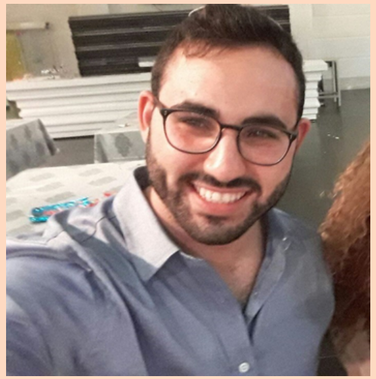
High.Teach

Demo 2

Contents:

The team:

David Cohen



Idan Rosenberg

Rany Whabi

Yarden Gazit

Shiraz Yom Tov

Shachar Levy

ERD

Study_Group		
PK	study_group_id	int
FK	group_owner	int
	field	varchar(30)
	group_description	textfield
	capacity	int

Group_Member		
FK	group_id	int
FK	private_id	int

Django user		
PK	user_id	int
	username	varchar(20)
	password	varchar(20)
	first_name	varchar(20)
	last_name	varchar(20)
	email	varchar(20)

Profile		
FK	user_id	int
	phone_number	varchar(20)
	city	location
	meeting_method	enum
	account_type	enum

Message		
PK	message_id	int
FK	sender_id	int
FK	student_course_id	int
FK	group_id	int
	date_time	date
	message	varchar(255)

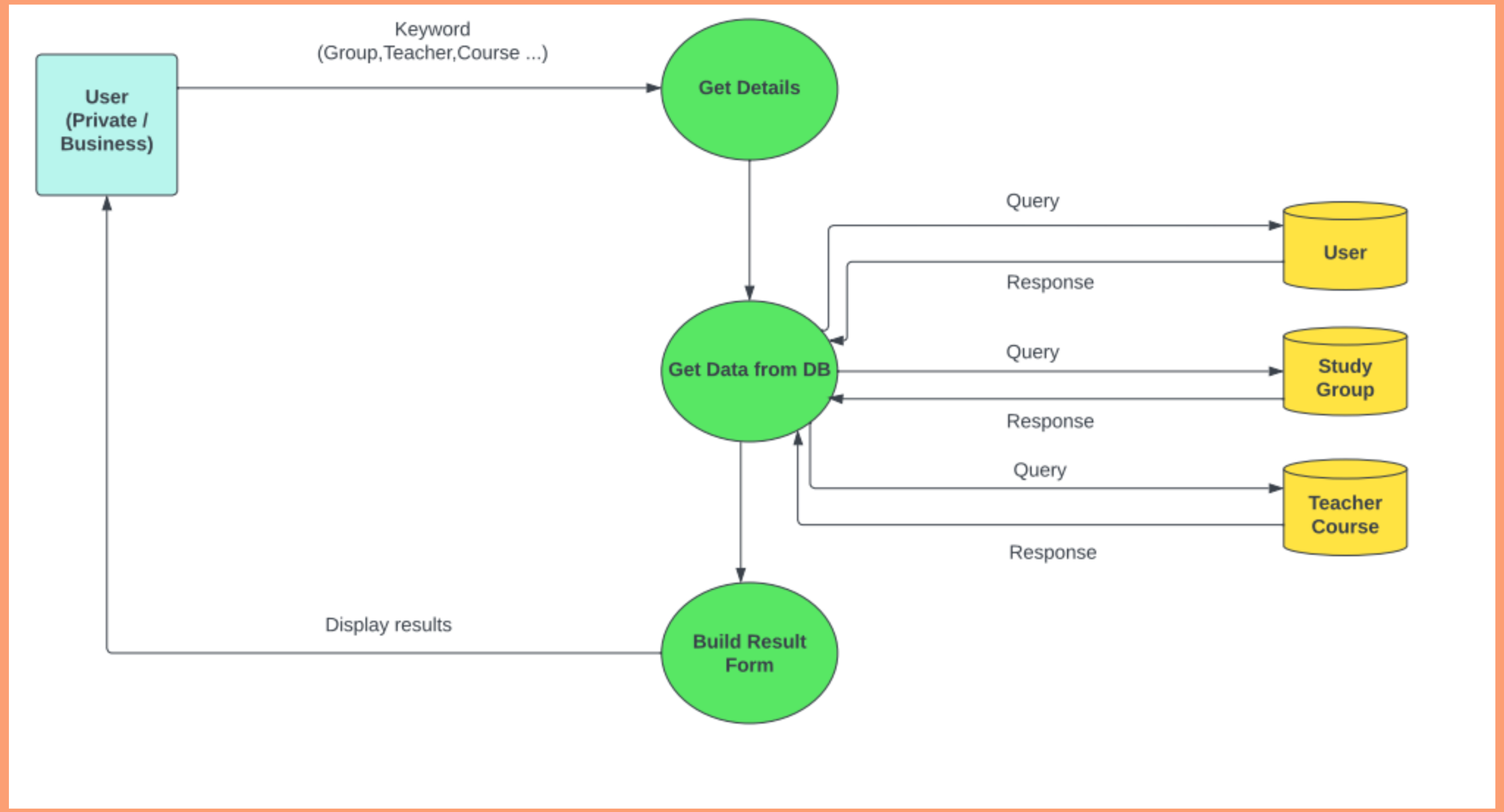
Post		
PK	post_id	int
FK	course_id	int
FK	user_id	int
	parent_post_id	int
	date	date
	content	varchar(100)
	post_image	image

Review		
PK	review_id	int
FK	student_id	int
FK	course_id	int
	rating	enum
	content	text
	date	date

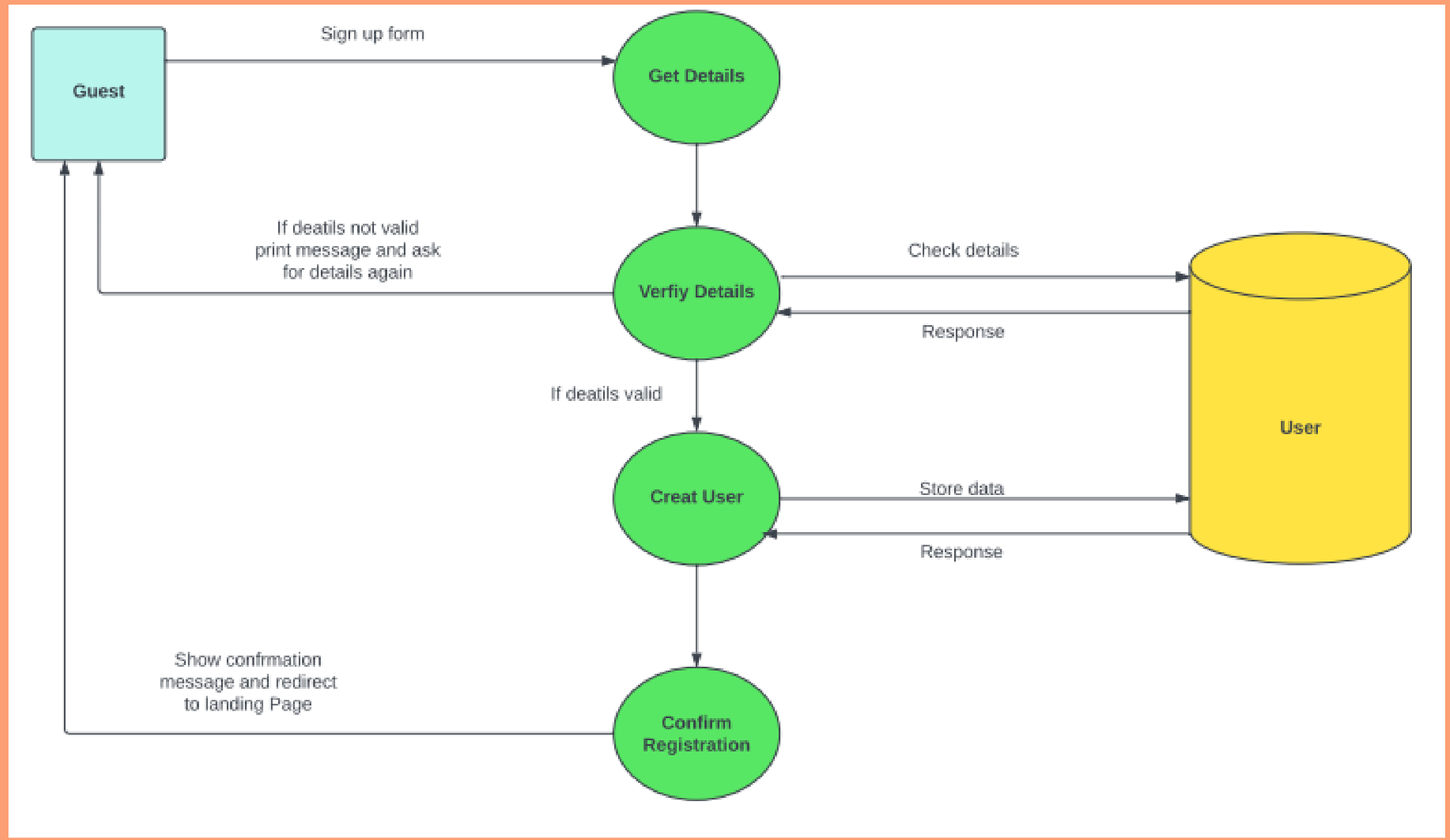
Student_Course		
PK	student_course_id	int
FK	student_id	int
FK	course_id	int
	status	enum
	request_date	date

Teacher_Course		
PK	course_id	int
FK	teacher_id	int
	course_name	varchar(20)
	description	text
	category	enum
	years_of_experience	enum
	difficulty_level	enum
	price	int

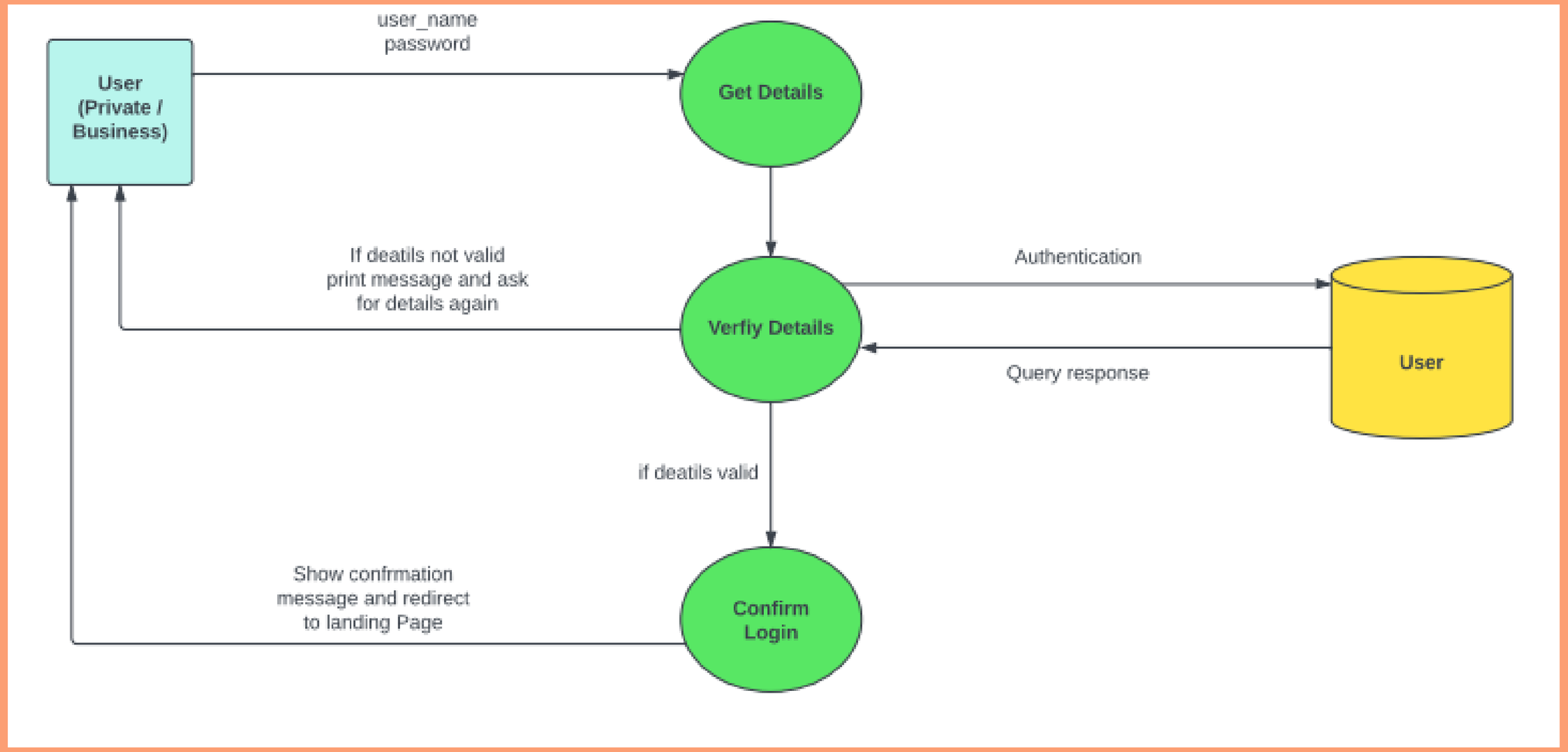
DFD - Search


Keyword
(Group,Teacher,Course ...)
User
(Private /
Business)
Get Details
Query
User
Response
Get Data from DB
Query
Study
Group
Response
Query
Teacher
Course
Response
Display results
Build Result
Form

