# HƯỚNG DẪN DJANGO

Toàn bộ code được lưu tại <https://github.com/aten040791/django-core>

## Cài đặt Python

Cài đặt Python tại trang chủ Python: <https://www.python.org/downloads/>

Trong Python hiện tại đã có sẵn pip (Package dependencies) của Python, tương tự npm của Nodejs hay Composer của PHP

Sau khi cài đặt, kiểm tra Python và pip đã được cài đặt thành công bằng câu lệnh

|  |
| --- |
| python3 -V |

Và

|  |
| --- |
| pip3 –version |

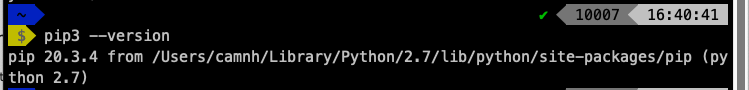


Figure 1: PIP3 version

## Cài đặt Django

Để chạy Django, cần cài đặt venv. Venv là một công cụ tạo ra môi trường ảo (Virtual env), giúp khi cài đặt các gói trong Django sẽ lưu vào chính venv, thay vì cài đặt ở ngoài. Module này tương tự như node\_modules.

Cài đặt venv qua câu lệnh

|  |
| --- |
| python3 -m venv .venv |

Activate môi trường

|  |
| --- |
| Source .venv/bin/activate |

Trong đó .venv là tên thư mục, có thể đổi tên tùy ý, các gói sau khi được cài đặt sẽ nằm trong .venv

Cài đặt Django trên chính thư mục cài đặt .venv thông qua

|  |
| --- |
| pip install django |

Tạo project mới

|  |
| --- |
| django-admin startproject my\_app |

Trong đó, my\_app là tên project, có thể đổi tên tùy ý. Đây cũng chính là module chính trong ứng dụng Django này.

Sau khi cài đặt project có cấu trúc sau

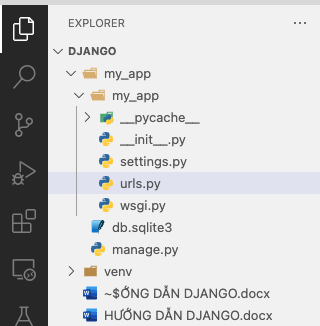


Figure 2: Cấu trúc django project

Sửa trong file **my\_app/urls.py** từ

|  |  |
| --- | --- |
| from django.urls import url |  |

thành

|  |
| --- |
| from django.urls import re\_path as url |

Khởi tạo server qua câu lệnh trên Terminal

|  |
| --- |
| python manage.py runserver |

Trang chủ Django hiện ra

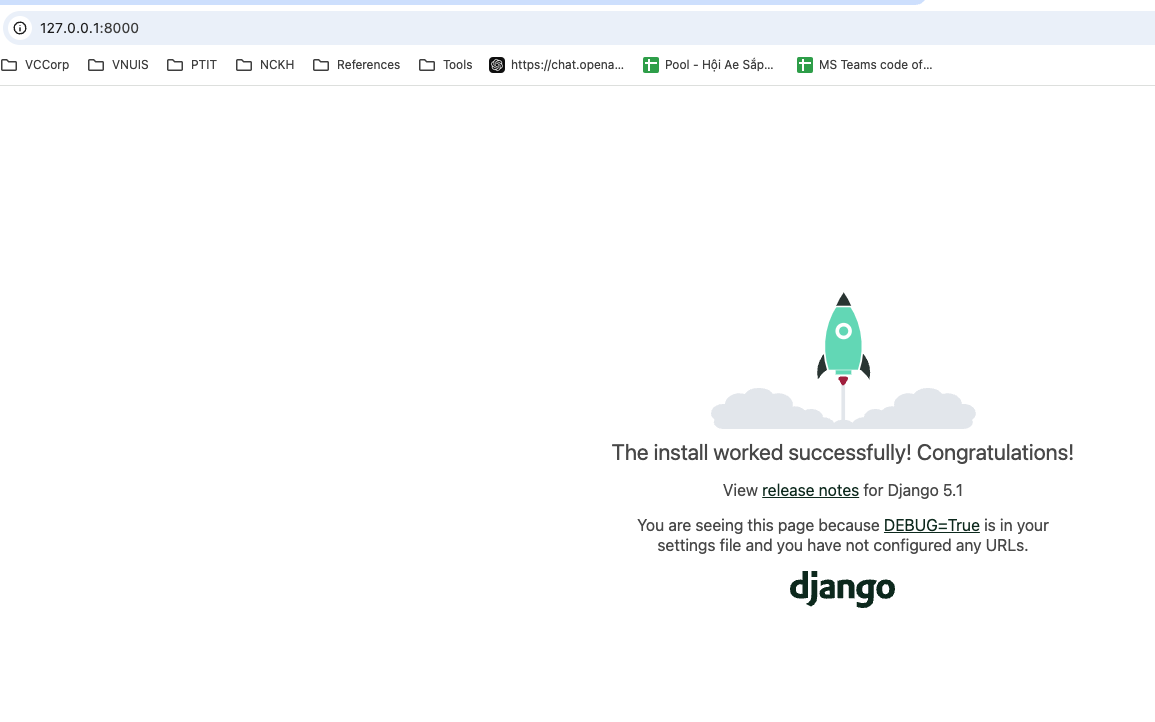
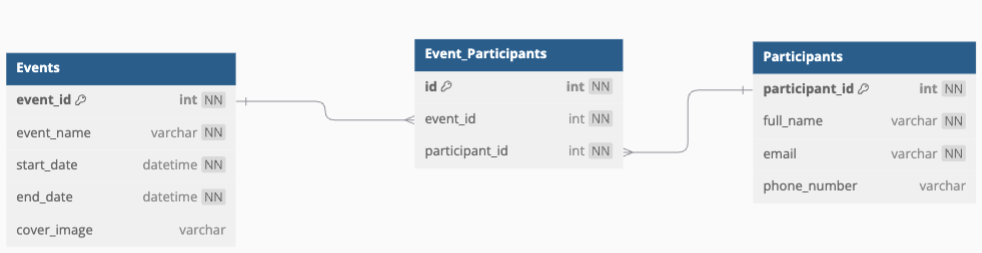


Figure 3: Kết quả

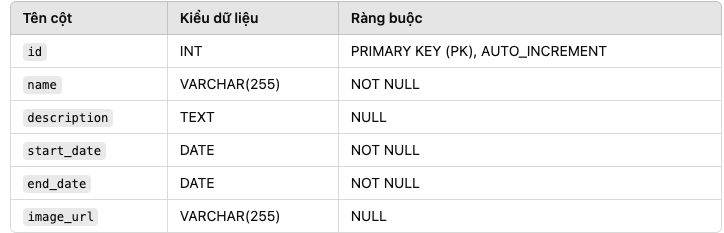
## Đặt vấn đề

Xây dựng một API để quản lý các sự kiện và người tham gia sự kiện. API cần cung cấp các chức năng sau:

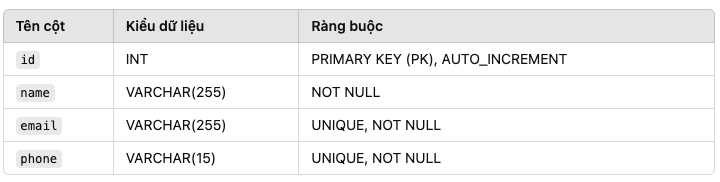


* **CRUD cho Sự kiện (Events):**
  + Thêm mới, xem chi tiết, sửa, xóa.
* **CRUD cho Người tham gia (Participants):**
  + Thêm mới, xem chi tiết, sửa, xóa.
* Quan hệ giữa Sự kiện và Người tham gia:
  + Một Sự kiện có thể có nhiều Người tham gia.
  + Một Người tham gia có thể tham gia nhiều Sự kiện.
* **Upload file:**
  + Cho phép tải lên ảnh bìa của sự kiện.
* **Xử lý ngày tháng:**
  + Lưu trữ và truy vấn các sự kiện dựa trên thời gian bắt đầu và kết thúc.

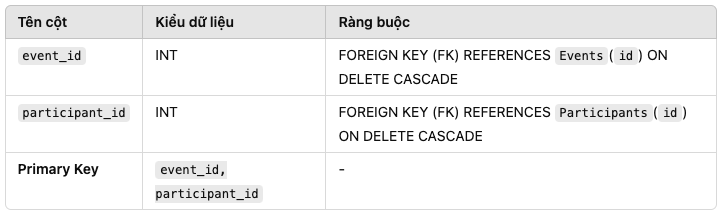
Bảng Events



Bảng Participant



Bảng Events Participant



### Xây dựng module

Bản thân Django khi triển khai đã có ý tưởng triển khai theo hướng module.Xây dựng module `event\_management` bằng câu lệnh sau

