UMass Roommate Finder Application Evaluation Plan

Authors: Anurati Bhosekar, Noah Dixon, Sijia Hao, Dishank Deven Jhaveri

UI Tests:

The following functional requirements were evaluated manually as shown in the video recording “Final_Recording.mp4” to ensure correctness from a user standpoint:

User Creation: Ensure the successful creation of a user account and profile.

User Verification: Ensure that only users with umass.edu email domains are allowed during signup.

Roommate Filtering: Ensure that roommate preference filters allow users to set preferences for a roommate and update their profile search results.

Log out/Sign in: Ensure that a user is able to log out of or sign in to an existing account.

Request Roommate Contact: Ensure that a user is able to send a contact request to another user.

Request Roommate Response: Ensure that a user is able to respond to a contact request and see the requester's information.

Account Deletion: Ensure that an admin user is able to delete a users’ account, and that the deleted profile will no longer appear in the search results or the contact page.

Test Suite Tests:

The following tests were run in the web app and register test suites to ensure correctness on the backend of the application.

1. Test User Creation and Profile Form:
   This test case ensures that users can successfully create accounts and profiles. It tests if the form for the profile is valid and saves the data correctly. This test is in the register app (found in register/tests)

```
def test_profile_form_valid(self):
    form = ProfileForm(data={'gender': 'M', 'year': '2023', 'college': 'Engineering'})
    self.assertTrue(form.is_valid())
```

2. Test User Verification:
This test case ensures that only users with a "@umass.edu" email domain can register on the platform. This test is in the register app (found in register/tests)

```
def test_form_valid_email(self):
    form = RegisterForm(data={"email": "test@umass.edu"})
    self.assertTrue(form.is_valid())
```

3. Test User Profile Page:
This test case checks if the profile page displays the correct user profile information, and if the profile page can be edited successfully. This test can be found in the webappl app (webappl/tests)

```
def test_profile_page(self):
    self.client.login(username='testuser1', password='1X<ISRUkw+tuK')
    response = self.client.get('/profile_page/')
    self.assertEqual(response.status_code, 200)
```

4. Test Roommate Matching:
This test case verifies that the system updates the list of matched roommates according to the user's preferences. This test can be found in the webappl app (webappl/tests)

```
def test_search_page(self):
    self.client.login(username='testuser1', password='1X<ISRUkw+tuK')
    response = self.client.get('/search_page/')
    self.assertEqual(response.status_code, 200)
```

5. Test Search Filters:
This test case tests whether users can filter potential roommates by various fields such as gender, level_of_study, and program. This test can be found in the webappl app (webappl/tests)

```
def test_profile_filter(self):
    filter = ProfileFilter({'gender': 'M'}, queryset=Profile.objects.all())
    self.assertIsNotNone(filter)
```

6. Test URL Routing:
This test case ensures that URL routing in the application works as expected, meaning that each URL leads to the appropriate view. This test can be found in the webappl app (webappl/tests)

```
def test_url_resolves(self):
    url = reverse("home_page")
    self.assertEquals(resolve(url).func, home_page)
```

7. Test Accept Contact Request:
This test case checks if a contact request can be successfully accepted. This test can be found in the webappl app (webappl/tests)

```
def test_accept_contact(self):
    self.client.login(username='testuser1', password='1X<ISRUkw+tuK')
    response = self.client.post(reverse('accept_contact', args=[self.request1.id]))
    self.assertEqual(response.status_code, 302)
```

8. Test Clean Email Method:
This test case verifies that the clean_email method in RegisterForm works correctly by ensuring the email field contains "@umass.edu". This test is in the register app (found in register/tests)

```
def test_clean_email(self):
    form = RegisterForm(data={"email": "test@umass.edu"})
    self.assertTrue(form.is_valid())
    self.assertEqual(form.clean_email(), "test@umass.edu")
```

These test cases cover a wide variety of functional aspects of the application to ensure it is working as expected.

9. Test User Authentication:
This test case ensures that users are correctly authenticated when they log in, and unauthenticated users are redirected to the login page when they try to access a restricted page. This test is in the webappl (webapp/tests)

```
def test_user_authentication(self):
    self.client.login(username='testuser1', password='1X<ISRUkw+tuK')
    response = self.client.get('/profile_page/')
    self.assertEqual(response.status_code, 200)
```

Sure, here's a summary for a test case for password matching:

10. Test Password Matching:
This test case verifies that the password and password confirmation fields match during the user registration process. If they do not match, the system should display an error. This test is in the register app (found in register/tests)

```
def test_password_matching(self):
    response = self.client.post('/register/', {
        'first_name': 'Test',
        'last_name': 'User',
        'username': 'testuser1',
        'email': 'testuser1@umass.edu',
        'password1': '1X<ISRUkw+tuK',
        'password2': 'differentpassword'
    })
    self.assertFormError(response, 'form', 'password2', "The two password fields didn’t match.")
```

In this test case, the user tries to register with two different passwords. The test checks if Django's built-in UserCreationForm correctly catches this and returns an error message indicating that the two password fields do not match. The test is successful if the expected error message is present in the form errors.