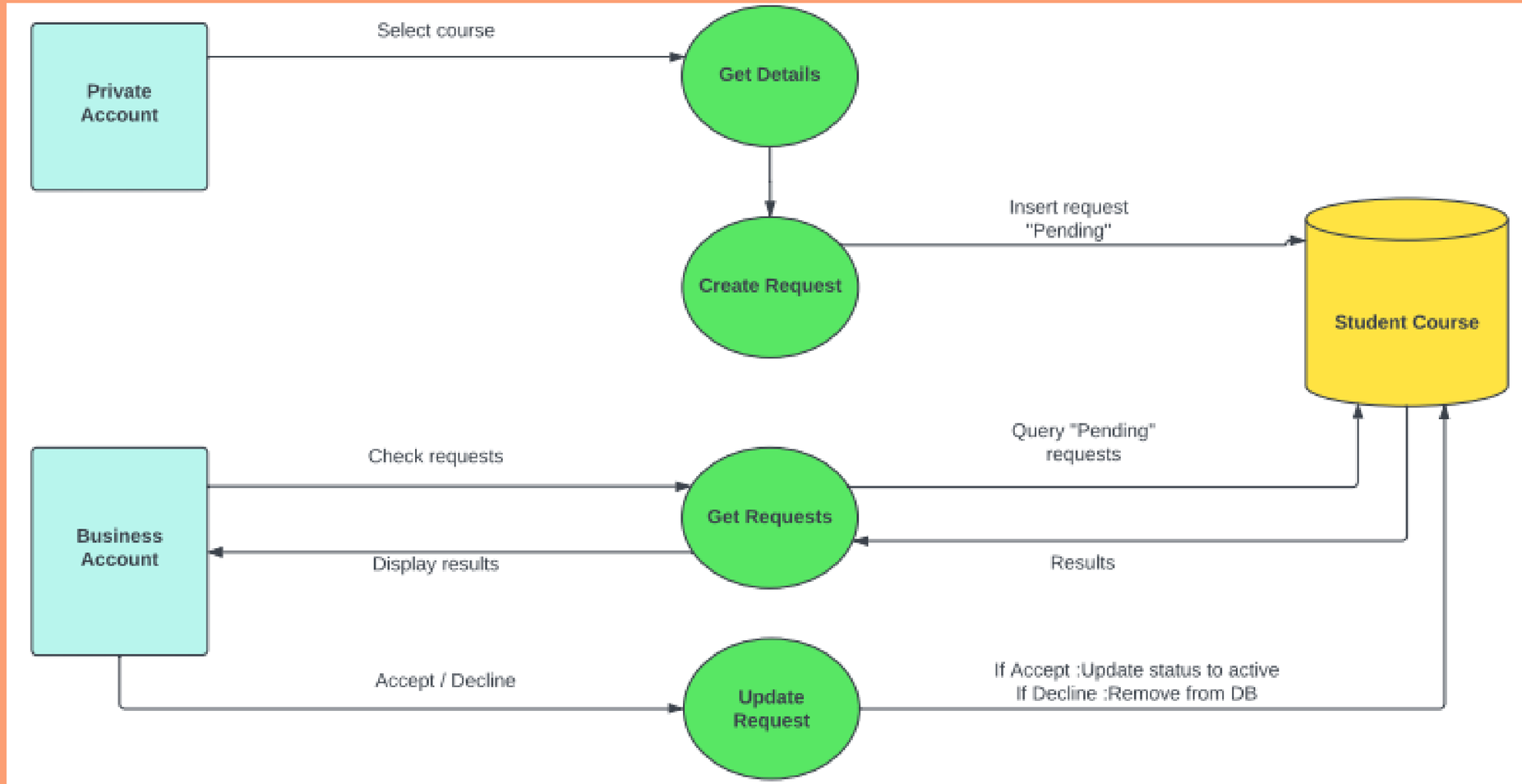
DFD - Signing up



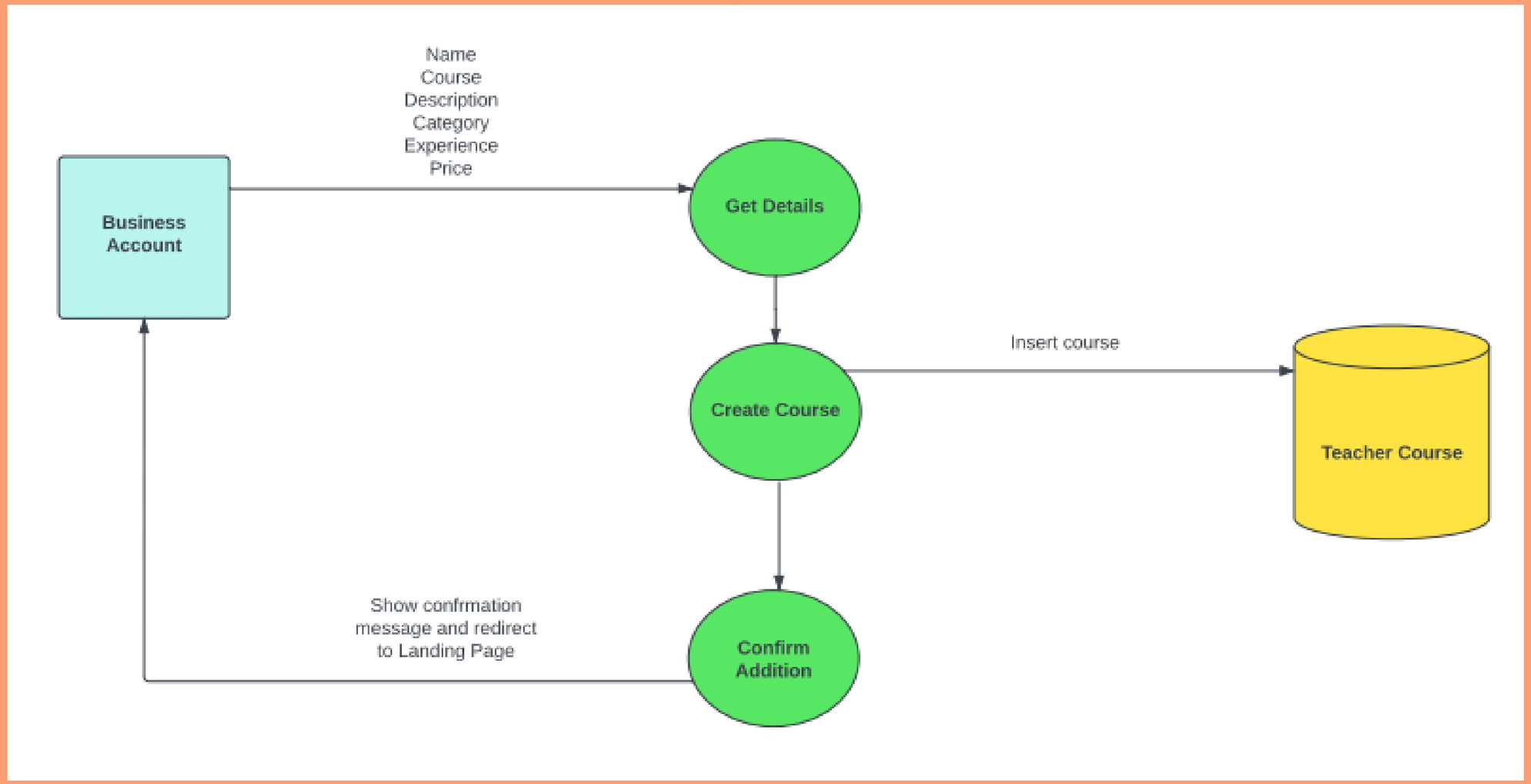
DFD - Signing in



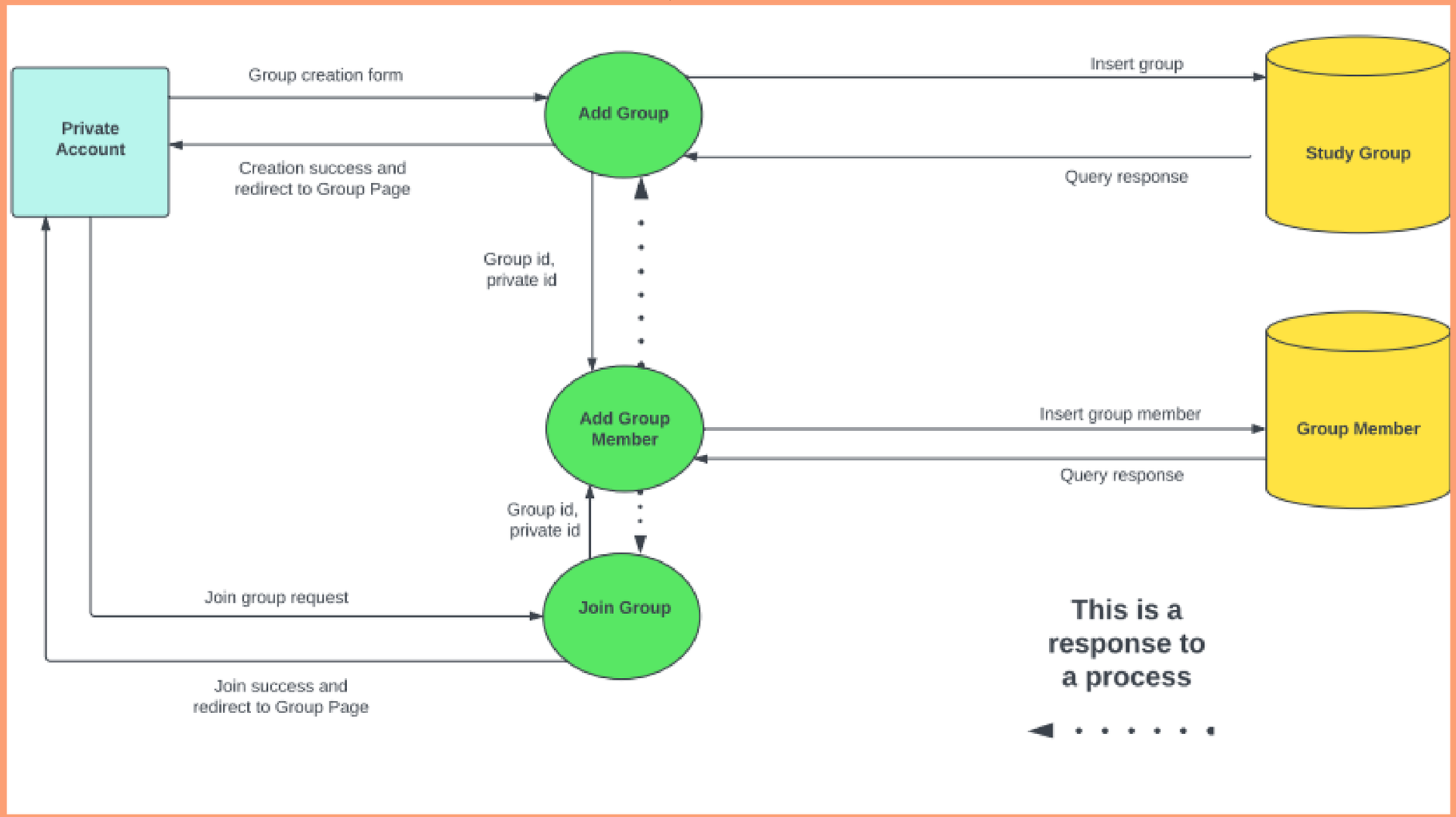
DFD - Sending Course Request



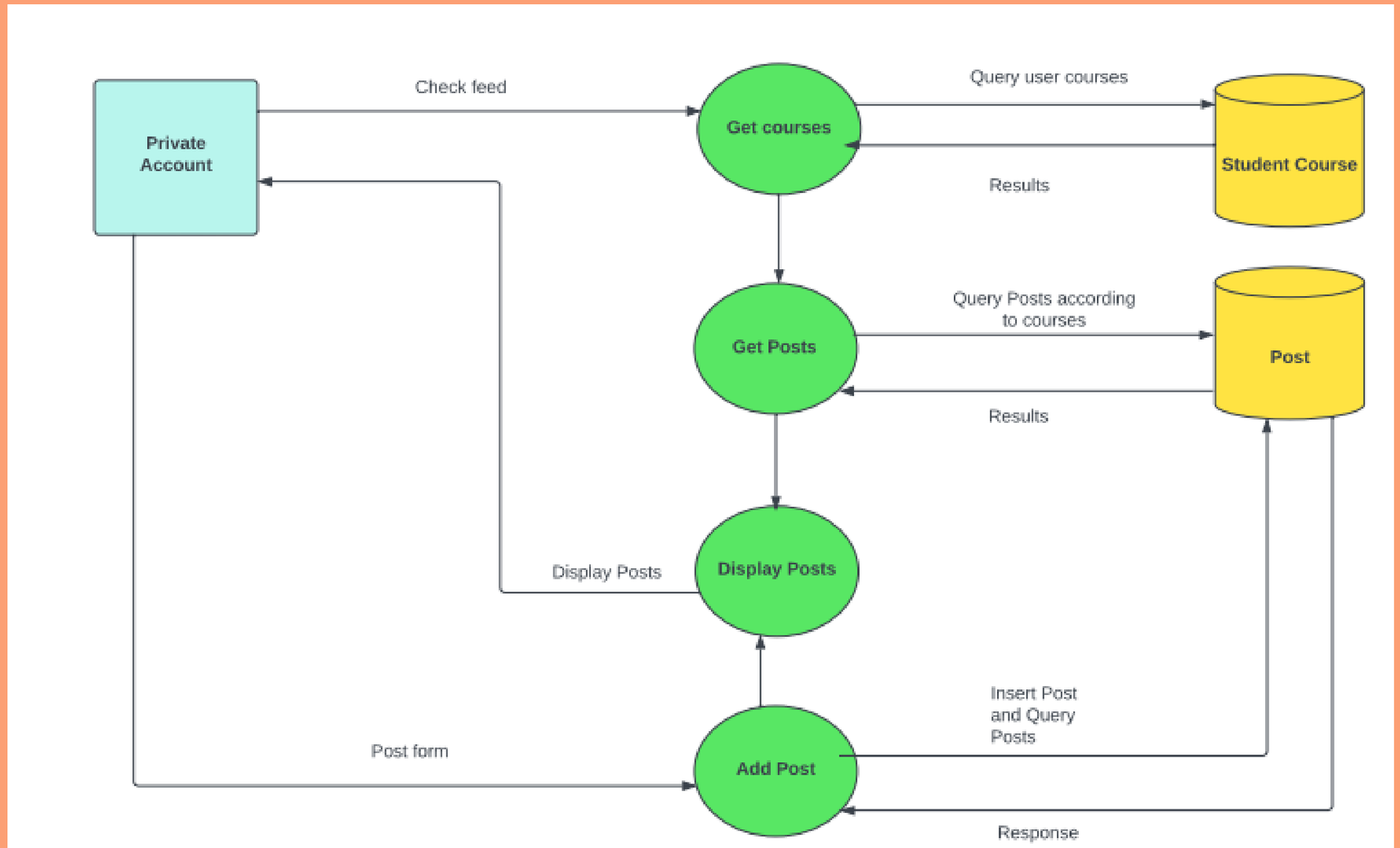
DFD - Add course



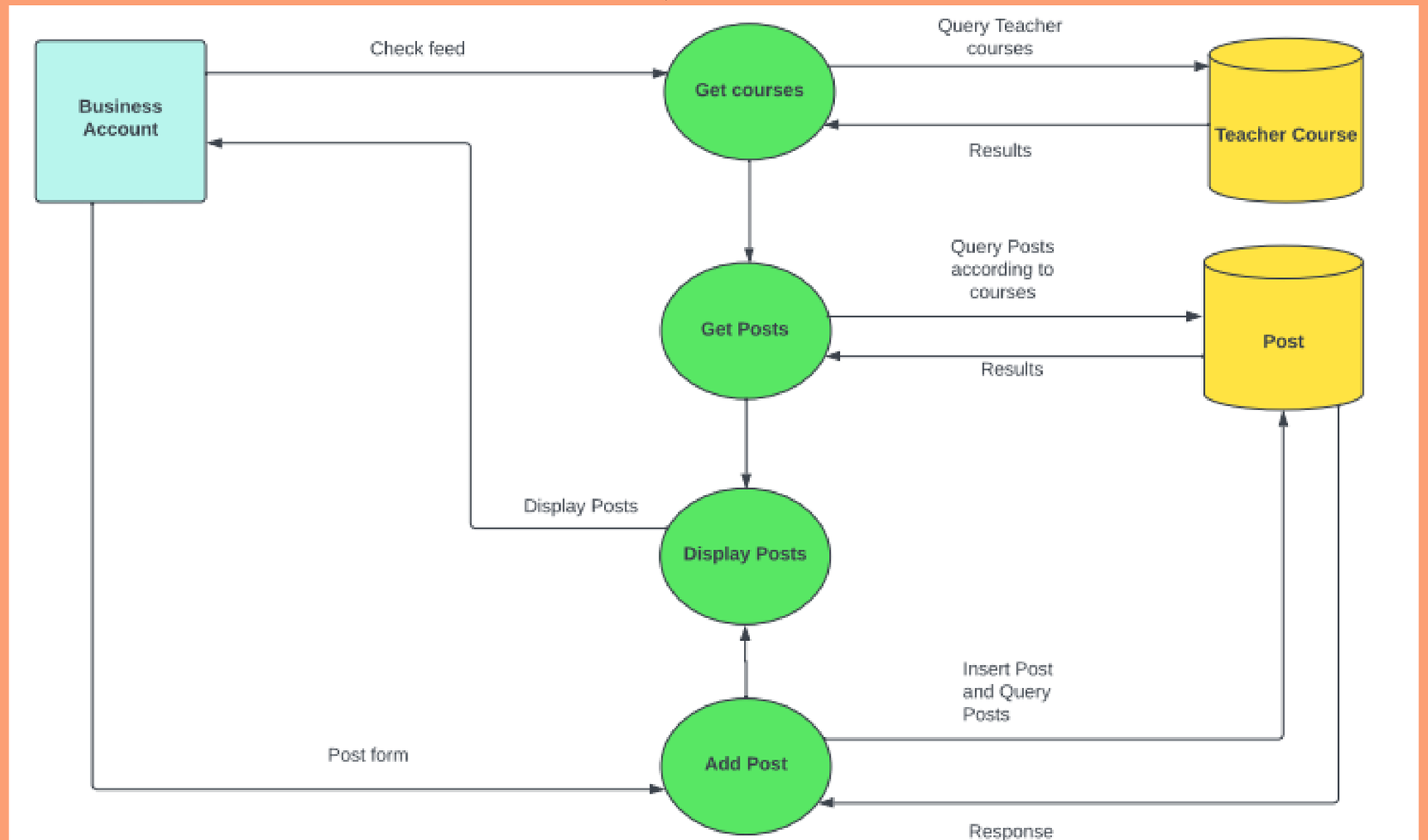
DFD - Study Group



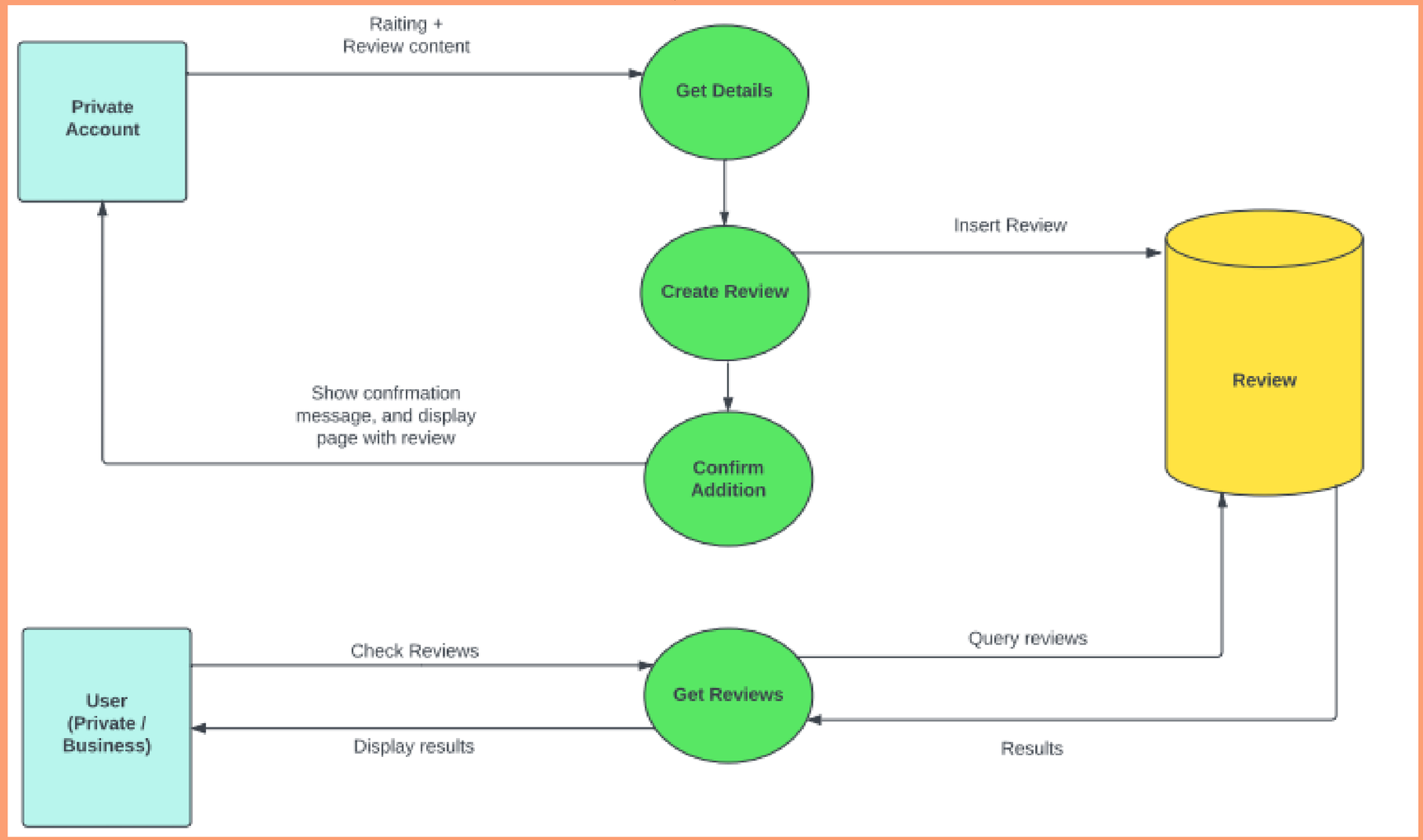
DFD - Feed


Private Account
Check feed
Get courses
Query user courses