|  |
| --- |
| python3 manage.py startapp event\_management |

Đổi tên module `event\_management` sang module khác tùy chọn

Cấu trúc sau khi tạo như sau

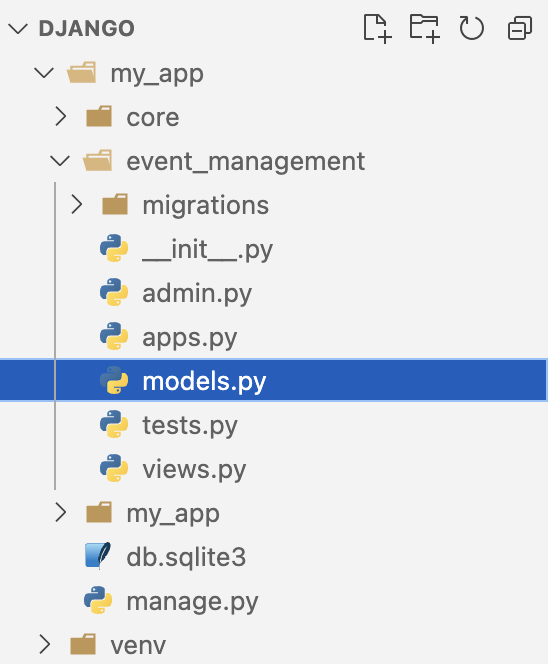


Figure 4: Cấu trúc sau khi tạo

Sau khi tạo module mới xong, tạo thử 1 view để kiểm tra luồng theo trình tự sau

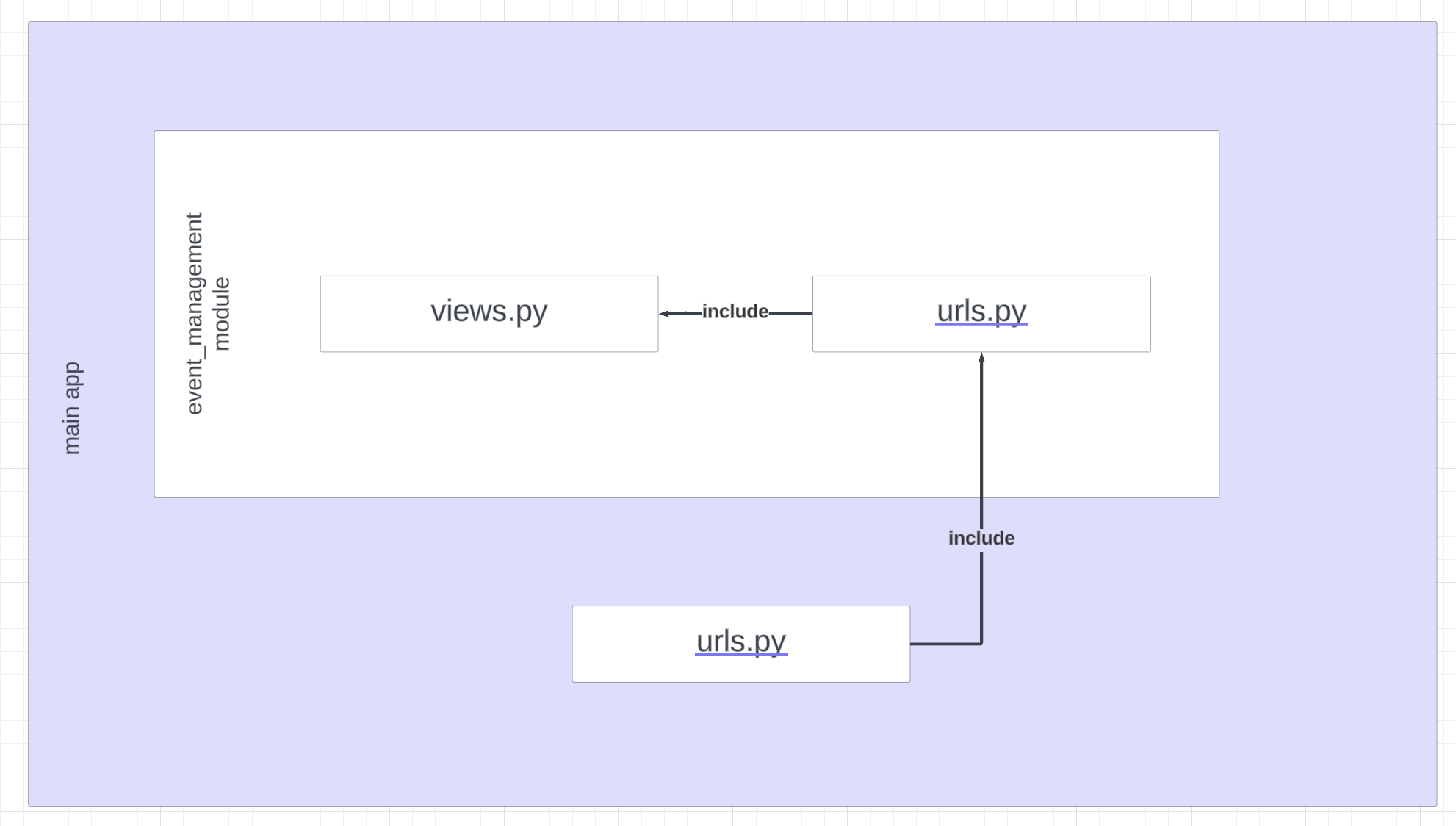


Figure 5: Luồng hoạt động chính

Event\_management/views.py

|  |
| --- |
| from django.shortcuts import render  # Create your views here.  from django.http import HttpResponse  def index(request):  return HttpResponse("Hello, world. You're at the polls index.") |

Tạo file **urls.py** trên thư mục **event\_management.** Nội dung trên file **urls.py** như sau

|  |
| --- |
| from django.urls import path  from . import views  urlpatterns = [  path("", views.index, name="index"), # index là tên hàm trong file views.py  ] |

Sau khi có view trên **event\_management/urls.py,** tiến hành nạp vào url chính như sau

|  |
| --- |
| from django.urls import path, include, re\_path as url  from django.contrib import admin  urlpatterns = [  path('event-management/', include("event\_management.urls")),  url(r'^admin/', admin.site.urls)  ] |

Với tên **event-management** là path prefix, khi nhận đường dẫn này sẽ nhận hết các đường dẫn bên trong module **event\_management.** Không có dấu “/” khi khai báo đường dẫn. Tiền tố r trước admin ám chỉ Biểu thức chính quy (Regex)

Kết quả trang như sau

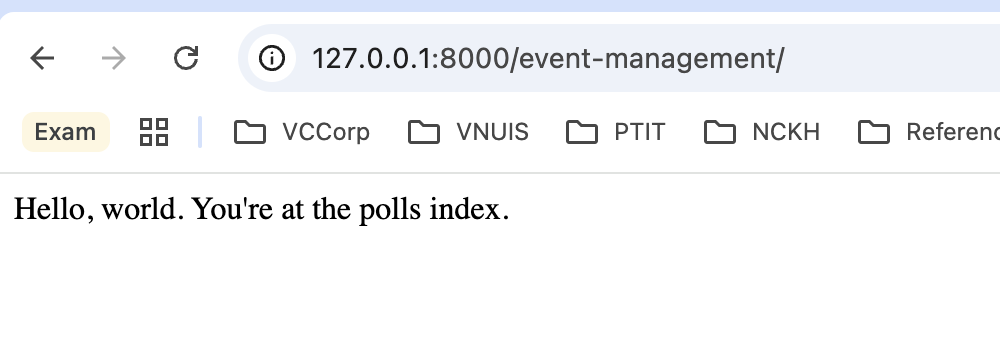
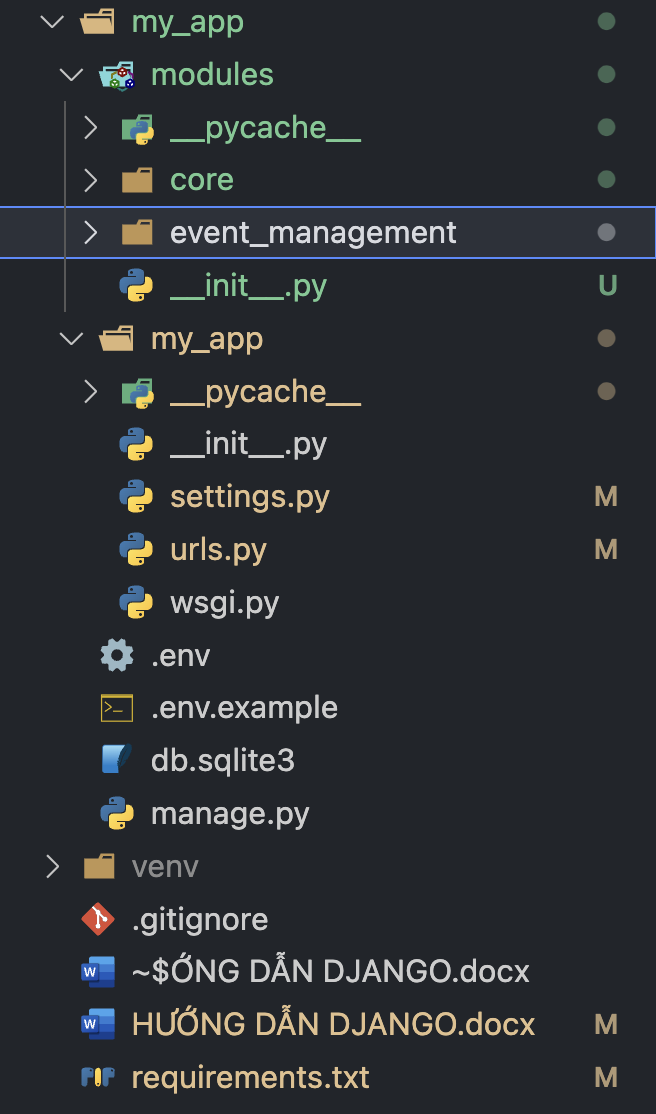


Figure 6: Kết quả

Tùy chỉnh đường dẫn thư mục: Trong trường hợp cần tạo thư mục modules chứa core và event\_management thì cần thực hiện các bước sau để ứng dụng trở lại bình thường



* Di chuyển thư mục /core và /event\_management vào thư mục /modules. Trong thư mục modules thêm file \_\_init\_\_.py để Django nhận thư mục modules là gói (packages)
* Sửa trong file my\_app/my\_app/settings.py bằng cách thêm tiền tố modules ở đầu.

|  |
| --- |
| INSTALLED\_APPS = [  'modules.event\_management.apps.EventManagementConfig',  'django.contrib.admin',  'django.contrib.auth',  'django.contrib.contenttypes',  'django.contrib.sessions',  'django.contrib.messages',  'django.contrib.staticfiles',  ] |

* Cập nhật trong module modules/event\_management/apps.py để cập nhật lại đường dẫn cho tên module

|  |
| --- |
| from django.apps import AppConfig  class EventManagementConfig(AppConfig):  default\_auto\_field = 'django.db.models.BigAutoField'  name = 'modules.event\_management' |

Cập nhật trong my\_app/urls.py để nhận tiền tố modules

|  |
| --- |
| from django.urls import path, include, re\_path as url  from django.contrib import admin  urlpatterns = [  path('event-management/', include("modules.event\_management.urls")),  url(r'^admin/', admin.site.urls)  ] |

Khởi động lại chương trình và xem lại kết quả.

