            first_name='shrikar',
            middle_name='shaileshbhai',
            last_name='padaliya',
            batch=2022,
            branch='ICT',
            program='B.Tech'
        )

        self.course = Course.objects.create(
            name='Computer Science 101',
            course_code='CS101',
            description='Introduction to Computer Science',
            faculty=self.faculty
        )

        self.assignment = Assignment.objects.create(
            name='Assignment 1',
            description='First assignment',
            duedate=datetime(2023, 12, 1, 0, 0, 0),
            max_grade=100,
            attachment='aakash_assignment1',
            assignment_course=self.course
        )


        self.my_course_url = reverse('mycourse')
        self.otp_ver = reverse('otp_ver')
        self.view_assignments_url = reverse('view_assignments', args=['CS101'])
        self.profile_url = reverse('students_profile')
        self.view_announcements_url = reverse('announcements', args=['CS101'])
        self.view_materials_url = reverse('materials', args=['CS101'])
        self.view_query_url = reverse('view_query', args=['CS101'])
        self.view_feedback_url = reverse('feedback', args=['CS101'])
        self.view_student_list_url = reverse('student_list', args=['CS101'])
        self.add_assignment_url = reverse('add_assignment', args=['CS101'])
```

```
        self.add_announcement_url = reverse('addannouncement', args=['CS101'])
        self.add_submission_url = reverse('add_submission', args=['CS101', 'Assignment 1'])

    # testing view enrolled courses page
    def test_my_course_GET(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        response = self.client.get(self.my_course_url)
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'my_course_student.html')
        self.assertTemplateUsed(response, 'base.html')

    # tests for view assignment page faculty side
    def test_view_assignments_faculty_GET(self):
        login = self.client.login(username='aakash', password='aakash123')
        response = self.client.get(self.view_assignments_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'view_assignments_faculty.html')

    #tests for view assignment page student side
    def test_view_assignments_GET(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        response = self.client.get(self.view_assignments_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'view_assignments.html')
        self.assertTemplateUsed(response, 'navbar.html')

    #tests for view profile page
    def test_view_profile_GET(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        response = self.client.get(self.profile_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'student_profile.html')

    #tests for view announcements page
    def test_view_announcements_GET(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        response = self.client.get(self.view_announcements_url)
        self.assertEqual(response.status_code, 302)

    #tests for view materials page
    def test_view_materials_GET(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        response = self.client.get(self.view_materials_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'materials.html')

    #tests for view query page
    def test_view_query_GET(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        response = self.client.get(self.view_query_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'view_query.html')

    #tests for view feedback page student side
    def test_view_feedback_GET(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        response = self.client.get(self.view_feedback_url)
        self.assertEquals(response.status_code, 200)
```

```
        self.assertTemplateUsed(response, 'add_feedback.html')

    #tests for view feedback page faculty side
    def test_view_feedback_faculty_GET(self):
        login = self.client.login(username='aakash', password='aakash123')
        response = self.client.get(self.view_feedback_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'feedback_faculty.html')

    #tests for view enrolled student list
    def test_view_student_list_GET(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        response = self.client.get(self.view_student_list_url)
        self.assertEquals(response.status_code, 200)
        self.assertTemplateUsed(response, 'student_list.html')

    # tests for adding assignment
    def test_add_assignment_POST(self):
        login = self.client.login(username='aakash', password='aakash123')
        data = {
            'name' : 'Assignment 2',
            'max_grade': '100',
            'description' : 'Second assignment',
            'duedate' : datetime(2023, 12, 1, 0, 0, 0),
            'attachment' : 'https://github.com',
            'assignment_course' : self.course}

        response = self.client.post(self.add_assignment_url, data)
        self.assertEquals(response.status_code, 302)

    #tests for adding announcement
    def test_add_announcement_POST(self):
        login = self.client.login(username='aakash', password='aakash123')
        data = {
            'user' : self.user_faculty,
            'course' : self.course,
            'title': 'Lecture file',
            'description' : 'PFA materials',
            'timestamp' :  datetime(2023, 12, 1, 0, 0, 0)}


        response = self.client.post(self.add_announcement_url, data)
        self.assertEquals(response.status_code, 302)

    #tests for adding submission
    def test_add_submission_POST(self):
        login = self.client.login(username='shrikar', password='shrikar123')
        data = {
            'student' : self.student,
            'work' : 'https://github.com',
            'title': 'Lecture file',
            'assignment' : self.assignment,
            'timestamp' :  datetime(2023, 12, 1, 0, 0, 0)}


        response = self.client.post(self.add_submission_url, data)
        self.assertEquals(response.status_code, 302)

```

```
    def test_get_registerPage(self):
        response = self.client.get(reverse('login_func',args=['student']))

        self.assertEqual(response.status_code,200)
        self.assertTemplateUsed(response, 'login_page_student.html')


    def test_user_registration(self):
        data = {
            'email':'test@daiict.ac.in',
            'password':'abcd1234',
            'first_name':'test1',
            'middle_name':'test2',
            'last_name':'test3',
            'batch':'1234',
            'branch':'test4',
            'program':'test5'
        }
        response = self.client.post(reverse('otp_ver'), data=data)

        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'register.html')

    def test_register_existing_email(self):
        User.objects.create_user(username='test@example.com', password='Password123')
        data = {
            'email':'test@daiict.ac.in',
            'password':'abcd1234',
            'first_name':'test1',
            'middle_name':'test2',
            'last_name':'test3',
            'batch':'1234',
            'branch':'test4',
            'program':'test5'
        }

        response = self.client.post(reverse('otp_ver'), data=data)

        self.assertEqual(response.status_code,200)
        self.assertTemplateUsed(response, 'register.html')

    def test_log_out(self):
        self.client.login(username='shrikar@daiict.ac.in', password='shrikar123')
        response = self.client.get(reverse('log_out'))

        self.assertEqual(response.status_code,302)
        self.assertRedirects(response, reverse('HomePage'))


    def test_log_out_evenif_logged_Out(self):
        response = self.client.get(reverse('log_out'))

        self.assertEqual(response.status_code,302)
        self.assertRedirects(response, reverse('HomePage'))

    def test_redirect_if_authenticated(self):
            request = self.factory.get('/otp_ver')
            request.user = self.user
```

```
            response = views.verifyRegistration(request)
            self.assertEqual(response.status_code, 302)
            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

```
Found 19 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.............
----------------------------------------------------------------------
Ran 19 tests in 16.157s

OK
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.