Student Course
Results
Get Posts
Query Posts according to courses
Post
Results
Display Posts
Display Posts
Post form
Add Post
Insert Post and Query Posts
Response

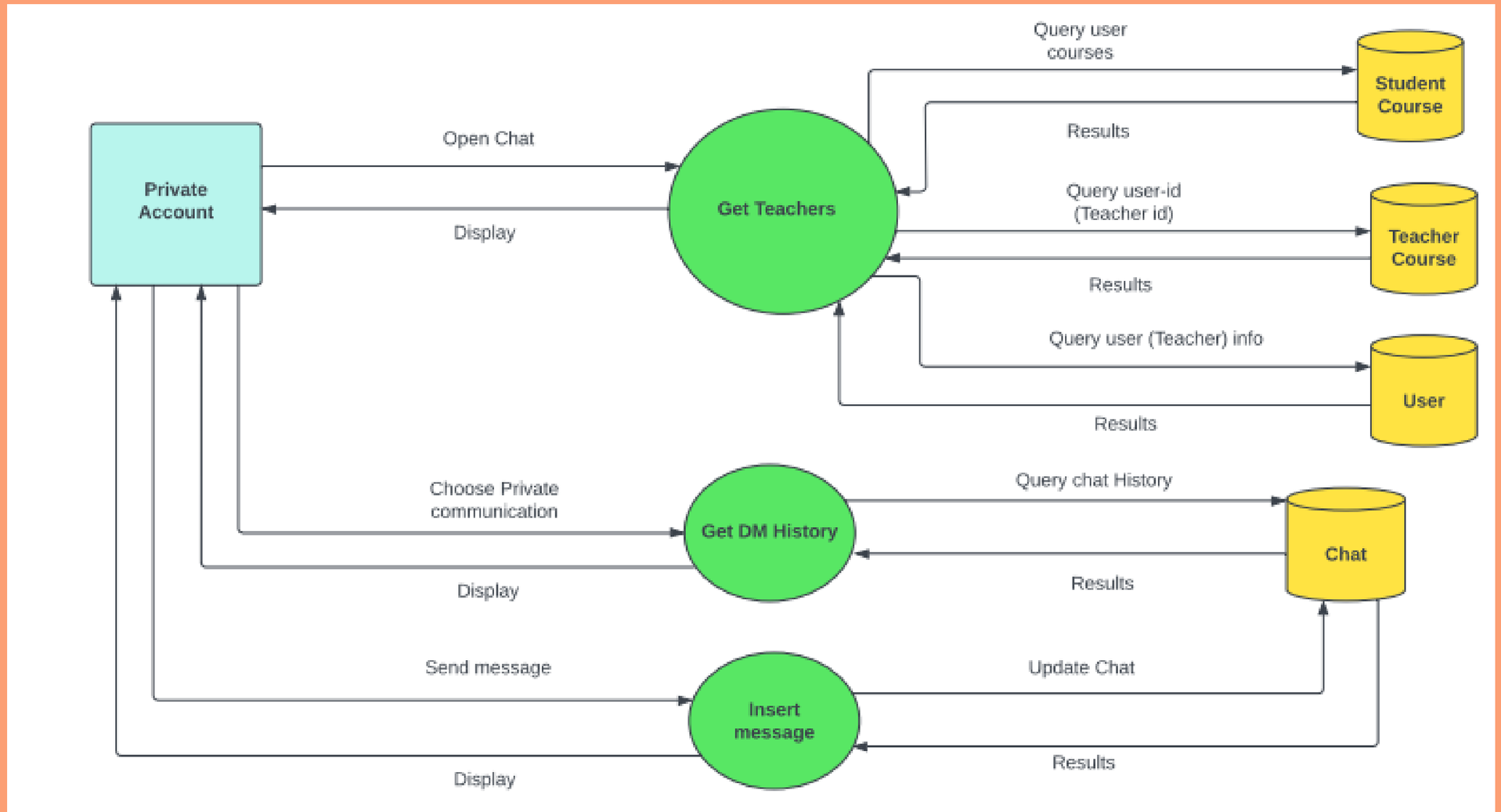
DFD - Feed


Business Account
Check feed
Get courses
Query Teacher courses
Teacher Course
Results
Get Posts
Query Posts according to courses
Post
Results
Display Posts
Display Posts
Add Post
Insert Post and Query Posts
Post form
Response

