```python
# Source: Oppia - https://github.com/oppia/oppia - an online learning tool that enables anyone to easily create and share interactive activities

"""Tests for the GAE mail API wrapper."""

from __future__ import absolute_import  # pylint: disable=import-only-modules
from __future__ import unicode_literals  # pylint: disable=import-only-modules

from core.platform.email import gae_email_services
from core.tests import test_utils
import feconf


class EmailTests(test_utils.GenericTestBase):
    """Tests for sending emails."""
    RECIPIENT_EMAIL = 'user@example.com'
    RECIPIENT_USERNAME = 'user'
    SENDER_EMAIL = 'Sender <sender@example.com>'

    def test_sending_email(self):
        # Emails are not sent if the CAN_SEND_EMAILS setting is not turned on.
        email_exception = self.assertRaisesRegexp(
            Exception, 'This app cannot send emails.')
        messages = self.mail_stub.get_sent_messages(
            to=feconf.ADMIN_EMAIL_ADDRESS)
        self.assertEqual(0, len(messages))
        with self.swap(feconf, 'CAN_SEND_EMAILS', False), email_exception:
            gae_email_services.send_mail(
                self.SENDER_EMAIL, feconf.ADMIN_EMAIL_ADDRESS,
                'subject', 'body', 'html', bcc_admin=False)
        messages = self.mail_stub.get_sent_messages(
            to=feconf.ADMIN_EMAIL_ADDRESS)
        self.assertEqual(0, len(messages))

        messages = self.mail_stub.get_sent_messages(
            to=feconf.ADMIN_EMAIL_ADDRESS)
        self.assertEqual(0, len(messages))
        with self.swap(feconf, 'CAN_SEND_EMAILS', True):
            gae_email_services.send_mail(
                self.SENDER_EMAIL, feconf.ADMIN_EMAIL_ADDRESS,
                'subject', 'body', 'html', bcc_admin=False)
        messages = self.mail_stub.get_sent_messages(
            to=feconf.ADMIN_EMAIL_ADDRESS)
        self.assertEqual(1, len(messages))

    def test_email_bcc_is_sent_if_bcc_admin_is_true(self):
        # Tests that email has ADMIN_EMAIL_ADDRESS as bcc if bcc_admin is True.
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))
        with self.swap(feconf, 'CAN_SEND_EMAILS', True):
            gae_email_services.send_mail(
                self.SENDER_EMAIL, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html', bcc_admin=True)
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(1, len(messages))
        self.assertEqual(messages[0].bcc, feconf.ADMIN_EMAIL_ADDRESS)

    def test_email_bcc_not_sent_if_bcc_admin_is_false(self):
        # Tests that email has no bcc attribute if bcc_admin is False.
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))
        with self.swap(feconf, 'CAN_SEND_EMAILS', True):
            gae_email_services.send_mail(
                self.SENDER_EMAIL, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html', bcc_admin=False)
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(1, len(messages))
        self.assertFalse(hasattr(messages[0], 'bcc'))

    def test_sending_email_with_reply_to_id_adds_reply_to_email(self):
        # Tests that email has reply_to_address if reply_to_id is passed.
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))

        reply_to_id = 'reply_to_id'
        with self.swap(feconf, 'CAN_SEND_EMAILS', True):
            gae_email_services.send_mail(
                self.SENDER_EMAIL, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html', reply_to_id=reply_to_id)
        messages = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAIL)
        expected_reply_to_address = 'reply+reply_to_id@example.com'
        self.assertEqual(messages[0].reply_to, expected_reply_to_address)

    def test_sending_email_without_reply_to_id_not_add_reply_to_email(self):
        """Tests that email does not have reply_to_address if reply_to_id is
        not passed.