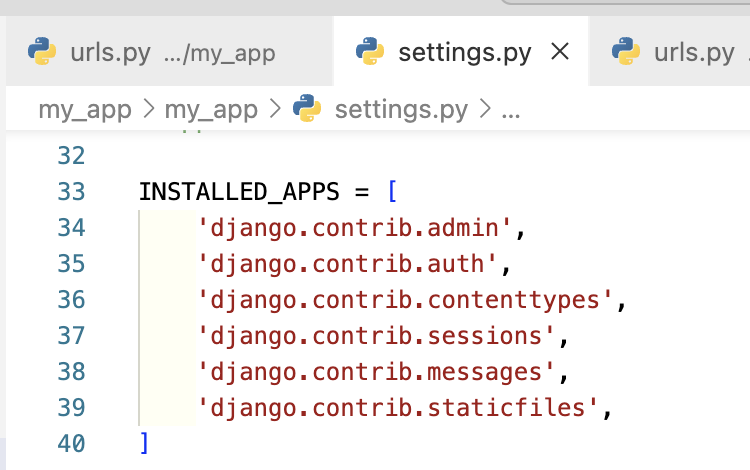
### Xây dựng migrations và cấu hình CSDL

Cơ sở dữ liệu là cần thiết cho bất kì ứng dụng nào. Django cho phép cấu hình cơ sở dữ liệu thông qua file **my\_app/settings.py.** File này viết tương tự với .env trong Laravel hay NodeJS Express. Phiên bản được khuyến nghị là Mysql version 8.x trở lên

Đầu tiên, chuyển vùng về timezone + 7 (Việt Nam)

|  |
| --- |
| TIME\_ZONE = 'Asia/Ho\_Chi\_Minh' |

Tương tự hình ở dưới là các package đã được đóng gói sẵn trong Python để sử dụng trực tiếp.



Setup CSDL Mysql trong Django cần cài đặt pymysql như sau

|  |
| --- |
| pip install pymysql |

Figure 7: Cài đặt mysqlclient

Viết vào **my\_app/\_\_init\_\_.py** để tiến hành nhận pymysql là Mysqldb.

|  |
| --- |
| import pymysql  pymysql.install\_as\_MySQLdb() |

Cuối cùng chạy câu lệnh pip freeze > requirements.txt nhằm lưu các packages vào file **requirements.txt**

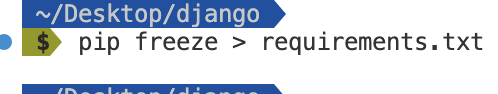


Figure 8: Pip freeze > requirements.txt

Sử dụng .env để lưu cấu hình môi trường

|  |
| --- |
| pip install python-dotenv |

Cấu hình python với .env cần chỉ định đúng đường dẫn file .env thông qua chỉ định đường dẫn BASE\_DIR với ENV\_PATH như sau

|  |
| --- |
| from dotenv import load\_dotenv  # Build paths inside the project like this: os.path.join(BASE\_DIR, ...)  BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  ENV\_PATH = os.path.join(BASE\_DIR, '.env')  load\_dotenv(ENV\_PATH) |

Nạp cấu hình khi đọc từ .env trong file **my\_app/settings.py.** Cần tạo cơ sở dữ liệu trước khi triển khai

|  |
| --- |
| DATABASES = {  'default': {  'ENGINE': 'django.db.backends.mysql',  'NAME': os.getenv('MYSQL\_DATABASE', 'XXX'),  'USER': os.getenv('MYSQL\_USERNAME', 'XXX'),  'PASSWORD': os.getenv('MYSQL\_PASSWORD', 'XXX'),  'HOST': os.getenv('MYSQL\_HOST', 'XXX'),  'PORT': os.getenv('MYSQL\_PORT', 'XXX')  }  } |

### Xây dựng model và migrations

Model trong Django được xây dựng tương tự như migrations trong Laravel. Trong model định nghĩa các trường, mối quan hệ giữa bảng trong cơ sở dữ liệu

Để sử dụng các tính năng của file (image, .docx, .pptx) cần sử dụng thư viên pillow. Thư viện này cho phép tương tác với file và ảnh một cách dễ dàng

|  |
| --- |
| pip install pillow |

Trong file **my\_app/settings.py**, tiến hành nhập module mới nhằm cho phép project chính nhận module phụ để sử dụng cho migrations

|  |
| --- |
| INSTALLED\_APPS = [  'event\_management.apps.EventManagementConfig',  'django.contrib.admin',  'django.contrib.auth',  'django.contrib.contenttypes',  'django.contrib.sessions',  'django.contrib.messages',  'django.contrib.staticfiles',  ] |

Class EventManagementConfig được tìm thấy tại **event\_management/apps.py**

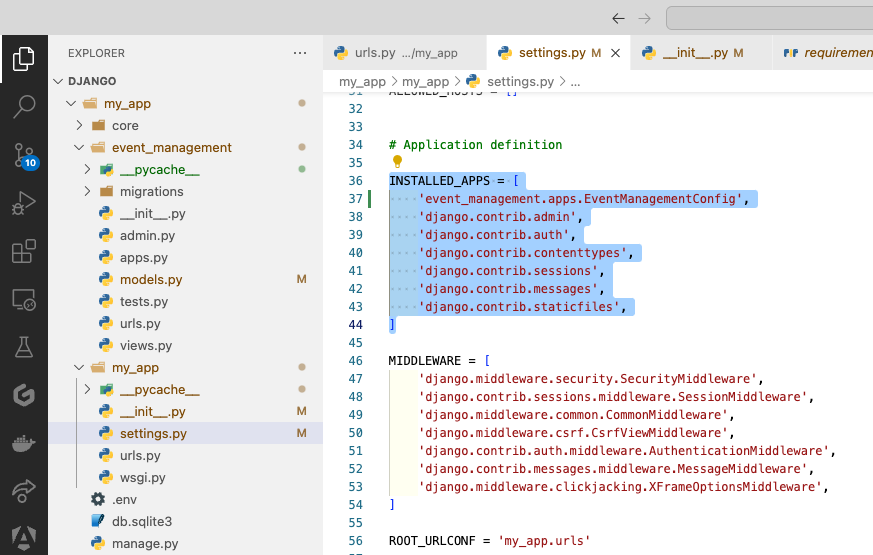


Figure 9: Import module event\_management

Django đã cung cấp sẵn một số migrations để bắt đầu. Migrations trong Django giúp làm việc với cơ sở dữ liệu thông qua code. Từ đó giúp áp dụng và quay ngược trạng thái trong CSDL dễ dàng hơn. Chạy câu lệnh để bắt đầu để áp dụng migrations mặc định vào cơ sở dữ liệu

|  |
| --- |
| python manage.py migrate |

Xây dựng model trong Python được xây dựng trong module **event\_management/models.py** như sau.

|  |
| --- |
| from django.db import models  # Create your models here.  class Event(models.Model):  event\_id = models.AutoField(primary\_key=True) # Primary Key  event\_name = models.CharField(max\_length=255) # Event name  start\_date = models.DateTimeField() # Start date  end\_date = models.DateTimeField() # End date  cover\_image = models.ImageField(upload\_to='event\_covers/', blank=True, null=True) # Cover image (optional)  class Meta:  db\_table='events'  class Participant(models.Model):  participant\_id = models.AutoField(primary\_key=True) # Primary Key  full\_name = models.CharField(max\_length=255) # Participant's full name  email = models.EmailField(unique=True) # Email (unique constraint)  phone\_number = models.CharField(max\_length=15, blank=True, null=True) # Phone number (optional)  class Meta:  db\_table='participants'  class EventParticipant(models.Model):  event = models.ForeignKey(Event, on\_delete=models.CASCADE, related\_name='participants') # Foreign Key to Event (event\_id)  participant = models.ForeignKey(Participant, on\_delete=models.CASCADE, related\_name='events') # Foreign Key to Participant (participant\_id)  class Meta:  unique\_together = ('event', 'participant') # Ensure unique pairs of Event and Participant  db\_table='event\_participant' |

Tài liệu xem Field nằm tại đây: <https://docs.djangoproject.com/en/5.1/ref/models/fields/#django.db.models.Field>

Tùy chỉnh tên bảng trong cơ sở dữ liệu bằng cách sử dụng class Meta (db\_table=…). Với khóa ngoại thì chỉ cần viết tên trường, \_id sẽ được tự động thêm vào sau. Ví dụ trong class EventParticipant như ở trên.