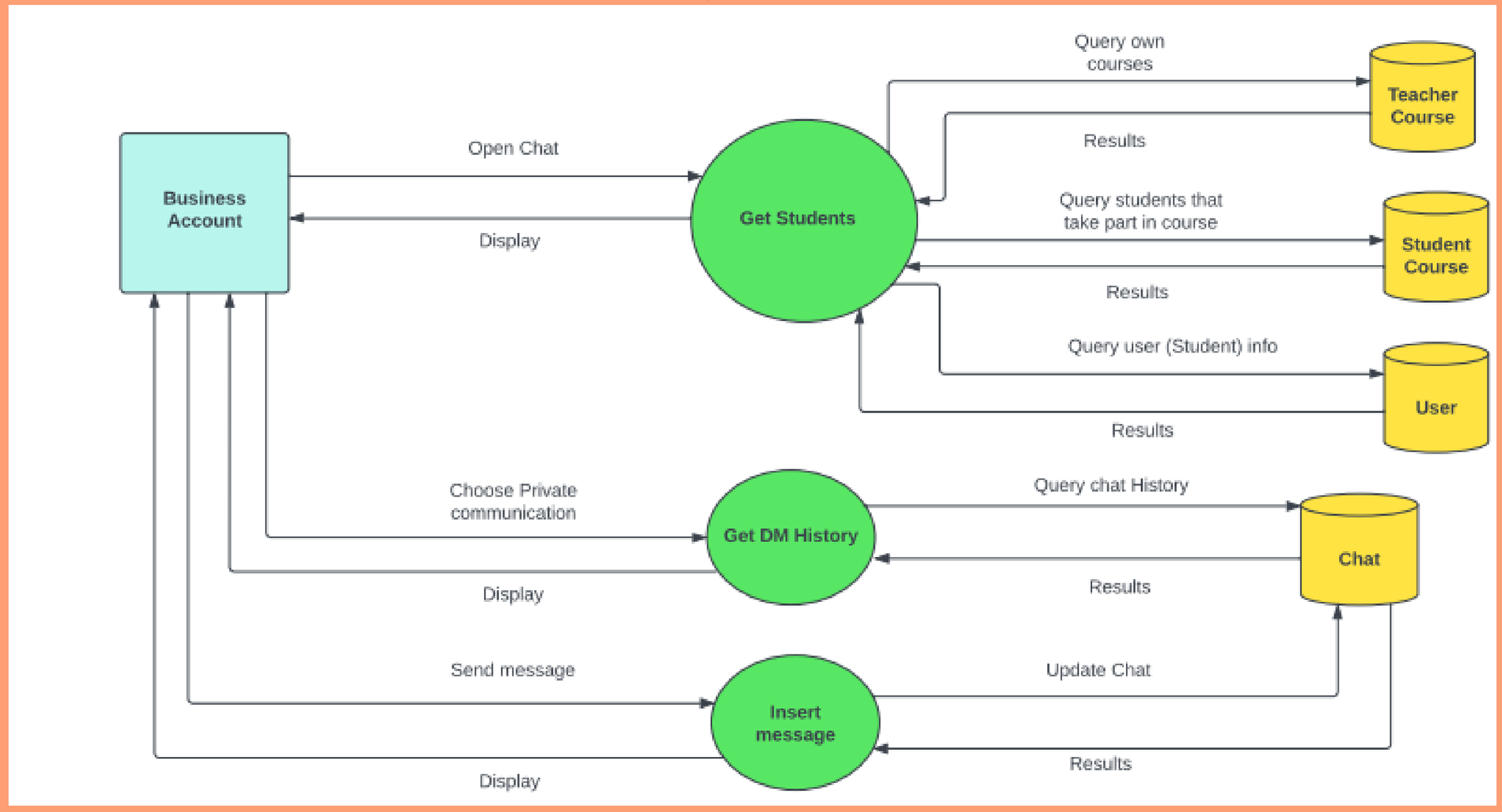
DFD - Review


Raiting +
Review content
Private Account
Get Details
Create Review
Insert Review
Review
Show confrmation message, and display page with review
Confirm Addition
Check Reviews
User (Private / Business)
Get Reviews
Query reviews
Display results
Results

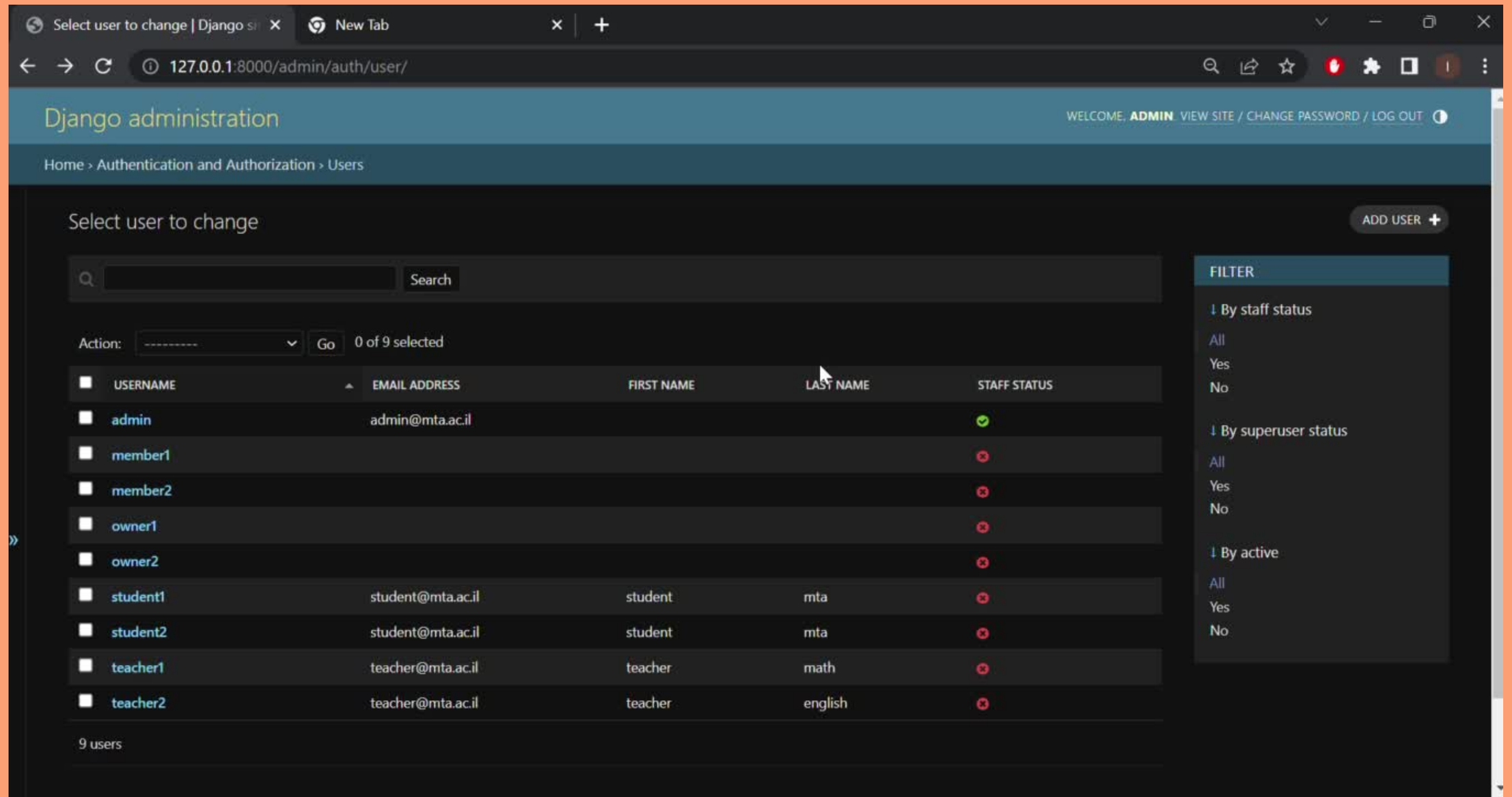
DFD - Chat


Private Account
Open Chat
Display
Get Teachers
Query user courses
Results
Student Course
Query user-id (Teacher id)
Results
Teacher Course
Query user (Teacher) info
Results
User
Choose Private communication
Display
Get DM History
Query chat History
Results
Chat
Send message
Display
Insert message
Update Chat
Results

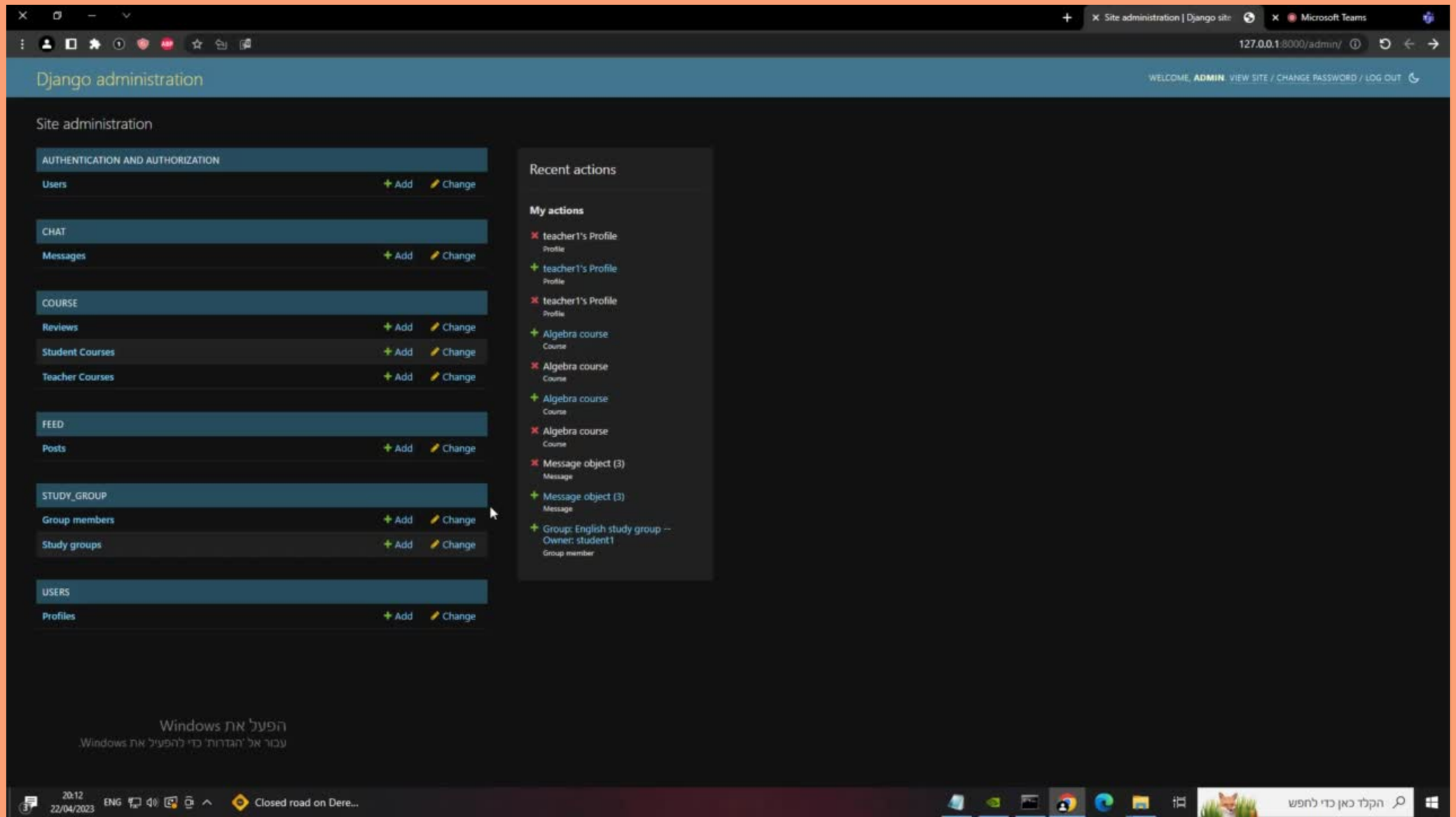
DFD - Chat


Business Account
Open Chat
Display
Get Students
Query own courses
Results
Teacher Course
Query students that take part in course
Results
Student Course
Query user (Student) info
Results
User
Choose Private communication
Display
Get DM History
Query chat History
Results
Chat
Send message
Display
Insert message
Update Chat
Results