        """
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))
        with self.swap(feconf, 'CAN_SEND_EMAILS', True):
            gae_email_services.send_mail(
                self.SENDER_EMAIL, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html')
        messages = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAIL)
        self.assertFalse(hasattr(messages[0], 'reply_to'))

    def test_email_not_sent_if_sender_address_is_malformed(self):
        # Tests that email is not send if sender email address is malformed.

        # Case when malformed_sender_email is None.
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))
        malformed_sender_email = None
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_mail(
                malformed_sender_email, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html')
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))

        # Case when malformed_sender_email is an empty string.
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))
        malformed_sender_email = ''
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_mail(
                malformed_sender_email, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html')
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))

        # Case when malformed_sender_email does not have name of the sender,
        # only the email address.
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))
        malformed_sender_email = '<malformed_sender_email@example.com>'
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_mail(
                malformed_sender_email, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html')
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))

        # Case when malformed_sender_email does not have email address of the
        # sender and only the name.
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))
        malformed_sender_email = 'MalfomedSender'
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_mail(
                malformed_sender_email, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html')
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))

        # Case when malformed_sender_email does not have email address of the
        # form '<sender_email_address>'.
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))
        malformed_sender_email = 'MalformedSender malformed_sender@gmail.com'
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_mail(
                malformed_sender_email, self.RECIPIENT_EMAIL,
                'subject', 'body', 'html')
        messages = self.mail_stub.get_sent_messages(to=self.RECIPIENT_EMAIL)
        self.assertEqual(0, len(messages))

    def test_email_not_sent_if_recipient_address_is_malformed(self):
        # Tests that email is not sent if recipient email address is
malformed.
```

```

        # Case when malformed_recipient_email is None.
        malformed_recipient_email = None
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed recipient email address: %s'
            % malformed_recipient_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_mail(
                self.SENDER_EMAIL, malformed_recipient_email,
                'subject', 'body', 'html')

        # Case when malformed_recipient_email is an empty string.
        malformed_recipient_email = ''
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed recipient email address: %s'
            % malformed_recipient_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_mail(
                self.SENDER_EMAIL, malformed_recipient_email,
                'subject', 'body', 'html')


class BulkEmailsTests(test_utils.GenericTestBase):
    SENDER_EMAIL = 'Sender <sender@example.com>'
    SENDER_USERNAME = 'sender'
    RECIPIENT_A_EMAIL = 'a@example.com'
    RECIPIENT_A_USERNAME = 'usera'
    RECIPIENT_B_EMAIL = 'b@example.com'
    RECIPIENT_B_USERNAME = 'userb'
    RECIPIENT_EMAILS = [RECIPIENT_A_EMAIL, RECIPIENT_B_EMAIL]

    def test_correct_bulk_emails_sent(self):
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(len(message_a), 0)
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(len(message_b), 0)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True):
            gae_email_services.send_bulk_mail(
                self.SENDER_EMAIL, self.RECIPIENT_EMAILS,
                'subject', 'body', 'html')
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(len(message_a), 1)
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(len(message_b), 1)

    def test_bulk_emails_not_sent_if_can_send_emails_is_false(self):
        # Emails are not sent if the CAN_SEND_EMAILS setting is not turned
on.
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(len(message_a), 0)
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(len(message_b), 0)
        email_exception = self.assertRaisesRegexp(
            Exception, 'This app cannot send emails.')
```

```python
        with email_exception:
            gae_email_services.send_bulk_mail(
                self.SENDER_EMAIL, self.RECIPIENT_EMAILS,
                'subject', 'body', 'html')
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(len(message_a), 0)
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(len(message_b), 0)

    def test_bulk_mails_not_sent_if_sender_email_is_malformed(self):
        """Tests that bulk emails are not sent if sender email address is
        malformed.