Sau khi đã định nghĩa model, tiếp theo tiến hành áp dụng migration trong model mới vào cơ sở dữ liệu. Mỗi lần chỉnh sửa model và chạy câu lệnh makemigrations thì được tính là 1 migration. Nói cách khác, Python quản lí migrations thông qua sự thay đổi của model.

|  |
| --- |
| python manage.py makemigrations |

Cập nhật vào CSDL dùng lệnh

|  |
| --- |
| python manage.py migrate |

Kết quả tên các bảng đã được tùy chỉnh

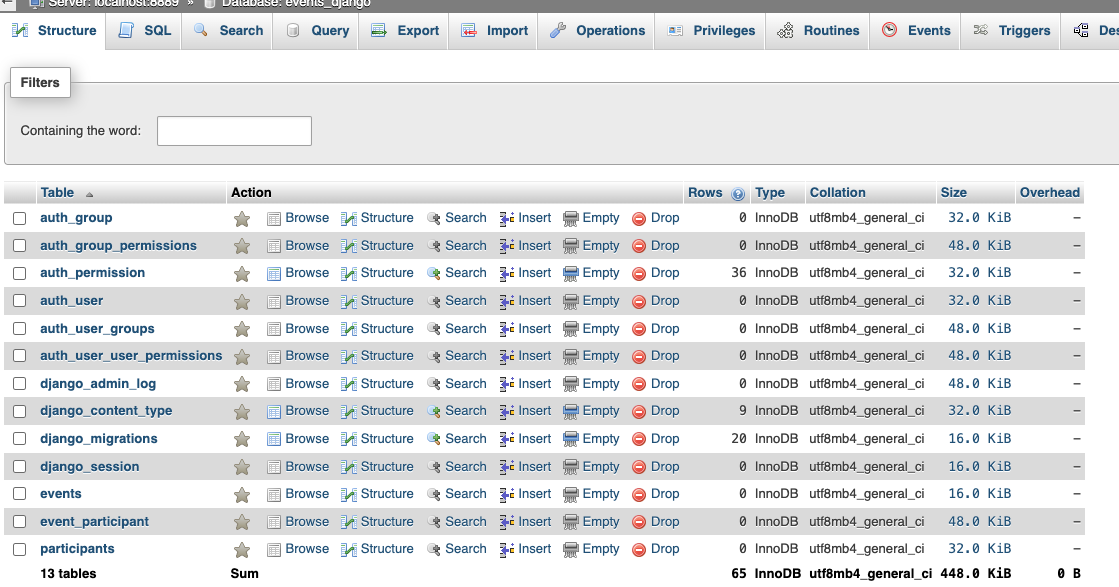


Figure 10: Tên bảng tùy chỉnh

### Xây dựng “Controller”

Trong Django, không có khái niệm controller. Thay vào đó Django theo mô hình (MTV) Model – Template – View. Controller thực chất chính là views trong Django.

Danh sách method gắn với bảng events

|  |  |  |
| --- | --- | --- |
| Method | API endpoint | Function |
| GET | /api/events | Xem danh sách events |
| POST | /api/events/store | Thêm mới events |
| PUT | /api/events/<int:event\_id> | Sửa thông tin event. |
| DELETE | /api/events/<int:event\_id> | Xóa thông tin event |

Luồng hoạt động

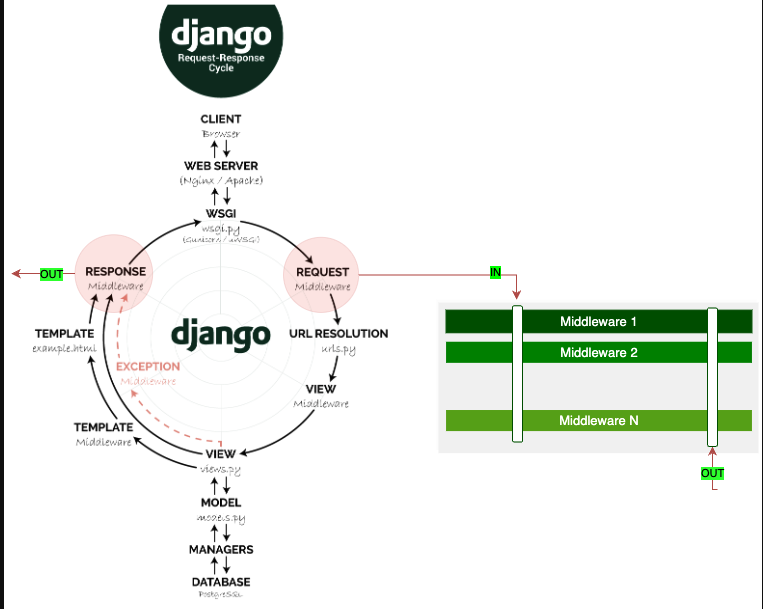


Figure 11: Django request lifecycle

### Xem danh sách events

Tạo event\_controller, import model **event\_management/models.py**

|  |
| --- |
| from django.http import JsonResponse  from ..models import Event  def all(request):  events = Event.objects.all()  return JsonResponse({  'success': True,  'data':{  'events': list(events.values())  },  'message': 'ok'  }) |

Khi import trong Django, bản chất là import hàm, dấu ..models tương đương với ../ trong NodeJS. Hàm list() sẽ trả về một mảng, tương đương array trong Javascript. Hàm events.values() tương đương SELECT \* FROM events. Nếu SELECT id, name sẽ viết events.values(‘event\_id’,’event\_name’,…)

File **event\_management/urls.py**

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  from . import views  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show)  ] |

File **my\_app/urls.py**

|  |
| --- |
| from django.urls import path, include, re\_path as url  from django.contrib import admin  urlpatterns = [  path('event-management/', include("event\_management.urls")),  url(r'^admin/', admin.site.urls)  ] |

Test api

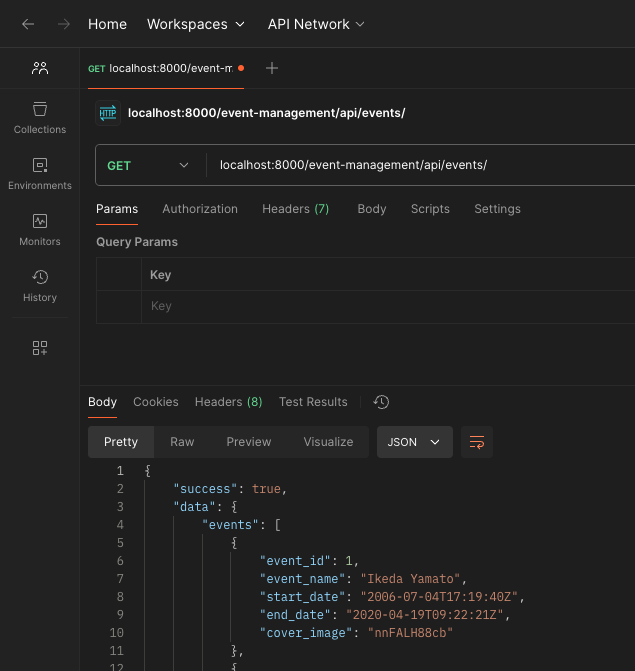


Figure API12: Test api

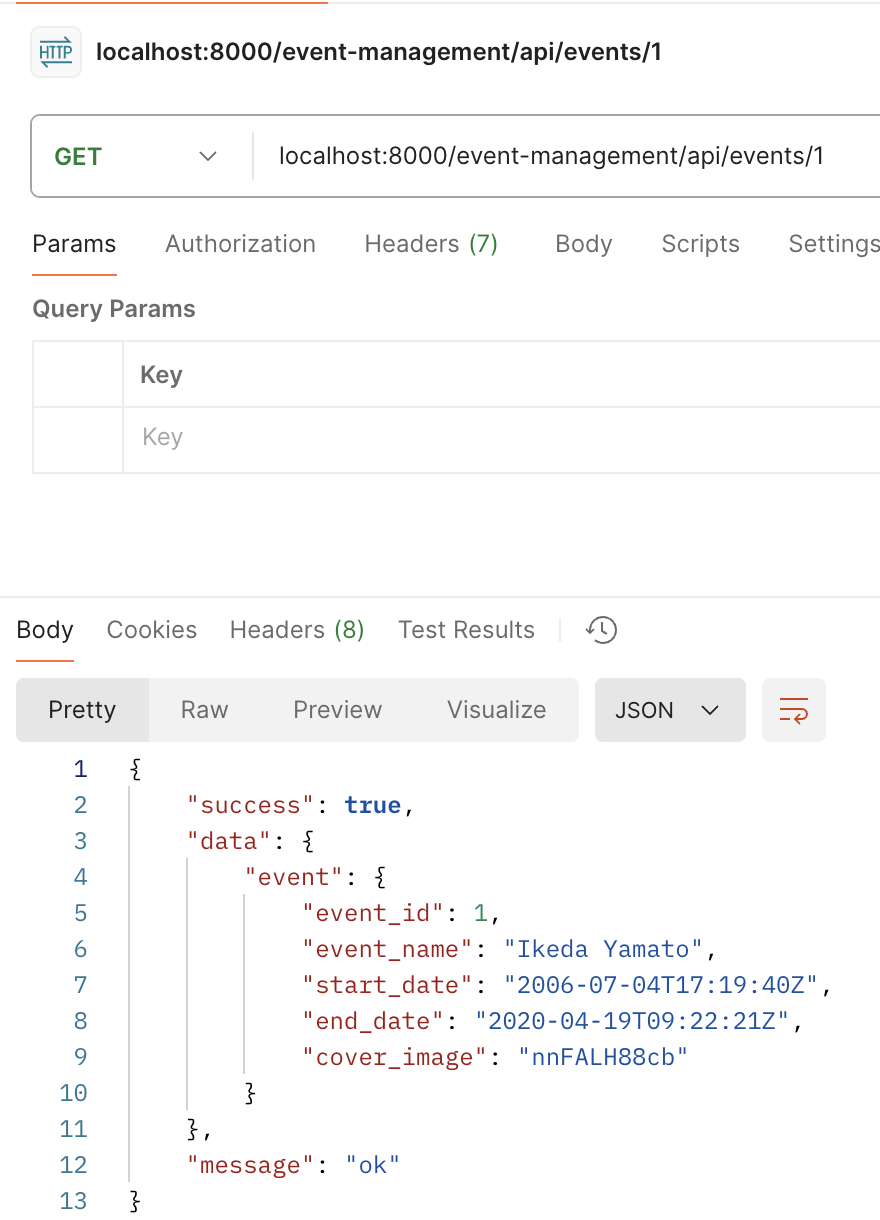
Tiếp theo xử lí url params. Để xử lí url params cần thêm param vào chính url trên path và có kiểu dữ liệu như bảng dưới.