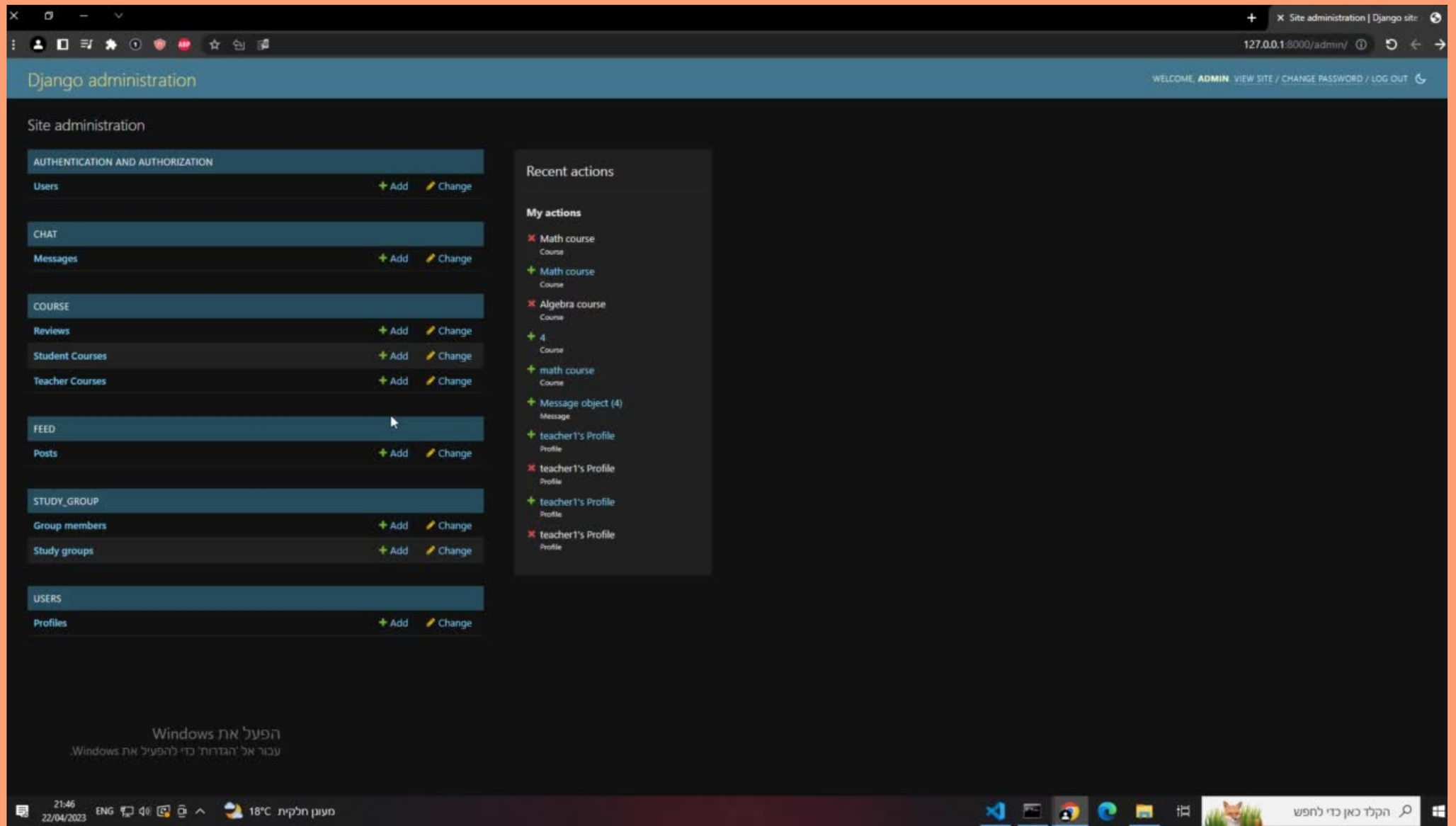
Data model via Django admin UI
User



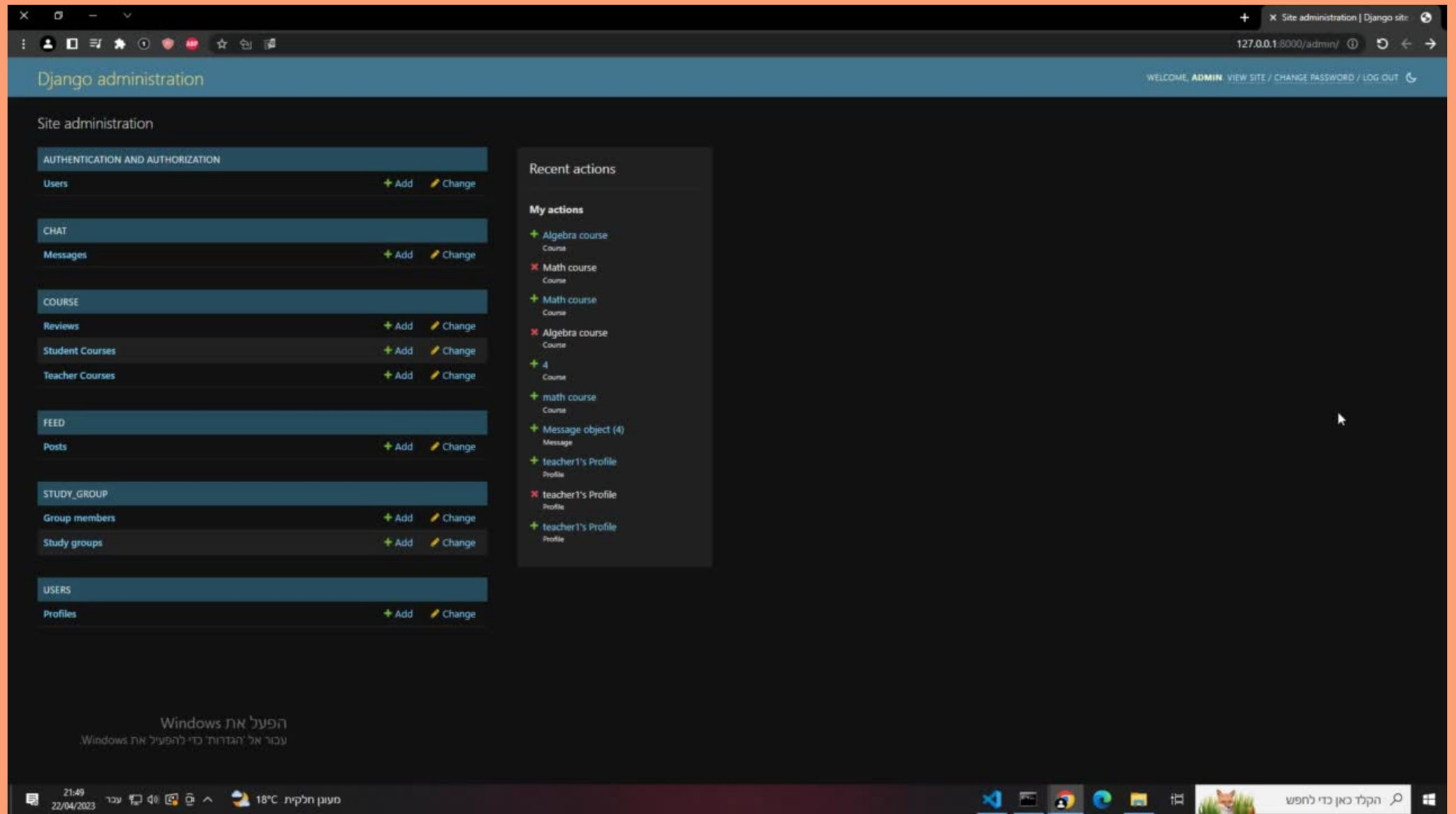
Data model via Django admin UI
Profile



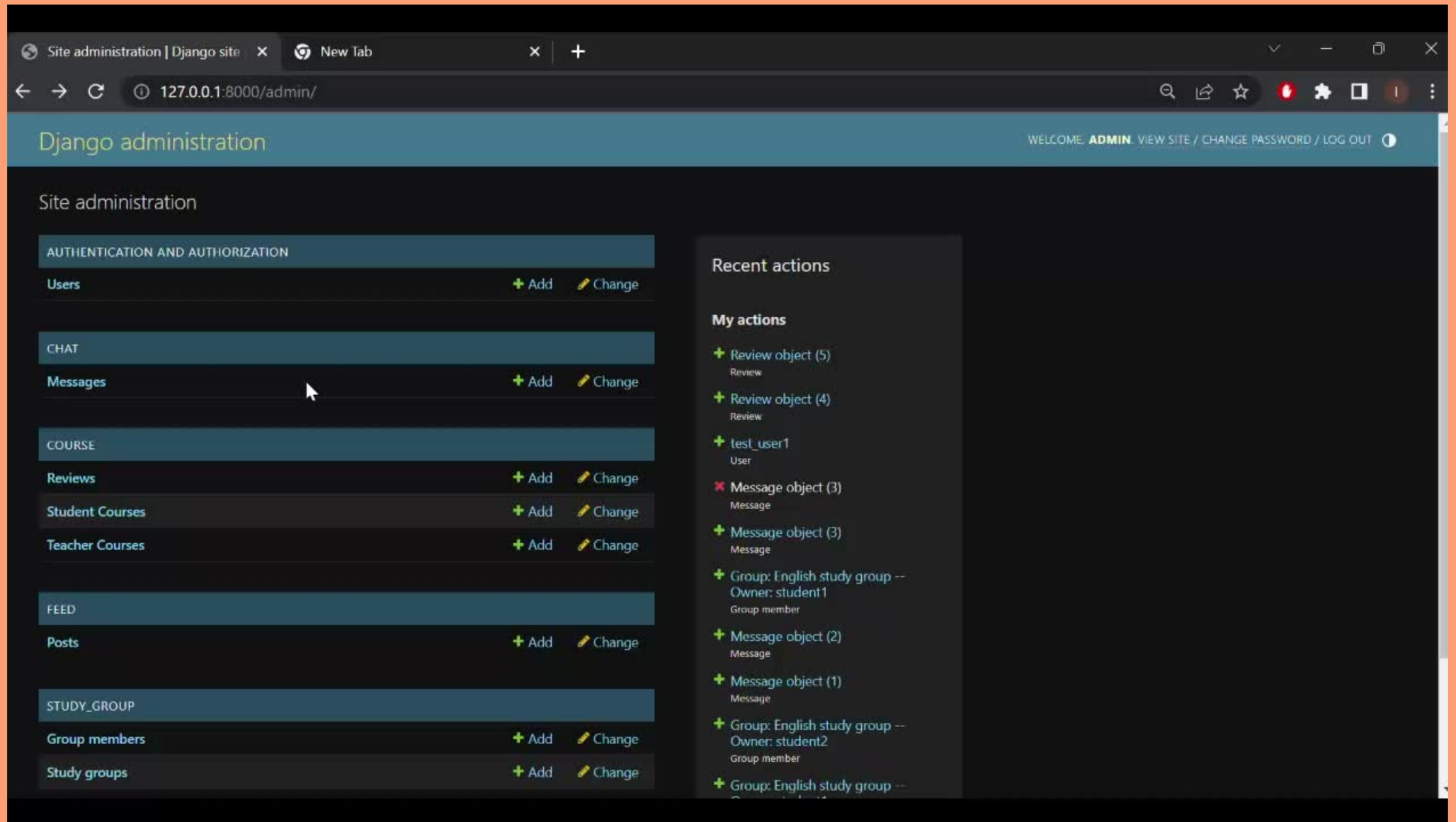
Data model via Django admin UI
Teacher Course



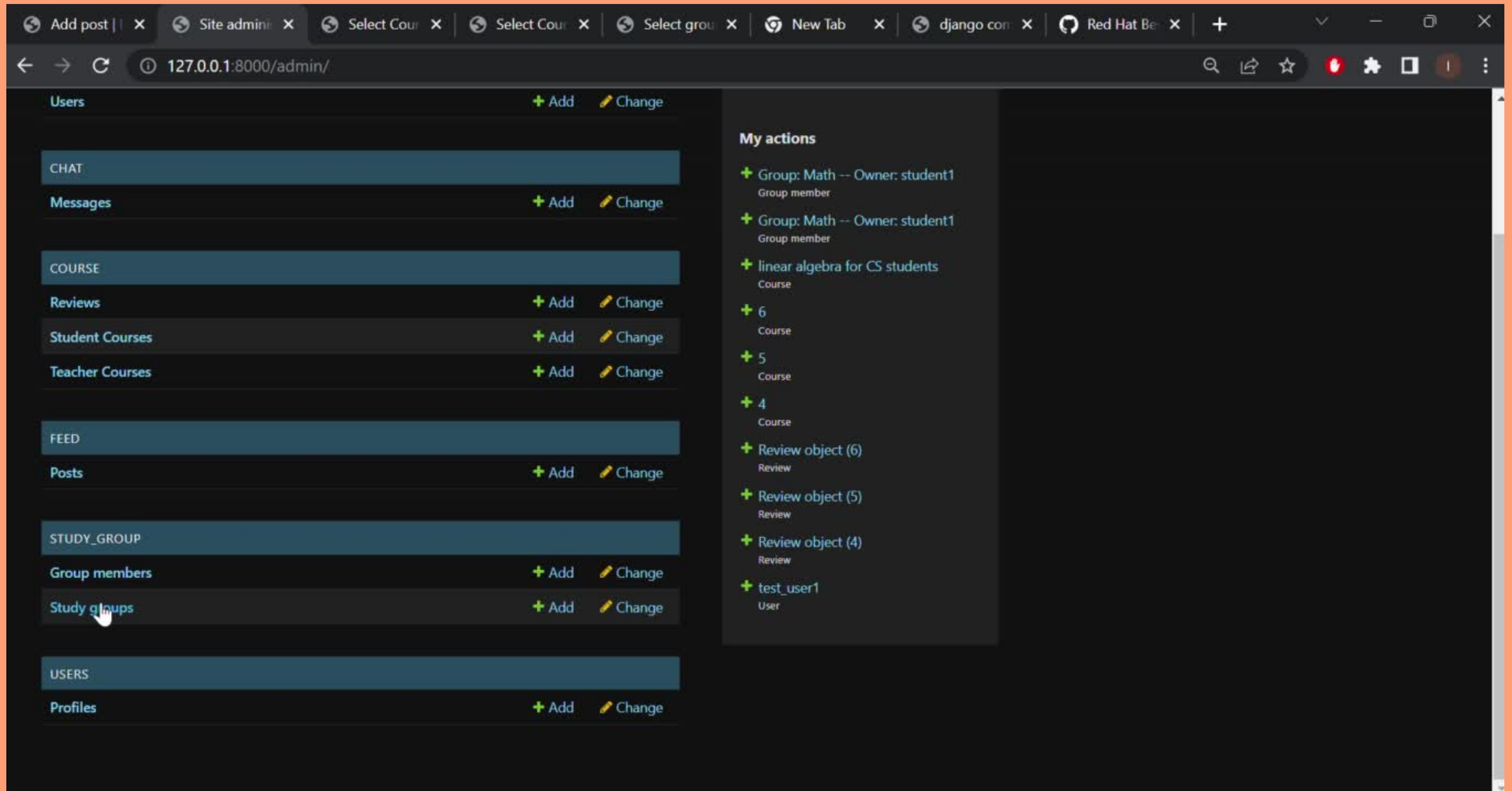
Data model via Django admin UI
Student Course