        """

        # Case when malformed_sender_email is None.
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))
        malformed_sender_email = None
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_bulk_mail(
                malformed_sender_email, self.RECIPIENT_EMAILS,
                'subject', 'body', 'html')
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))

        # Case when malformed_sender_email is an empty string.
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))
        malformed_sender_email = ''
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_bulk_mail(
                malformed_sender_email, self.RECIPIENT_EMAILS,
                'subject', 'body', 'html')
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))

        # Case when malformed_sender_email does not have name of the sender
        # and only the email address.
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))
        malformed_sender_email = '<malformed_sender_email@example.com>'
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_bulk_mail(
                malformed_sender_email, self.RECIPIENT_EMAILS,
                'subject', 'body', 'html')
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))

        # Case when malformed_sender_email does not have email address of the
        # sender and only the name.
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))
        malformed_sender_email = 'MalformedSender'
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_bulk_mail(
                malformed_sender_email, self.RECIPIENT_EMAILS,
                'subject', 'body', 'html')
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))

        # Case when malformed_sender_email does not have email address of the
        # form '<sender_email_address>'.
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))
        malformed_sender_email = 'Malformed Sender
malformed_sender@example.com'
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed sender email address: %s'
            % malformed_sender_email)
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
```

```python
            gae_email_services.send_bulk_mail(
                malformed_sender_email, self.RECIPIENT_EMAILS,
                'subject', 'body', 'html')
        message_a = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[0])
        self.assertEqual(0, len(message_a))
        message_b = self.mail_stub.get_sent_messages(
            to=self.RECIPIENT_EMAILS[1])
        self.assertEqual(0, len(message_b))

    def test_bulk_mails_not_sent_if_recipient_email_is_malformed(self):
        """Tests that bulk emails are not sent if recipient email address is
        malformed.
        """

        # Case when both recipient email address strings are None.
        malformed_recipient_emails = [None, None]
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed recipient email address: %s'
            % malformed_recipient_emails[0])
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_bulk_mail(
                self.SENDER_EMAIL, malformed_recipient_emails,
                'subject', 'body', 'html')
        messages_a = self.mail_stub.get_sent_messages(
            to=malformed_recipient_emails[0])
        self.assertEqual(len(messages_a), 0)
        messages_b = self.mail_stub.get_sent_messages(
            to=malformed_recipient_emails[1])
        self.assertEqual(len(messages_b), 0)

        # Case when both recipient email address strings are empty.
        malformed_recipient_emails = ['', '']
        email_exception = self.assertRaisesRegexp(
            ValueError, 'Malformed recipient email address: %s'
            % malformed_recipient_emails[0])
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_bulk_mail(
                self.SENDER_EMAIL, malformed_recipient_emails,
                'subject', 'body', 'html')
        messages_a = self.mail_stub.get_sent_messages(
            to=malformed_recipient_emails[0])
        self.assertEqual(len(messages_a), 0)
        messages_b = self.mail_stub.get_sent_messages(
            to=malformed_recipient_emails[1])
        self.assertEqual(len(messages_b), 0)

        # Case when one of the recipient email address strings is None.
        malformed_recipient_emails = [self.RECIPIENT_A_EMAIL, None]
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_bulk_mail(
                self.SENDER_EMAIL, malformed_recipient_emails,
                'subject', 'body', 'html')
        messages_a = self.mail_stub.get_sent_messages(
            to=malformed_recipient_emails[0])
        self.assertEqual(len(messages_a), 0)
        messages_b = self.mail_stub.get_sent_messages(
            to=malformed_recipient_emails[1])
        self.assertEqual(len(messages_b), 0)

        # Case when one of the recipient email address strings is empty.
        malformed_recipient_emails = [self.RECIPIENT_A_EMAIL, '']
        with self.swap(feconf, 'CAN_SEND_EMAILS', True), email_exception:
            gae_email_services.send_bulk_mail(
                self.SENDER_EMAIL, malformed_recipient_emails,
                'subject', 'body', 'html')
        messages_a = self.mail_stub.get_sent_messages(
            to=malformed_recipient_emails[0])
        self.assertEqual(len(messages_a), 0)
        messages_b = self.mail_stub.get_sent_messages(
            to=malformed_recipient_emails[1])
        self.assertEqual(len(messages_b), 0)

```