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  from . import views  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show)  ] |

Tham số event\_id ứng với event\_id trong hàm show()

|  |
| --- |
| from django.http import JsonResponse  from django.forms.models import model\_to\_dict  from ..models import Event  def all(request):  events = Event.objects.all()  return JsonResponse({  'success': True,  'data':{  'events': list(events.values())  },  'message': 'ok'  })    def show(request, event\_id):  event = Event.objects.get(event\_id=event\_id)  cover\_image = event.cover\_image.name  # Chuyển sang dạng dict (key:value)  event = model\_to\_dict(event, exclude=['cover\_image'])  event['cover\_image'] = cover\_image  return JsonResponse({  'success': True,  'data': {  'event': event  },  'message': 'ok'  }, safe=False) |

Kết quả như sau



Xử lí với querystring sử dụng request.GET.get(‘querystring\_key’)

Trong **event\_management/urls.py** thêm route show.

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show),  path('api/events/show', event\_controller.show\_querystring)  ] |

Hàm xử lí querystring, kết quả trả về tương tự

|  |
| --- |
| def show\_querystring(request):  # Get querystring key and default value  event\_id = request.GET.get('event\_id', 'default\_value')  event = Event.objects.get(event\_id=event\_id)  cover\_image = event.cover\_image.name  # Chuyển sang dạng dict (key:value)  event = model\_to\_dict(event, exclude=['cover\_image'])  event['cover\_image'] = cover\_image  return JsonResponse({  'success': True,  'data': {  'event': event,  },  'message': 'ok'  }, safe=False) |

### Thêm mới events

* + - 1. Thêm mới events không cần validation

Với dữ liệu dạng **x-www-formurlencoded** hoặc **form-data/multipart,** sử dụng request.POST.get(‘field’) để lấy các trường hoặc sử dụng request.POST để lấy toàn bộ các trường

Với dạng dữ liệu json raw body, sử dụng json.loads(request.body) để lấy toàn bộ các trường, kết quả trả về dưới dạng **dict.** Sử dụng csrf\_exempt để không chỉ định csrf khi gửi dữ liệu do API sử dụng cơ chế xác thực đúng người dùng khác với API.

|  |  |
| --- | --- |
| Form data | JSON raw data |
| from django.views.decorators.http import require\_POST, require\_http\_methods  @require\_POST  @csrf\_exempt  def store(request):  # request.POST form data  body = request.POST  Event.objects.create(  event\_name=body.get('event\_name'),  start\_date=body.get('start\_date'),  end\_date=body.get('end\_date'),  cover\_image = body.get('cover\_image')  )  return JsonResponse({  'success': True,  'message': 'ok'  }) | from django.views.decorators.http import require\_POST, require\_http\_methods  @require\_POST  @csrf\_exempt  def store(request):  body = json.loads(request.body)  Event.objects.create(  event\_name=body.get('event\_name'),  start\_date=body.get('start\_date'),  end\_date=body.get('end\_date'),  cover\_image = body.get('cover\_image')  )  return JsonResponse({  'success': True,  'message': 'ok'  }) |

Trong trường hợp muốn lưu rất nhiều trường, không cần viết từng trường một, sử dụng Django Form

File **forms/event\_form.py**

|  |
| --- |
| from django import forms  from ..models import Event  class EventForm(forms.ModelForm):  class Meta:  model = Event  # fiilable  fields = ['event\_name', 'start\_date', 'end\_date', 'cover\_image'] |

Sử dụng trong **event\_controller.py**

|  |  |
| --- | --- |
| Form data | JSON raw data |
| from django.views.decorators.csrf import csrf\_exempt  from ..forms.event\_form import EventForm  from django.forms.models import model\_to\_dict  from django.views.decorators.http import require\_POST, require\_http\_methods  @require\_POST  @csrf\_exempt  def store\_fillable(request):  # Receive form request  form = EventForm(json.loads(request.body))  if form.is\_valid():  event = form.save()  return JsonResponse({  'success': True,  'message': 'ok',  'data': {  'event': model\_to\_dict(event, exclude=['cover\_image'])  }  })    # Get form error  return JsonResponse({  'success': False,  'data': {  'errors': form.errors  },  'message': 'validation failed'  }) | from django.views.decorators.csrf import csrf\_exempt  from ..forms.event\_form import EventForm  from django.forms.models import model\_to\_dict  from django.views.decorators.http import require\_POST, require\_http\_methods  @require\_POST  @csrf\_exempt  def store\_fillable(request):  # Receive form request  form = EventForm(json.loads(request.body))  if form.is\_valid():  event = form.save()  return JsonResponse({  'success': True,  'message': 'ok',  'data': {  'event': model\_to\_dict(event, exclude=['cover\_image'])  }  })    # Get form error  return JsonResponse({  'success': False,  'data': {  'errors': form.errors  },  'message': 'validation failed'  }) |

File **event\_management/urls.py**

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show),  path('api/events/show', event\_controller.show\_querystring),  path('api/events/store', event\_controller.store),  path('api/events/store\_fillable', event\_controller.store\_fillable),  ] |

* + - 1. Validate event (Laravel Form Request và Laravel Rule)

Document: <https://docs.djangoproject.com/en/5.1/ref/forms/validation/>

Thông thường, một ứng dụng có 2 loại validate: Validate có sẵn và validate tùy chỉnh. Validate có sẵn sẽ dùng chính các rule có sẵn trong Django nhằm validate dữ liệu. Với validate tùy chỉnh sẽ tự viết hàm để validate dữ liệu

Với validate có sẵn chỉ cần sử dụng Django Form kết hợp với loại trường cần validate (URLField, CharField, …).

Với validate tùy chỉnh cần viết hàm validate có có pháp *clean\_<field>*, ví dụ trường **end\_date** phải sau trường **start\_date** và chiều dài số kí tự của **event\_name** phải lớn hơn 3. Các rule nếu thất bại raise ra ValidationError

|  |
| --- |
| from django import forms  from ..models import Event  class EventForm(forms.ModelForm):  class Meta:  model = Event  # fiilable  fields = ['event\_name', 'start\_date', 'end\_date', 'cover\_image']    # available rule:  # event\_name = forms.URLField()  # custom rule:  def clean\_event\_name(self):  event\_name = self.cleaned\_data.get('event\_name')  if len(event\_name) < 3:  raise forms.ValidationError('message.minlength:3')  return event\_name    def clean\_end\_date(self):  end\_date = self.cleaned\_data.get('end\_date')  start\_date = self.cleaned\_data.get('start\_date')  if end\_date < start\_date:  raise forms.ValidationError('end\_date must greater than start\_date')  return end\_date |

File url tương tự

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show),  path('api/events/show', event\_controller.show\_querystring),  path('api/events/store', event\_controller.store),  path('api/events/store\_fillable', event\_controller.store\_fillable),  path('api/events/<int:event\_id>/update', event\_controller.update)  ] |

Sử dụng trong **event\_controller.py**

|  |  |
| --- | --- |
| Form data | JSON raw data |
| from django.views.decorators.csrf import csrf\_exempt  from ..forms.event\_form import EventForm  from django.forms.models import model\_to\_dict  from django.views.decorators.http import require\_POST, require\_http\_methods  @require\_POST  @csrf\_exempt  def store\_fillable(request):  # Receive form request  form = EventForm(json.loads(request.body))  if form.is\_valid():  event = form.save()  return JsonResponse({  'success': True,  'message': 'ok',  'data': {  'event': model\_to\_dict(event, exclude=['cover\_image'])  }  })    # Get form error  return JsonResponse({  'success': False,  'data': {  'errors': form.errors  },  'message': 'validation failed'  }) | from django.views.decorators.csrf import csrf\_exempt  from ..forms.event\_form import EventForm  from django.forms.models import model\_to\_dict  from django.views.decorators.http import require\_POST, require\_http\_methods  @require\_POST  @csrf\_exempt  def store\_fillable(request):  # Receive form request  form = EventForm(json.loads(request.body))  if form.is\_valid():  event = form.save()  return JsonResponse({  'success': True,  'message': 'ok',  'data': {  'event': model\_to\_dict(event, exclude=['cover\_image'])  }  })    # Get form error  return JsonResponse({  'success': False,  'data': {  'errors': form.errors  },  'message': 'validation failed'  }) |