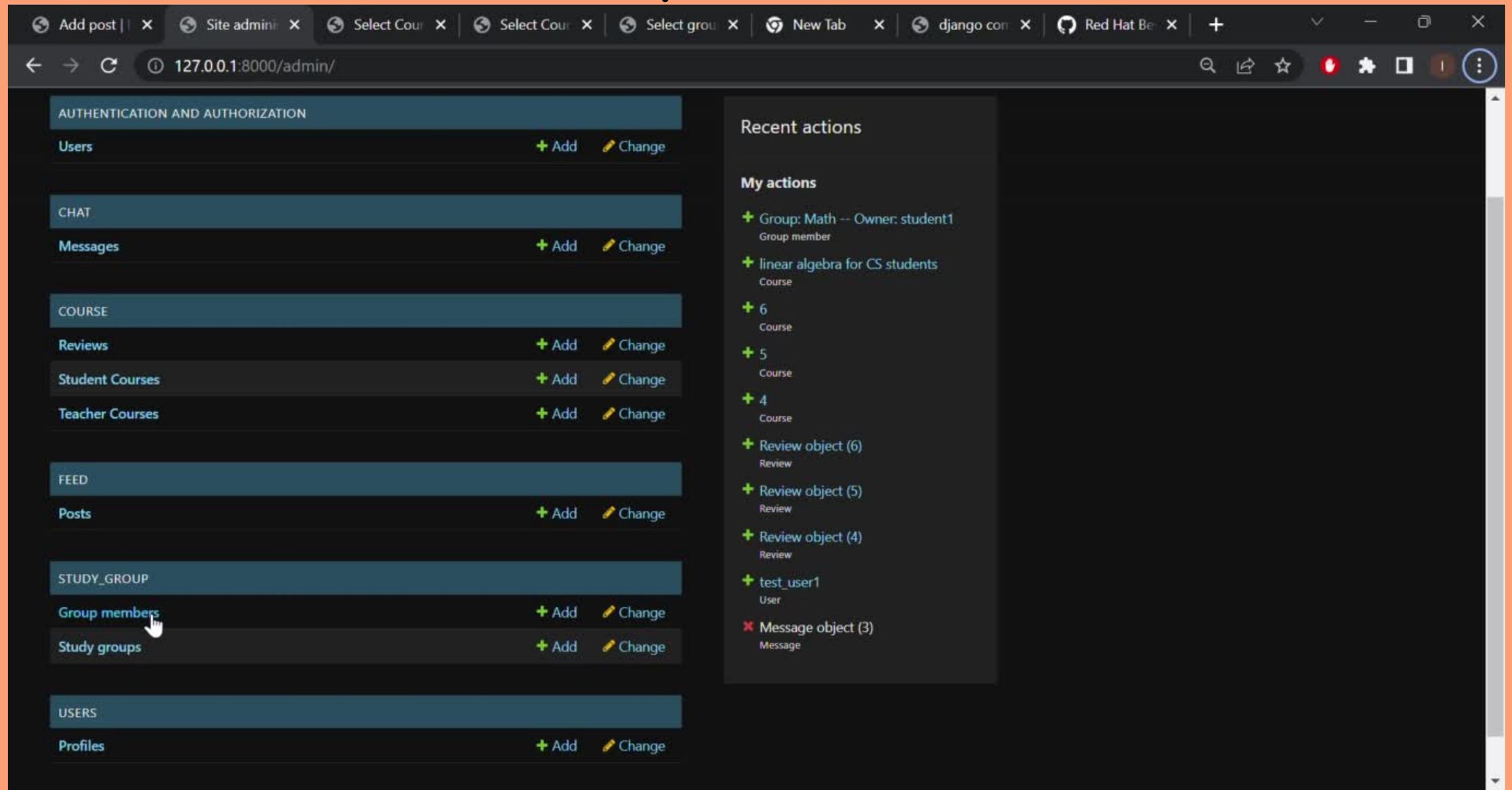
Data model via Django admin UI Review



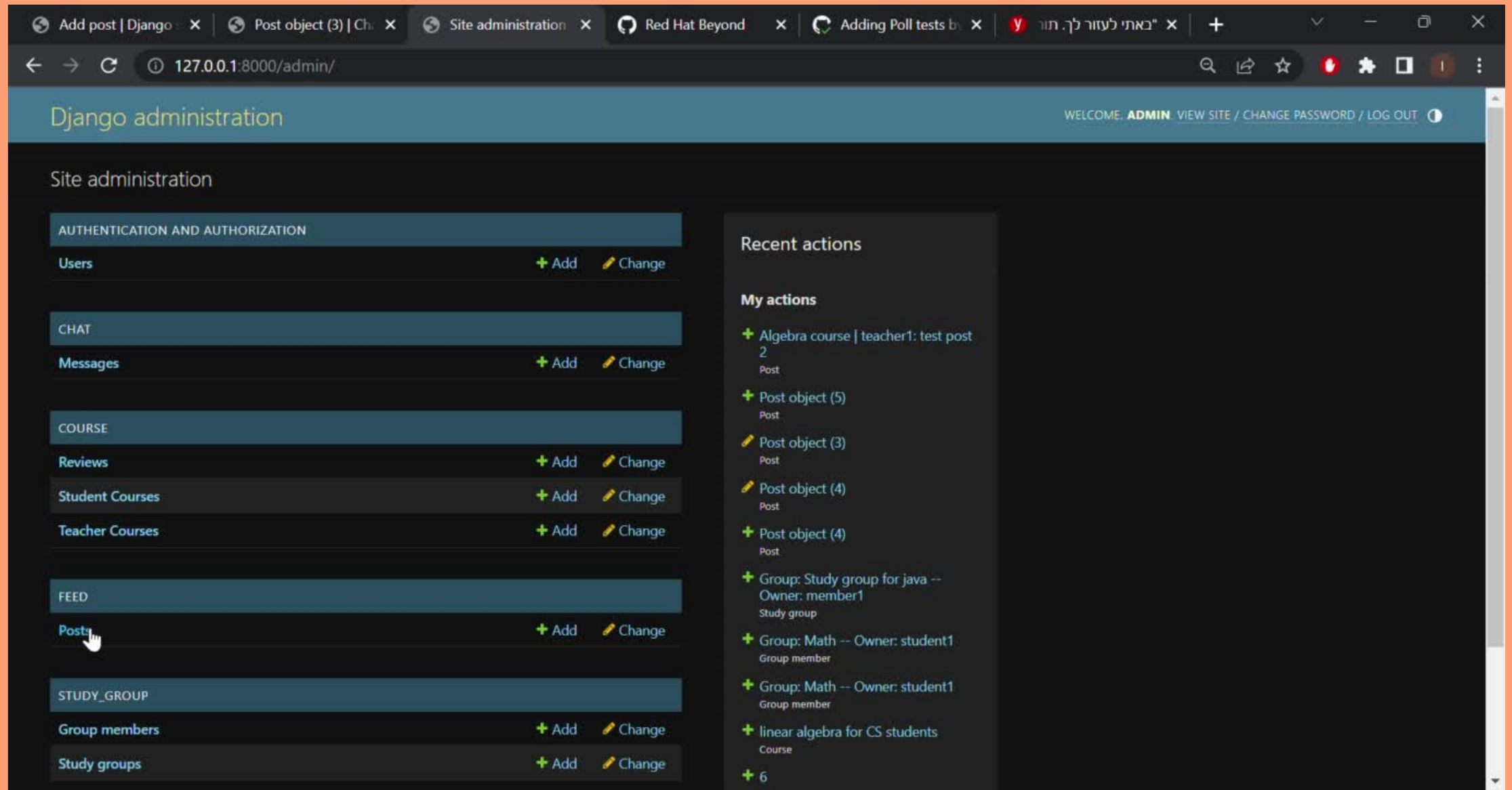
Data model via Django admin UI
Study Group



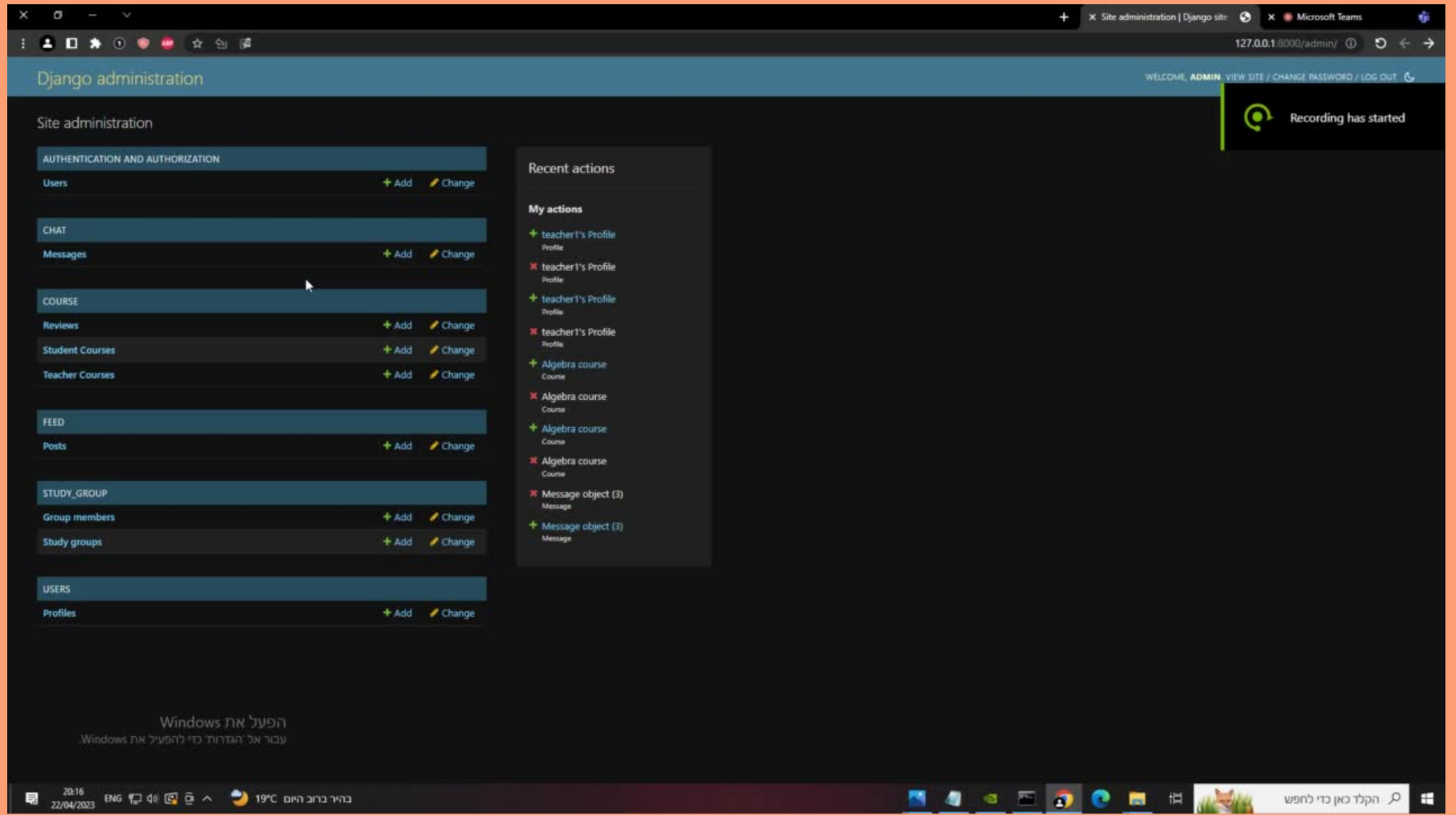
Data model via Django admin UI
Group member



Data model via Django admin UI
Post



Data model via Django admin UI
Chat



data manipulation functions via the Django shell
User - Profile

```
[vagrant@fedora vagrant]$ pipenv run python manage.py shell
^[[APython 3.9.2 (default, Feb 20 2021, 00:00:00)
[GCC 11.0.0 20210210 (Red Hat 11.0.0-0)] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from users.models import User
>>> users = User.objects.all()
>>> print(users)
<QuerySet [<User: admin>, <User: Davidcoh>, <User: RanyWhabi>, <User: IdanRosenberg>, <User: ShacharLevy>, <User: YardenGazit>, <User: ShirazYom>,
 <User: stam@gmail.com>]>
>>> # Create a new user
>>> new_user = User.objects.create_user(
... "newTestUser@gmail.com", "test_first_name", "test_last_name"
... )
>>> print(new_user)
newTestUser@gmail.com
>>> users = User.objects.all()
>>> print(users)
<QuerySet [<User: admin>, <User: Davidcoh>, <User: RanyWhabi>, <User: IdanRosenberg>, <User: ShacharLevy>, <User: YardenGazit>, <User: ShirazYom>,
 <User: stam@gmail.com>, <User: newTestUser@gmail.com>]>
>>> exit()
[vagrant@fedora vagrant]$
```

data manipulation functions via the Django shell Teacher Course

```
>>> from course.models import StudentCourses
>>> StudentCourses.objects.all()
<QuerySet [<StudentCourses: 3>, <StudentCourses: 5>, <StudentCourses: 6>]>
>>>
>>>
>>> StudentCourses.objects.get_student_courses(11)
<QuerySet [<StudentCourses: 3>]>
>>> StudentCourses.objects.get_student_courses(9)
<QuerySet [<StudentCourses: 5>, <StudentCourses: 6>]>
>>>
>>>
>>> StudentCourses.objects.get_student_pending(11)
<QuerySet [<StudentCourses: 3>]>
>>> StudentCourses.objects.get_student_pending(9)
<QuerySet [<StudentCourses: 5>, <StudentCourses: 6>]>
>>>
>>>
>>> StudentCourses.objects.get_teacher_pending(10)
<QuerySet [2]>
>>> StudentCourses.objects.get_teacher_pending(8)
<QuerySet [5, 7]>
>>>
>>>
>>> student = StudentCourses.objects.first()
>>> student
<StudentCourses: 3>
>>> student.status
'Pending'
>>> student.change_to_confirmed()
>>> student.status
'Confirmed'
>>>
>>>
>>> StudentCourses.objects.get_student_courses(11)
<QuerySet [<StudentCourses: 3>]>
>>> StudentCourses.objects.get_student_pending(11)
<QuerySet []>
>>> StudentCourses.objects.get_teacher_pending(10)
<QuerySet []>
>>>
>>>
```

data manipulation functions via the Django shell Student Course

```
>>> from course.models import StudentCourses
>>> StudentCourses.objects.all()
<QuerySet [<StudentCourses: 3>, <StudentCourses: 5>, <StudentCourses: 6>]>
>>>
>>>
>>> StudentCourses.objects.get_student_courses(11)
<QuerySet [<StudentCourses: 3>]>
>>> StudentCourses.objects.get_student_courses(9)
<QuerySet [<StudentCourses: 5>, <StudentCourses: 6>]>
>>>
>>>
>>> StudentCourses.objects.get_student_pending(11)
<QuerySet [<StudentCourses: 3>]>
>>> StudentCourses.objects.get_student_pending(9)
<QuerySet [<StudentCourses: 5>, <StudentCourses: 6>]>
>>>
>>>
>>> StudentCourses.objects.get_teacher_pending(10)
<QuerySet [2]>
>>> StudentCourses.objects.get_teacher_pending(8)
<QuerySet [5, 7]>
>>>
>>>
>>> student = StudentCourses.objects.first()
>>> student
<StudentCourses: 3>
>>> student.status
'Pending'
>>> student.change_to_confirmed()
>>> student.status
'Confirmed'
>>>
>>>
>>> StudentCourses.objects.get_student_courses(11)
<QuerySet [<StudentCourses: 3>]>
>>> StudentCourses.objects.get_student_pending(11)
<QuerySet []>
>>> StudentCourses.objects.get_teacher_pending(10)
<QuerySet []>
>>>
>>>
```

data manipulation functions via the Django shell Review

```
>>> exit()
[vagrant@fedora vagrant]$ pipenv run python manage.py shell
Python 3.9.2 (default, Feb 20 2021, 00:00:00)
[GCC 11.0.0 20210210 (Red Hat 11.0.0-0)] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from course.models import *
>>> Review.objects.all()
<QuerySet [<Review: 8>, <Review: 9>, <Review: 10>]>
>>> reviews = Review.objects.get_reviews_by_course(1)
>>> reviews
<QuerySet [<Review: 8>, <Review: 9>]>
>>> reviews = Review.objects.get_avg_rating_by_course(1)
>>> reviews
2.5
>>> reviews = Review.objects.get_number_of_review_of_course(1)
>>> reviews
2
>>>
```