Kết quả

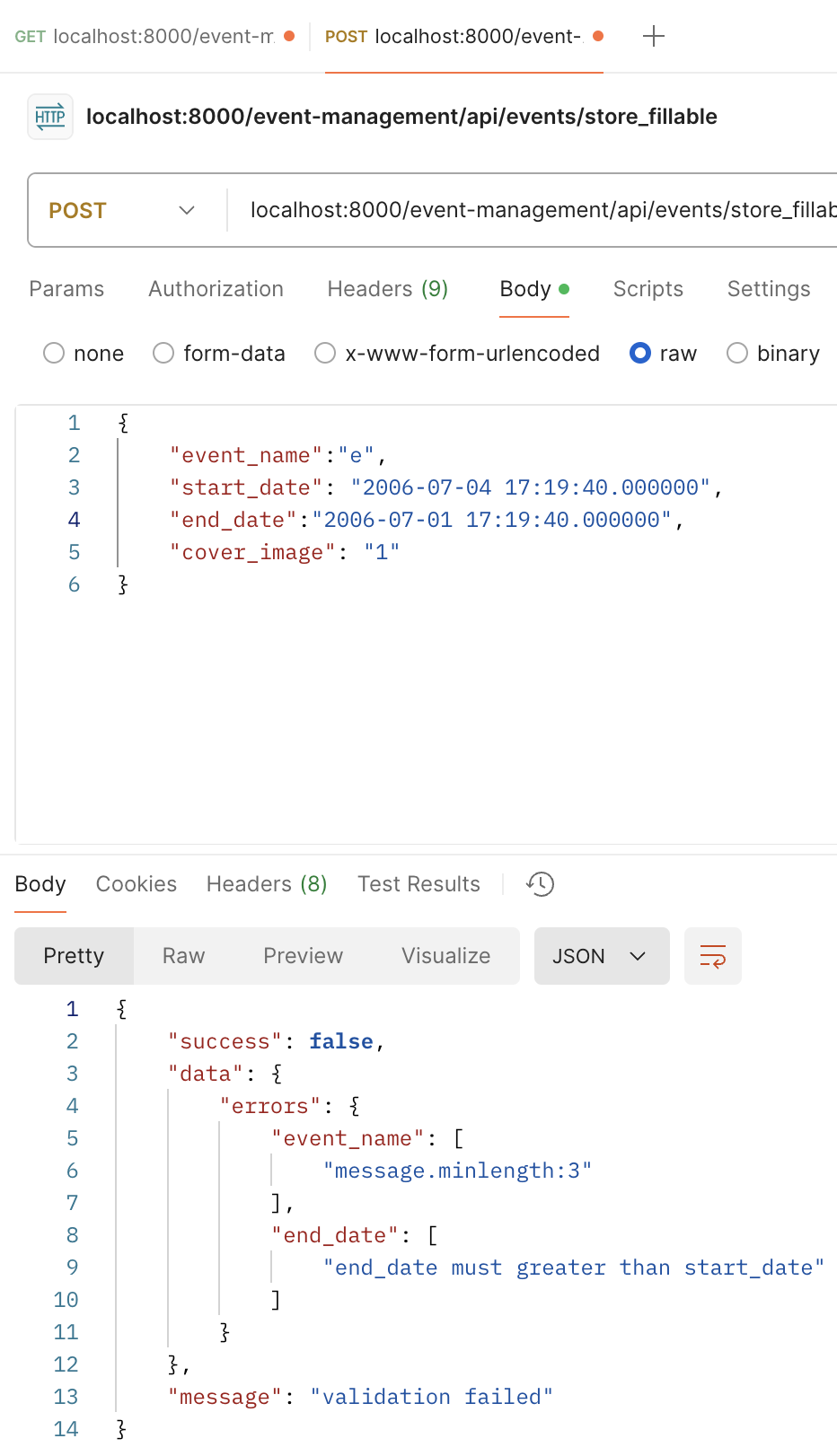


Figure 13: : Validation failed

### Cập nhật events

Trong Django, nếu như model không tìm thấy, thay vì kiểm tra bằng If else thông thường, sử dụng try…except… để kiểm tra trường hợp Model instance không tìm thấy

File **event\_management/controllers/event\_controller.py** xử lí dữ liệu cần cập nhật dưới dạng JSON.

|  |
| --- |
| from django.views.decorators.csrf import csrf\_exempt  from django.views.decorators.http import require\_POST, require\_http\_methods  from ..forms.event\_form import EventForm  @require\_http\_methods(["PUT"])  @csrf\_exempt  def update(request, event\_id):  try:  event = Event.objects.get(event\_id = event\_id)  except Event.DoesNotExist:  return JsonResponse({  'message': 'Model not found',  'success': False  }, status=404)    form = EventForm(json.loads(request.body))  if form.is\_valid():  event = form.save()  return JsonResponse({  'success': True,  'message': 'ok',  'data': {  'event': model\_to\_dict(event, exclude=['cover\_image'])  }  })  return JsonResponse({  'success': False,  'data': {  'errors': form.errors  },  'message': 'Cannot save model'  }) |

**Event\_management/urls.py**

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show),  path('api/events/show', event\_controller.show\_querystring),  path('api/events/store', event\_controller.store),  path('api/events/store\_fillable', event\_controller.store\_fillable),  path('api/events/<int:event\_id>/update', event\_controller.update)  ] |

### Xóa events

File **event\_management/controllers/event\_controller.py** xử lí dữ liệu cần cập nhật dưới dạng JSON.

|  |
| --- |
| @require\_http\_methods(["DELETE"])  @csrf\_exempt  def delete(request, event\_id):  try:  event = Event.objects.get(event\_id = event\_id)  event.delete()  return JsonResponse({  'success': True,  'message': 'ok'  })  except Event.DoesNotExist:  return JsonResponse({  'message': 'Model not found',  'success': False  }, status=404) |

**Event\_management/urls.py**

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show),  path('api/events/show', event\_controller.show\_querystring),  path('api/events/store', event\_controller.store),  path('api/events/store\_fillable', event\_controller.store\_fillable),  path('api/events/<int:event\_id>/update', event\_controller.update),  path('api/events/<int:event\_id>/delete', event\_controller.delete),  ] |

### Relationship

Relationship trong Django được định nghĩa qua các hàm OneToOne(), ManyToMany() hoặc ForeignKey (One-to-Many)

|  |  |  |  |
| --- | --- | --- | --- |
| One-to-one | One-to-many | Many-to-many | Many-to-many  pivot table (trong trường hợp bảng pivot có nhiều trường phụ) |
| **class** **Place**(  models.Model  ):  name= models.  CharField(  max\_length=50)  address = models.  CharField(  max\_length=80)  **class** **Restaurant**(models.Model):  place = models.OneToOneField(Place,  on\_delete=  models.CASCADE,  primary\_key=  True)å  serves\_hot\_dogs = models.  BooleanField(  default=**False**)  serves\_pizza = models.  BooleanField(  default=**False**) | **class** **Reporter**(  models.Model):  first\_name = models.  CharField(max\_length=30)  last\_name = models.  CharField(max\_length=30)  email = models.EmailField()  **class** **Article**(  models.Model):  headline = models.  CharField(max\_length=100)  pub\_date = models.  DateField()  reporter = models.  ForeignKey(  Reporter, on\_delete=  models.CASCADE) | **class** **Publication**(  models.Model):  title = models.  CharField(  max\_length=30)  **class** **Article**(  models.Model):  headline = models.  CharField(max\_length=100)  publications = models.  ManyToManyField(Publication) | **class Event**(models.Model):  event\_id = models.  AutoField(primary\_key=True) # Primary Key  event\_name = models.  CharField(max\_length=100) # Event name  start\_date = models.  DateTimeField()  end\_date = models.  DateTimeField() # End date  cover\_image = models.  ImageField(upload\_to='event\_covers/', blank=True, null=True) # Cover image (optional)  **class Participant** (models.Model):  participant\_id = models.AutoField(primary\_key=True)  full\_name = models.CharField(max\_length=255)  email = models.EmailField  (unique=True)  phone\_number = models.CharField(max\_length=15, blank=True, null=True)    **class EventParticipant** (models.Model):  event = models.  ForeignKey(Event, on\_delete=models.CASCADE, related\_name='participants')  participant = models.ForeignKey(Participant, on\_delete=models.CASCADE, related\_name='events') |

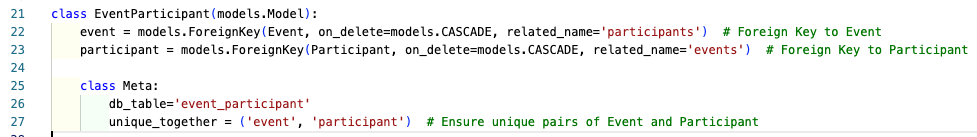
Bắt đầu với **urls.py**

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show),  path('api/events/show', event\_controller.show\_querystring),  path('api/events/store', event\_controller.store),  path('api/events/store\_fillable', event\_controller.store\_fillable),  path('api/events/<int:event\_id>/update', event\_controller.update),  path('api/events/<int:event\_id>/delete', event\_controller.delete),  path('api/events/<int:event\_id>/relationship', event\_controller.show\_with\_relationship)  ] |

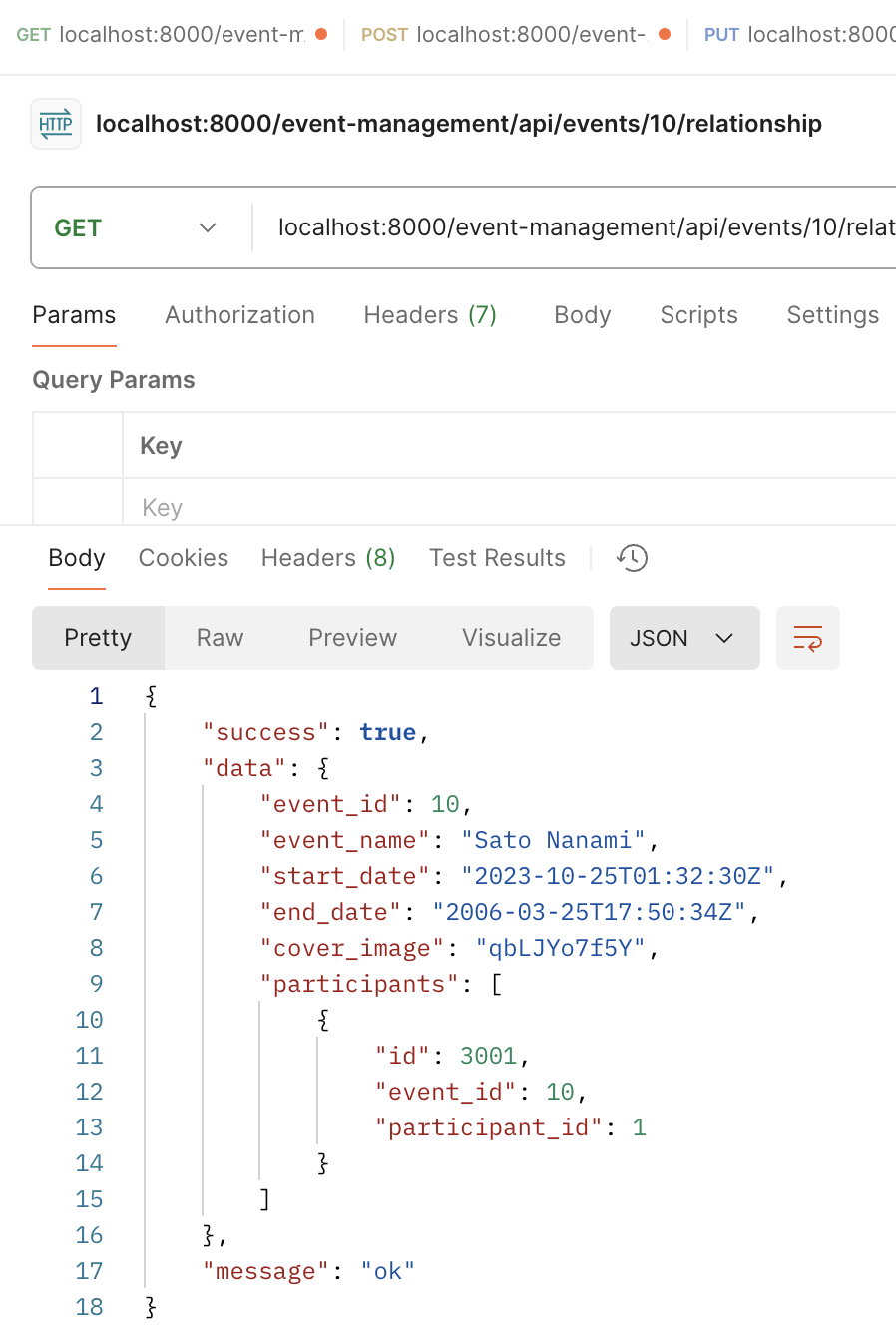
Trong event\_controller định nghĩa hàm show\_with\_relationship như sau