data manipulation functions via the Django shell
Study Group

```
>>> StudyGroup.objects.search_group_by_keyword("not existing keyword")
<QuerySet []>
>>> StudyGroup.objects.search_group_by_keyword("math")
<QuerySet [<StudyGroup: Group: Math -- Owner: owner1>]>
>>>
```

```
>>>
>>> from study_group.models import StudyGroup
>>>
>>> StudyGroup.objects.get_all_non_full_groups()
<QuerySet [<StudyGroup: Group: Math -- Owner: owner1>, <StudyGroup: Group: History -- Owner: owner1>, <StudyGroup: Group: Science -- Owner: owner2>, <St
udyGroup: Group: Study group for java -- Owner: member1>, <StudyGroup: Group: English study group -- Owner: student1>]>
>>>
```

data manipulation functions via the Django shell Post

```
>>> from feed.models import Post
>>> from django.contrib.auth.models import User
>>>
>>> user = User.objects.get(username="teacher1")
>>> Post.objects.get_posts_for_user(user)
{<Post: Algebra course | teacher1: This is a message for all the student in this course.>: [<Post: Algebra course | student1: This is a replay from a st
udent to the post of the teacher.>], <Post: Algebra course | teacher1: This is a post by a user>: [<Post: Algebra course | teacher1: test post 2>, <Post
: Algebra course | teacher1: this is a reply post.>]}
>>>
```

data manipulation functions via the Django shell Message

```
>>> group0.get_all_group_members()
<QuerySet [<User: user0>]>
>>> message = Message(sender=user1, group=group0, message='Not in group')
>>> message.save()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/vagrant/chat/models.py", line 39, in save
    self.full_clean()
  File "/home/vagrant/.local/share/virtualenvs/vagrant-gKDsaKU3/lib/python3.9/site-packages/django/db/models/base.py",
ine 1480, in full_clean
    raise ValidationError(errors)
django.core.exceptions.ValidationError: {'sender': ['The message sender is not a part of the chat group']}
>>>
>>> message = Message(sender=user0, group=group0, message='Message 1')
>>> message.save()
>>> message2 = Message(sender=user0, group=group0, message='Message 2')
>>> message2.save()
>>>
>>> message3 = Message(sender=user1, group=group1, message='Different group')
>>> message3.save()
>>>
>>> Message.objects.get_group_chat(group_id=group0.study_group_id)
<QuerySet [<Message: Message 1>, <Message: Message 2>]>
>>>
>>> Message.objects.get_group_chat(group_id=group1.study_group_id)
<QuerySet [<Message: Different group>]>
```

data manipulation functions via the Django shell Message

```
>>> message4 = Message(sender=user0, student_course=course0, message='Course message 1')
>>> message4.save()
>>> message5 = Message(sender=user0, student_course=course0, message='Course message 2')
>>> message5.save()
>>>
>>> message6 = Message(sender=user0, student_course=course1, message='Different course')
>>> message6.save()
>>>
>>> Message.objects.get_student_course_chat(student_course_id=course0.student_course_id)
<QuerySet [<Message: Course message 1>, <Message: Course message 2>]>
>>>
>>> Message.objects.get_student_course_chat(student_course_id=course1.student_course_id)
<QuerySet [<Message: Different course>]>
>>>
```

Tests - Profile

```
104
105     # Check invalid phone number
106     def test_validate_phone_number(self):
107         with pytest.raises(ValidationError):
108             user = User(
109                         username=USERNAME,
110                         first_name=FIRSTNAME,
111                         last_name=LASTNAME,
112                         password=PASSWORD,
113                         email=EMAIL)
114             user.save()
115             user.profile.phone_number = PHONE_NUMBER
116             user.profile.clean_fields()
117
118     # Test for invalid city
119     def test_invalid_city_length(self, new_user):
120         with pytest.raises(ValidationError):
121             new_user.save()
122             new_user.profile.city = LONG_PROFESSION
123             new_user.profile.clean_fields()
124
125     # Test for invalid account_type
126     def test_invalid_account_type(self, new_user):
127         with pytest.raises(ValidationError):
128             new_user.save()
129             new_user.profile.account_type = 'T'
130             new_user.profile.clean_fields()
131
132     # Test for invalid meeting_method
133     def test_invalid_meeting_method(self, new_user):
134         with pytest.raises(ValidationError):
135             new_user.save()
136             new_user.profile.meeting_method = 'B'
137             new_user.profile.clean_fields()
```

Tests - Profile

```python
@pytest.mark.django_db()
class TestUserModel():

    # In this test we delete only the profile part, and check if the profile was deleted and not the whole user
    def test_delete_only_profile_from_db(self, new_user):
        new_user.save()
        new_user.profile.delete()
        assert new_user in User.objects.all()
        assert new_user.profile not in Profile.objects.all()

    # Check invalid email
    def test_validate_email_addr(self):
        with pytest.raises(ValidationError):
            user = User(
                        username=USERNAME,
                        first_name=FIRSTNAME,
                        last_name=LASTNAME,
                        password=PASSWORD,
                        email="check")
            user.full_clean()
```

Tests - Teacher Course

```python
class TestTeacherCourse:

    def test_price_filter(self, teacher_course_one):
        assert teacher_course_one in TeacherCourses.objects.in_price_range(PRICE-1, PRICE+1)
        assert teacher_course_one not in TeacherCourses.objects.in_price_range(PRICE+1, PRICE+2)
        assert teacher_course_one not in TeacherCourses.objects.in_price_range(PRICE-1, PRICE-2)
        assert teacher_course_one in TeacherCourses.objects.in_price_range(PRICE, PRICE)

    def test_category_filter(self, teacher_course_one):
        assert teacher_course_one in TeacherCourses.objects.in_category(CATEGORY)
        assert teacher_course_one not in TeacherCourses.objects.in_category(CATEGORY + "not a category")

    def test_experience_filter(self, teacher_course_one):
        assert teacher_course_one in TeacherCourses.objects.got_experience(YEARS_OF_EXP)
        assert teacher_course_one in TeacherCourses.objects.got_experience(YEARS_OF_EXP - 1)
        assert teacher_course_one not in TeacherCourses.objects.got_experience(YEARS_OF_EXP + 1)

    def test_search_name_filter(self, teacher_course_one):
        assert teacher_course_one in TeacherCourses.objects.search_name(COURSE_NAME)
        assert teacher_course_one in TeacherCourses.objects.search_name(DESCRIPTION)
        assert teacher_course_one in TeacherCourses.objects.search_name(COURSE_NAME[:2])
        assert teacher_course_one in TeacherCourses.objects.search_name(DESCRIPTION[:2])
        assert teacher_course_one not in TeacherCourses.objects.search_name("not a course name or description")

    def test_difficulty_level_filter(self, teacher_course_one):
        assert teacher_course_one in TeacherCourses.objects.get_level(DIFFICULTY)
        assert teacher_course_one not in TeacherCourses.objects.get_level('B')

    def test_get_teacher_courses_filter(self, teacher_course_one, teacher_course_two):
        teacher_id1 = teacher_course_one.teacher_id.pk
        teacher_id2 = teacher_course_two.teacher_id.pk

        assert teacher_course_one in TeacherCourses.objects.get_teacher_courses(teacher_id1)
        assert teacher_id1 != teacher_id2
        assert teacher_course_one not in TeacherCourses.objects.get_teacher_courses(teacher_id2)

    def test_course_name_too_long_fails_validation(self, db, teacher_course_zero):
        with pytest.raises(ValidationError):
            teacher_course_zero.course_name = NAME_TOO_LONG
            teacher_course_zero.save()

    def test_course_name_empty_fails_validation(self, db, user_one):
        with pytest.raises(ValidationError):
            teacher_course = TeacherCourses(teacher_id=user_one, description=DESCRIPTION,
                                            difficulty_level=DIFFICULTY, category=CATEGORY, price=PRICE,
                                            years_of_experience=YEARS_OF_EXP)
            teacher_course.save()
```

Tests - Teacher Course

```python
def test_price_negative_amount_fails_validation(self, db, teacher_course_zero):
    with pytest.raises(IntegrityError):
        teacher_course_zero.price = -PRICE
        teacher_course_zero.save()

def test_years_of_experience_negative_amount_fails_validation(self, db, teacher_course_zero):
    with pytest.raises(ValidationError):
        teacher_course_zero.years_of_experience = -YEARS_OF_EXP
        teacher_course_zero.save()

def test_years_of_experience_invaled_fails_validation(self, db, teacher_course_zero):
    with pytest.raises(ValidationError):
        teacher_course_zero.years_of_experience = NOT_VALED_YEARS_OF_EXP
        teacher_course_zero.save()

def test_difficulty_level_invaled_choice_fails_validation(self, db, teacher_course_zero):
    with pytest.raises(ValidationError):
        teacher_course_zero.difficulty_level = NOT_VALED_DIFFICULTY
        teacher_course_zero.save()

def test_difficulty_level_empty_fails_validation(self, db, user_one):
    with pytest.raises(ValidationError):
        teacher_course = TeacherCourses(teacher_id=user_one, course_name=COURSE_NAME, description=DESCRIPTION,
                                        category=CATEGORY, price=PRICE, years_of_experience=YEARS_OF_EXP)
        teacher_course.save()

def test_category_invaled_choice_fails_validation(self, db, teacher_course_zero):
    with pytest.raises(ValidationError):
        teacher_course_zero.category = NOT_VALED_CATEGORY
        teacher_course_zero.save()

def test_category_empty_fails_validation(self, db, user_one):
    with pytest.raises(ValidationError):
        teacher_course = TeacherCourses(teacher_id=user_one, course_name=COURSE_NAME, description=DESCRIPTION,
                                        difficulty_level=DIFFICULTY, price=PRICE, years_of_experience=YEARS_OF_EXP)
        teacher_course.save()

def test_teacher_id_empty_fails_validation(self, db, user_zero):
    with pytest.raises(ValidationError):
        teacher_course = TeacherCourses(teacher_id=user_zero, course_name=COURSE_NAME, description=DESCRIPTION,
                                        difficulty_level=DIFFICULTY, price=PRICE, years_of_experience=YEARS_OF_EXP)
        teacher_course.save()
```

Tests - Student Course

```python
class TestStudentCourse:

    def test_change_to_confirmed(self, student_course_one):
        assert student_course_one.status == PENDING
        student_course_one.change_to_confirmed()
        assert student_course_one.status == CONFIRMED

    def test_teacher_pending(self, student_course_one, student_course_two, user_zero, user_one):
        course_id1 = student_course_one.teacher_course_id.course_id
        course_id2 = student_course_two.teacher_course_id.course_id
        teacher_id1 = user_zero.pk
        teacher_id2 = user_one.pk

        assert course_id1 in StudentCourses.objects.get_teacher_pending(teacher_id1)
        assert course_id1 not in StudentCourses.objects.get_teacher_pending(teacher_id2)

        assert course_id2 not in StudentCourses.objects.get_teacher_pending(teacher_id1)
        assert course_id2 in StudentCourses.objects.get_teacher_pending(teacher_id2)

    def test_teacher_pending_with_change_to_confirmed(self, student_course_one, user_zero):
        course_id1 = student_course_one.teacher_course_id.course_id
        teacher_id1 = user_zero.pk

        assert course_id1 in StudentCourses.objects.get_teacher_pending(teacher_id1)
        student_course_one.change_to_confirmed()
        assert course_id1 not in StudentCourses.objects.get_teacher_pending(teacher_id1)

    def test_student_pending(self, student_course_one, student_course_two, user_zero, user_one):
        student_id1 = user_one
        student_id2 = user_zero

        assert student_course_one in StudentCourses.objects.get_student_pending(student_id1)
        assert student_course_one not in StudentCourses.objects.get_student_pending(student_id2)

        assert student_course_two not in StudentCourses.objects.get_student_pending(student_id1)
        assert student_course_two in StudentCourses.objects.get_student_pending(student_id2)

    def test_student_pending_with_change_to_confirmed(self, student_course_one, user_one):
        student_id1 = user_one

        assert student_course_one in StudentCourses.objects.get_student_pending(student_id1)
        student_course_one.change_to_confirmed()
        assert student_course_one not in StudentCourses.objects.get_student_pending(student_id1)
```

הפעל את Windows

Tests - Student Course

```python
def test_student_courses(self, student_course_one, student_course_two, user_zero, user_one):
    student_id1 = user_one
    student_id2 = user_zero

    assert student_course_one in StudentCourses.objects.get_student_courses(student_id1)
    assert student_course_two not in StudentCourses.objects.get_student_courses(student_id1)

    assert student_course_one not in StudentCourses.objects.get_student_courses(student_id2)
    assert student_course_two in StudentCourses.objects.get_student_courses(student_id2)

def test_regitser_to_your_own_course_fails_validation(self, db, teacher_course_one, user_zero):
    with pytest.raises(ValidationError):
        student_course = StudentCourses(student_id=user_zero, teacher_course_id=teacher_course_one)
        student_course.save()

def test_regitser_to_the_same_course_twice__fails_validation(self, db, teacher_course_one, user_one):
    with pytest.raises(ValidationError):
        student_course = StudentCourses(student_id=user_one, teacher_course_id=teacher_course_one)
        student_course.save()
        student_course1 = StudentCourses(student_id=user_one, teacher_course_id=teacher_course_one)
        student_course1.save()

def test_invaled_status_fails_validation(self, db, teacher_course_one, user_one):
    with pytest.raises(ValidationError):
        student_course = StudentCourses(student_id=user_one, teacher_course_id=teacher_course_one,
                                        status=INVALID_STATUS)
        student_course.save()

def test_registering_a_student_without_user_fails_validation(self, db, teacher_course_two, user_zero):
    with pytest.raises(ValidationError):
        student_course = StudentCourses(student_id=user_zero, teacher_course_id=teacher_course_two)
        student_course.save()

def test_registering_a_student_without_teacher_course_fails_validation(self, db, teacher_course_zero, user_one):
    with pytest.raises(ValidationError):
        student_course = StudentCourses(student_id=user_one, teacher_course_id=teacher_course_zero)
        student_course.save()
```

Tests - Review

```python
@pytest.mark.django_db()
class TestReview:
    @pytest.mark.parametrize("rating", [0, 6])
    def test_rating_limit(self, rating, new_review):
        new_review.rating = rating
        with pytest.raises(ValidationError):
            new_review.full_clean()

    def test_review_average_function(self, persist_review, persist_review_number_two):
        assert Review.objects.get_avg_rating_by_course(1) == 1.5

    def test_get_review_by_course_id(self, persist_review):
        assert [persist_review] == list(Review.objects.get_reviews_by_course(course_id=1))

    def test_student_reviewer(self, persist_review):
        assert persist_review.student_id.username != persist_review.course_id.teacher_id.username

    def test_get_number_of_review_of_course(self, persist_review, persist_review_number_two):
        result = Review.objects.get_number_of_review_of_course(persist_review.course_id)
        assert result == 2
```

Tests - Study Group

```python
def test_get_all_group_members(self, persist_group, persist_user, persist_second_user):
    assert not list(persist_group.get_all_group_members())

    persist_group.join_group(persist_user)
    persist_group.join_group(persist_second_user)
    assert set([persist_user, persist_second_user]) == set(persist_group.get_all_group_members())

def test_is_user_in_group(self, persist_group, persist_user):
    assert not persist_group.is_user_in_group(persist_user)
    persist_group.join_group(persist_user)
    assert persist_group.is_user_in_group(persist_user)

def test_join_group_already_in(self, persist_group, persist_user):
    persist_group.join_group(persist_user)

    with pytest.raises(ValueError):
        persist_group.join_group(persist_user)

@pytest.mark.parametrize("negative_capacity", [-2])
def test_negative_capacity(self, new_group, negative_capacity):
    new_group.capacity = negative_capacity
```

TestStudyGroup › test_join_group_already_in()

Tests - Study Group

```python
@pytest.mark.django_db
class TestStudyGroup:
    def test_join_group(self, persist_group, persist_user):
        persist_group.join_group(persist_user)
        assert GroupMember.objects.filter(group_id=persist_group, private_id=persist_user).exists()
        assert persist_group.group_members.count() == 1
        assert persist_group.group_members.first().private_id == persist_user

    def test_is_group_full(self, persist_group):
        for i in range(persist_group.capacity):
            assert not persist_group.is_group_full()
            persist_group.join_group(User.objects.create(username=f'user{i}', password='pass'))

        assert persist_group.is_group_full()

    def test_join_full_group(self, persist_user, persist_group):
        for i in range(persist_group.capacity):
            persist_group.join_group(User.objects.create(username=f'user{i}', password='pass'))

        with pytest.raises(ValueError):
            persist_group.join_group(persist_user)
```

```python
    @pytest.mark.parametrize("negative_capacity", [-2])
    def test_negative_capacity(self, new_group, negative_capacity):
        new_group.capacity = negative_capacity
        with pytest.raises(IntegrityError):
            new_group.save()

    @pytest.mark.parametrize("over_char_limit", ["Hippopotomonstrosesquippedaliophobia Support Group"])
    def test_field_over_char_limit(self, persist_group, over_char_limit):
        persist_group.field = over_char_limit
        with pytest.raises(ValidationError):
            persist_group.save()
```

Tests - Study Group

```python
@pytest.mark.django_db
class TestStudyGroupManager:
    def test_get_all_non_full_groups(self, persist_group, persist_second_group):
        for i in range(persist_group.capacity):
            persist_group.join_group(User.objects.create(username=f'user{i}', password='pass'))

        result = StudyGroup.objects.get_all_non_full_groups()
        assert persist_group not in result
        assert persist_second_group in result

    def test_search_group_by_keyword(self, persist_group, persist_second_group):
        empty = StudyGroup.objects.search_group_by_keyword("this keyword should not exist")
        assert not empty

        math_keyword_set = StudyGroup.objects.search_group_by_keyword("cs")
        assert list(math_keyword_set) == [persist_group]

        guitar_keyword_set = StudyGroup.objects.search_group_by_keyword("guitar")
        assert list(guitar_keyword_set) == [persist_second_group]

        groups_keyword_set = StudyGroup.objects.search_group_by_keyword("fixture")
        assert set(groups_keyword_set) == set([persist_group, persist_second_group])
```

Tests - Post

```python
def test_is_root_post(self, persist_post_for_first_course, persist_reply_for_first_cours
    assert persist_post_for_first_course.is_root_post()
    assert not persist_reply_for_first_course.is_root_post()

def test_post_is_only_reply_to_root_post(self, persist_reply_for_first_course,
                                         persist_post_for_second_course_student):
    persist_post_for_second_course_student.parent_post_id = persist_reply_for_first_course
    with pytest.raises(ValidationError):
        persist_post_for_second_course_student.save()

def test_is_user_able_to_post_in_course(self, persist_user, persist_course):
    assert not Post.is_user_able_to_post_in_course(persist_user, persist_course)
    user_joined_group = StudentCourses(student_id=persist_user, teacher_course_id=persist_course,
                                       status="Confirmed")
    user_joined_group.save()
    assert Post.is_user_able_to_post_in_course(persist_user, persist_course)

def create_post_for_user_not_related_to_course(self, persist_user, persist_course):
    assert not Post.is_user_able_to_post_in_course(persist_user, persist_course)
    invalid_post = Post(course_id=persist_course, user_id=persist_user, content="should be invalid")
    with pytest.raises(ValidationError):
        invalid_post.save()
```

Tests - Post

```python
@pytest.mark.django_db
class TestPostManager:
    def test_get_post_for_user(self, persist_user,
                               persist_first_student_course, persist_second_student_course,
                               persist_post_for_first_course, persist_reply_for_first_course,
                               persist_post_for_second_course_student):
        expected = {persist_post_for_first_course: [persist_reply_for_first_course]}
        posts_user = PostManager.get_posts_for_user(persist_user)

        assert posts_user == expected
        assert persist_post_for_second_course_student not in expected.items()

    @pytest.mark.parametrize("num_of_posts", [4])
    def test_posts_order(self, num_of_posts, persist_user, persist_course, persist_first_student_course):
        for i in range(num_of_posts):
            time.sleep(1)
            post = Post(course_id=persist_course, user_id=persist_user, content=f"post num {i}")
            post.save()

        posts_for_user = list(PostManager.get_posts_for_user(persist_user).keys())
        assert all(earlier.date <= later.date for earlier, later in zip(posts_for_user, posts_for_user[1:
```

Tests - Message

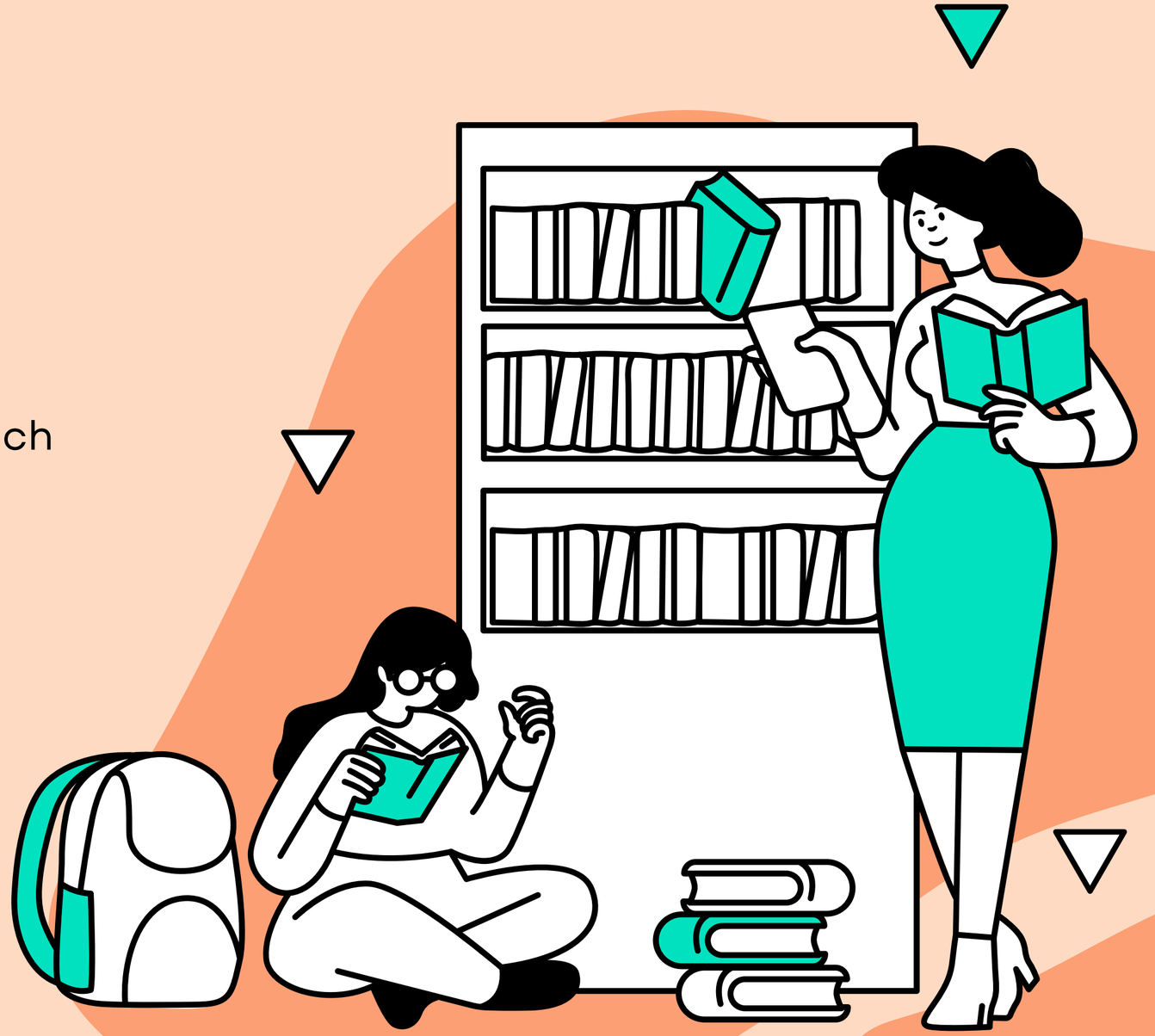
```python
@pytest.mark.django_db
class TestMessage:
    def test_message_save_with_group(self, message_with_group):
        message_with_group.save()
        assert message_with_group in Message.objects.all()

    def test_message_save_with_student_course_id(self, message_with_student_course):
        message_with_student_course.save()
        assert message_with_student_course in Message.objects.all()

    def test_message_save_with_both_group_and_course_id(self, user0, student_course0, study_group0):
        message = Message(sender=user0, group=study_group0, student_course=student_course0, message='test')

        with pytest.raises(ValidationError):
            message.save()

    def test_message_save_without_group_and_course_id(self):
        message = Message(message='test')

        with pytest.raises(ValidationError):
            message.save()

    def test_message_save_with_group_invalid_sender(self, study_group0, user_without_group_and_course):
        message = Message(sender=user_without_group_and_course, group=study_group0, message='test')

        with pytest.raises(ValidationError):
            message.save()

    def test_message_save_with_student_course_invalid_sender(self, student_course0, user_without_group_and_course):
        message = Message(sender=user_without_group_and_course, student_course=student_course0, message='test')

        with pytest.raises(ValidationError):
            message.save()
```

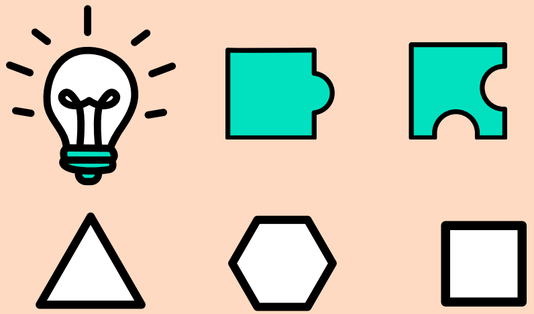
Tests - Message

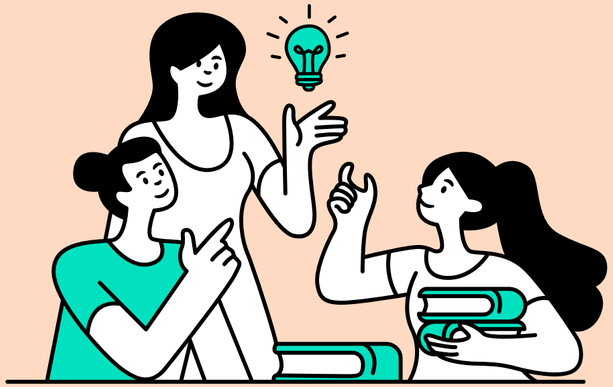
```python
@pytest.mark.django_db
class TestChatManager:
    def test_get_group_chat(self, message_with_group):
        message_with_group.save()

        group_messages = Message.objects.get_group_chat(group_id=message_with_group.group.study_group_id)

        assert message_with_group in group_messages

    def test_get_group_chat_different_group_id(self, message_with_group):
        message_with_group.save()

        group_messages = Message.objects.get_group_chat(group_id=message_with_group.group.study_group_id + 1)

        assert message_with_group not in group_messages

    def test_get_student_course_chat(self, message_with_student_course):
        message_with_student_course.save()

        course_messages = Message.objects.get_student_course_chat(
            student_course_id=message_with_student_course.student_course.student_course_id)

        assert message_with_student_course in course_messages
```

Git Repository

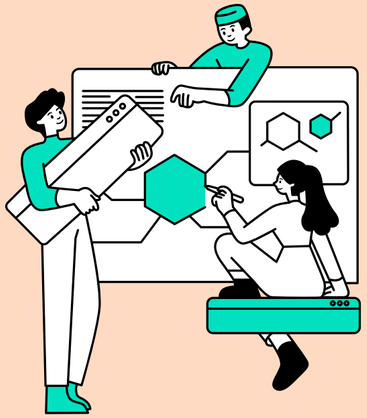
https://github.com/redhat-beyond/HighTeach







The road to success has never been easier...



Thank You!

Do you have any question?