|  |
| --- |
| def show\_with\_relationship(request, event\_id):  try:  event = Event.objects.get(event\_id=event\_id)  cover\_image = event.cover\_image.name  event\_data = model\_to\_dict(event, exclude=[''])  event\_data['cover\_image'] = cover\_image  # Relationship .participants  event\_data['participants'] = list(event.participants.all().values())  return JsonResponse({  'success': True,  'data': event\_data,  'message': 'ok'  })  except Event.DoesNotExist:  return JsonResponse({'success': False, 'message': 'Event not found'}, status=404) |

Event.participants.all() được lấy từ **related\_name** trong models



Kết quả



### File storage

Django cung cấp một hệ thống **File Storage API** mạnh mẽ để xử lý các tệp trong ứng dụng web. Tệp trong Django thường được sử dụng để lưu trữ hình ảnh, tài liệu, file tải lên từ người dùng hoặc các file media phục vụ cho ứng dụng.

Tương tự vẫn sử dụng file **urls.txt** dưới đây

|  |
| --- |
| from django.urls import path  from .controllers import event\_controller  urlpatterns = [  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show),  path('api/events/show', event\_controller.show\_querystring),  path('api/events/store', event\_controller.store),  path('api/events/store\_fillable', event\_controller.store\_fillable),  path('api/events/<int:event\_id>/update', event\_controller.update),  path('api/events/<int:event\_id>/delete', event\_controller.delete),  path('api/events/<int:event\_id>/relationship', event\_controller.show\_with\_relationship)  ] |

Sửa migrations thêm disk: Chỉ định disk lưu trữ tương tự như Laravel (Aws s3, local hoặc Azure) và tiến hành migrate. Với local sẽ lưu trên project hiện tại

|  |
| --- |
| from django.db import models  from django.contrib.auth.hashers import check\_password, make\_password  # Create your models here.  class Event(models.Model):  event\_id = models.AutoField(primary\_key=True) # Primary Key  event\_name = models.CharField(max\_length=100) # Event name  start\_date = models.DateTimeField() # Start date  end\_date = models.DateTimeField() # End date  cover\_image = models.ImageField(upload\_to='modules/event\_management/storage/event\_covers/', blank=True, null=True) # Cover image (optional)  cover\_disk\_storage=models.CharField(max\_length=20, null=False, default="local")    class Meta:  db\_table='events' |

Cách 1 (Sử dụng nhanh): Sử dụng **FileSystemStorage** để chỉ định đường dẫn lưu, hàm **event\_controller.store()** cập nhật khu vực lưu file và đường dẫn base\_url để lấy file ra ngoài, sau đó gọi hàm .save() để tiến hành lưu vào vào đường dẫn được chỉ định

|  |
| --- |
| @require\_POST  @csrf\_exempt  def store(request):  # request.POST form data  body = request.POST  # Upload file  cover\_image = request.FILES['cover\_image']  # Upload file - Specify path  file\_storage = FileSystemStorage(location='modules/event\_management/storage/event\_covers', base\_url='event\_management/event\_covers')  file\_storage.save(cover\_image.name, cover\_image) # cover\_image.name can change to any string file name (e.g. haha.jpg)  # json.loads(request.body) for JSON data  event = Event.objects.create(  event\_name= body.get('event\_name'),  start\_date=body.get('start\_date'),  end\_date=body.get('end\_date'),  cover\_image= cover\_image.name  )  event\_dict = model\_to\_dict(event, exclude=['cover\_image'])  # Get storage url  event\_dict['cover\_image'] = file\_storage.url(cover\_image)  return JsonResponse({  'success': True,  'data': {  'event': event\_dict  },  'message': 'ok'  }) |

Cách trên có ưu điểm nhanh, gọn; tuy nhiên phải viết đường dẫn bằng tay ở mọi chỗ. Do đó sử dụng form có lợi thế validate file và chỉ định đường dẫn khi lưu ngay trong model.

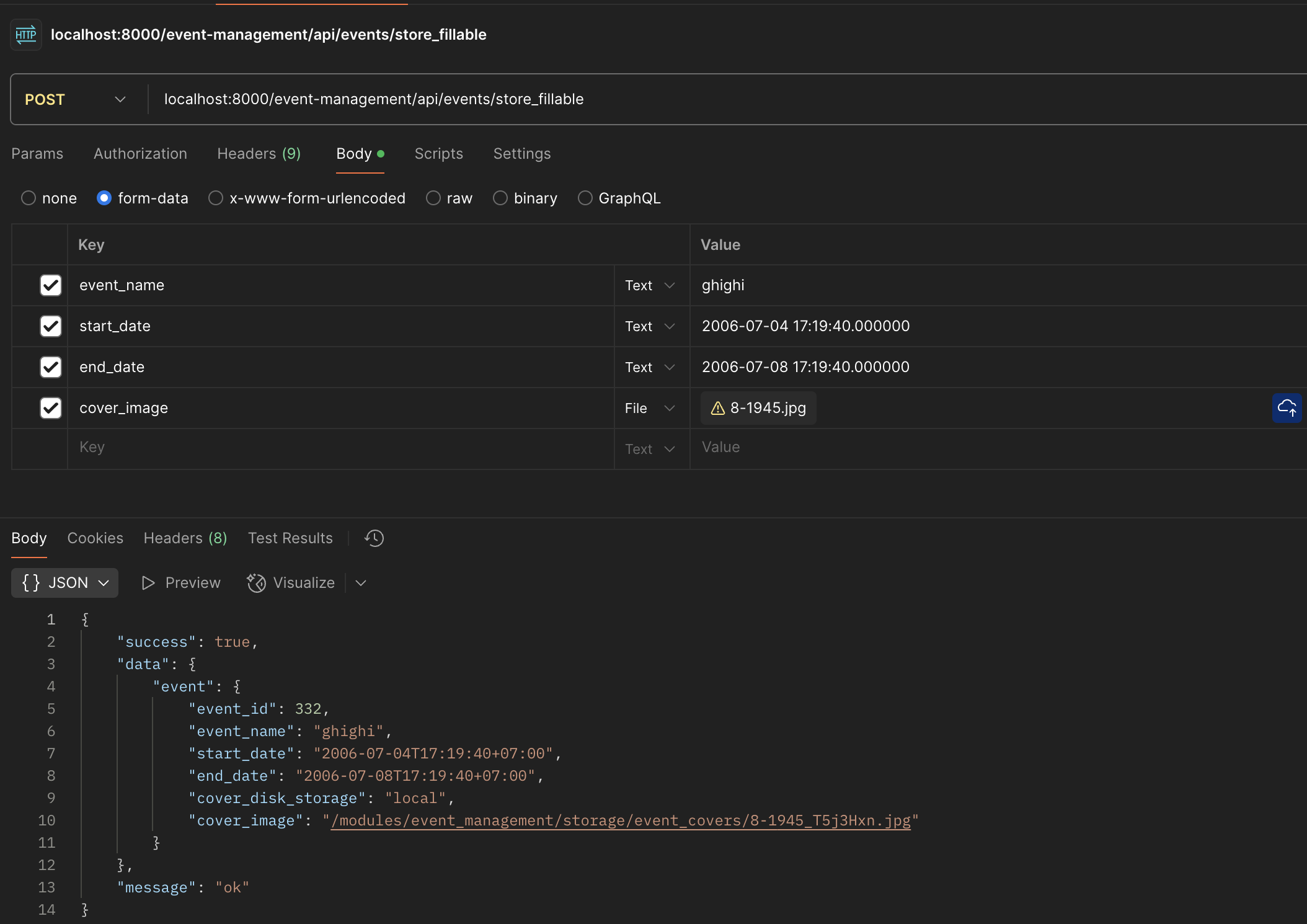
Cập nhật lại **models.py**

|  |
| --- |
| from django.db import models  from django.contrib.auth.hashers import check\_password, make\_password  # Create your models here.  class Event(models.Model):  event\_id = models.AutoField(primary\_key=True) # Primary Key  event\_name = models.CharField(max\_length=100) # Event name  start\_date = models.DateTimeField() # Start date  end\_date = models.DateTimeField() # End date  cover\_image = models.ImageField(upload\_to='modules/event\_management/storage/event\_covers/', blank=True, null=True) # Cover image (optional)  cover\_disk\_storage=models.CharField(max\_length=20, null=False, default="local")    class Meta:  db\_table='events' |

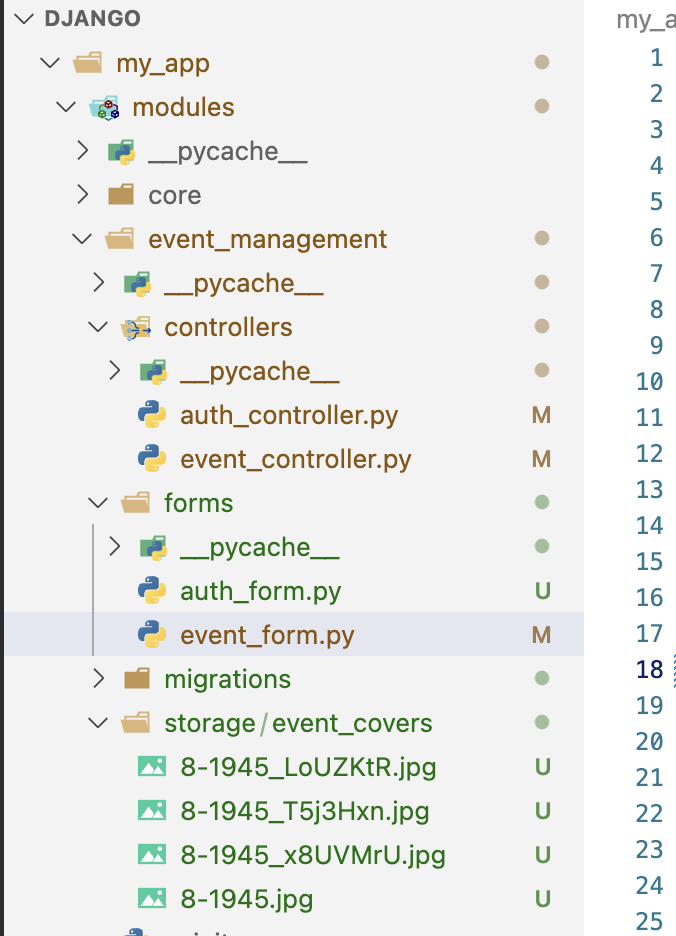
Viết lại file **event\_controller.py.** Khi gọi form.save(commit=True), file sẽ tự động upload vào đường dẫn upload\_to trong model.

|  |
| --- |
| @require\_http\_methods("POST")  @csrf\_exempt  def store\_fillable(request):  # Receive form request  form = EventForm(request.POST, files=request.FILES)  if form.is\_valid():  event = form.save(commit=True)  event\_dict = model\_to\_dict(event, exclude=['cover\_image'])  event\_dict['cover\_image'] = event.cover\_image.url  return JsonResponse({  'success': True,  'data': {  'event': event\_dict  },  'message': 'ok',  })  # Get form error  return JsonResponse({  'success': False,  'data': {  'errors': form.errors  },  'message': 'validation failed'  }) |

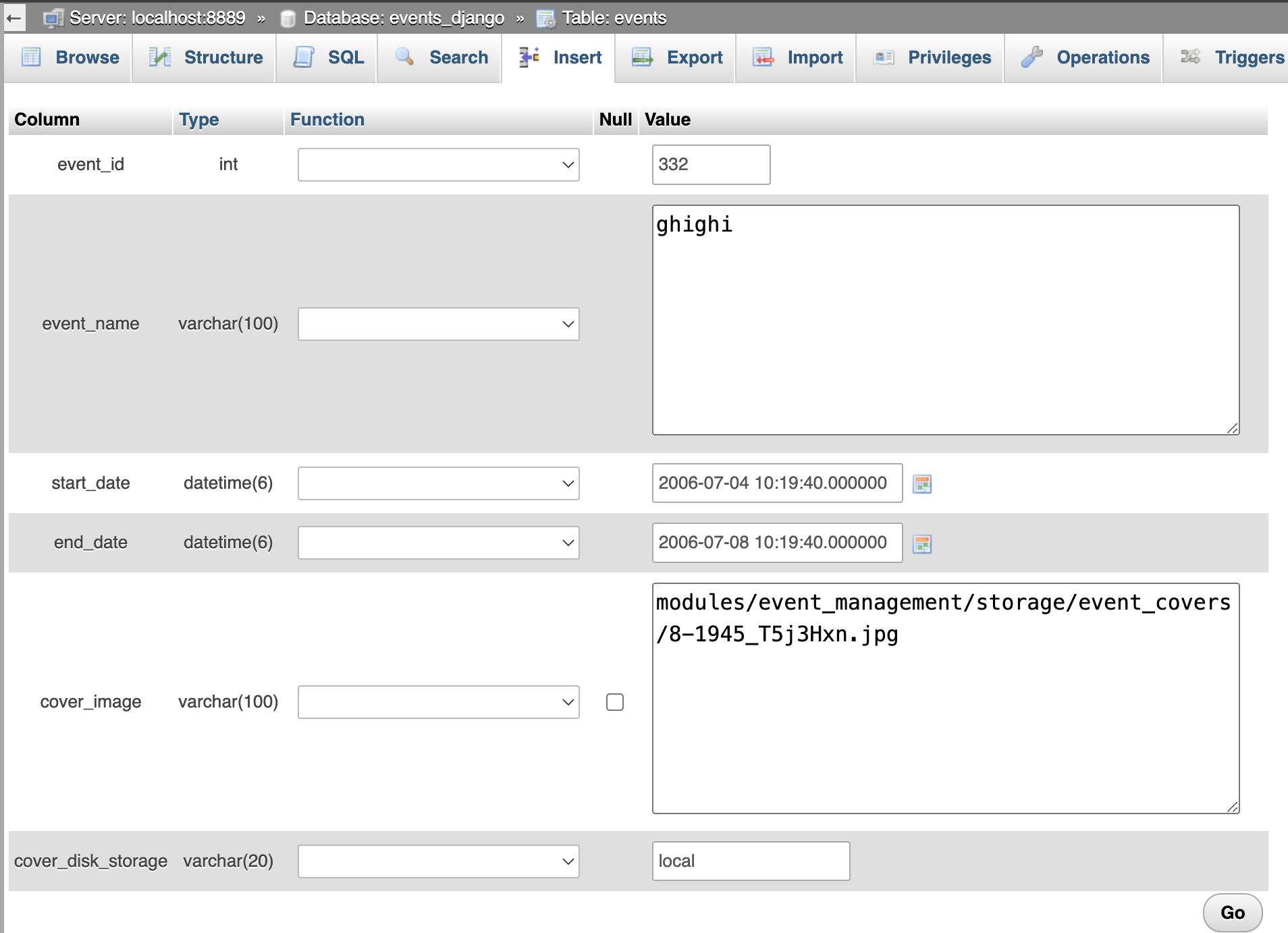
Kết quả thu được



File được lưu trong đường dẫn trong thư mục modules/event\_management/storage/event\_covers



Trong CSDL thì bản ghi id = 332 được lưu như sau



### Auth api

Sử dụng thư viện PyJWT để tiến hành sử dụng JWT đăng nhập

|  |
| --- |
| pip install pyjwt |

Tạo route */api/auth/register* và /*api/auth/login* trong file **urls.py**

|  |
| --- |
| from django.urls import path  from .controllers import auth\_controller, event\_controller  urlpatterns = [  path('api/auth/login', auth\_controller.login),  path('api/auth/register', auth\_controller.register),  path('api/events/', event\_controller.all),  path('api/events/<int:event\_id>', event\_controller.show),  path('api/events/show', event\_controller.show\_querystring),  path('api/events/store', event\_controller.store),  path('api/events/store\_fillable', event\_controller.store\_fillable),  path('api/events/<int:event\_id>/update', event\_controller.update),  path('api/events/<int:event\_id>/delete', event\_controller.delete),  path('api/events/<int:event\_id>/relationship', event\_controller.show\_with\_relationship)  ] |

Hàm register() được viết trong **auth\_controller.py,** sử dụng thư viện jwt để tiến hành mã hóa payload qua hàm jwt.encode(payload,key, algorithm).

|  |
| --- |
| from django.http import JsonResponse  import jwt, json  from ..forms.auth\_form import AuthLoginForm, AuthRegisterForm  from my\_app import settings  import datetime  from ..models import AuthUserApi  def register(request):  form = AuthRegisterForm(json.loads(request.body))  if form.is\_valid():  api\_user = form.save(commit=True) # Save to database directly  current\_time = datetime.datetime.now(datetime.timezone.utc)  jwt\_access\_token = jwt.encode(  payload={  'data': {  'id': api\_user.id,  'email': api\_user.email  },  'iat': current\_time,  'exp': current\_time + datetime.timedelta(minutes=5), # 5 minutes  }, key=settings.JWT\_SECRET, algorithm="HS256")    jwt\_refresh\_token = jwt.encode(  payload={  'data': {  'id': api\_user.id,  'email': api\_user.email  },  'iat': current\_time,  'exp': current\_time + datetime.timedelta(days=7), # 7 days    }, key=settings.JWT\_SECRET, algorithm="HS256")  return JsonResponse({  'status': 200,  'data': {  'access\_token': jwt\_access\_token,  'refresh\_token': jwt\_refresh\_token,  },  'message': 'ok',  'success': True  }, status=200)    return JsonResponse({  'success': False,  'data': {  'errors': form.errors  },  'message': 'Cannot register'  }, status=522) |

Class **AuthRegisterForm.py**

|  |
| --- |
| from django import forms  from ..models import AuthUserApi  # This validation goes directly to model  class AuthRegisterForm(forms.ModelForm):  class Meta:  model = AuthUserApi  # fillable  fields = ['email', 'password'] |

Chú ý khi đăng kí, mật khẩu cần phải được mã hóa, thay vì mã hóa trong controller thì mã hóa trong models để tránh việc mã hóa mật khẩu nhiều lần ở nhiều nơi khác nhau khi làm việc với model User bằng việc viết hàm save() trực tiếp vào model. Phương pháp này tương tự với mutator setPassword() trong Laravel.

Trước hết cần cài đặt thư viện bcrypt để tiến hành mã hóa

|  |
| --- |
| pip install bcrypt |

Thêm đoạn sau vào my\_app/settings.py để thêm phương thức mã hóa khi sử dụng hàm make\_password()

|  |
| --- |
| PASSWORD\_HASHERS = [  "django.contrib.auth.hashers.BCryptSHA256PasswordHasher"  ] |

Kết quả thu được như sau

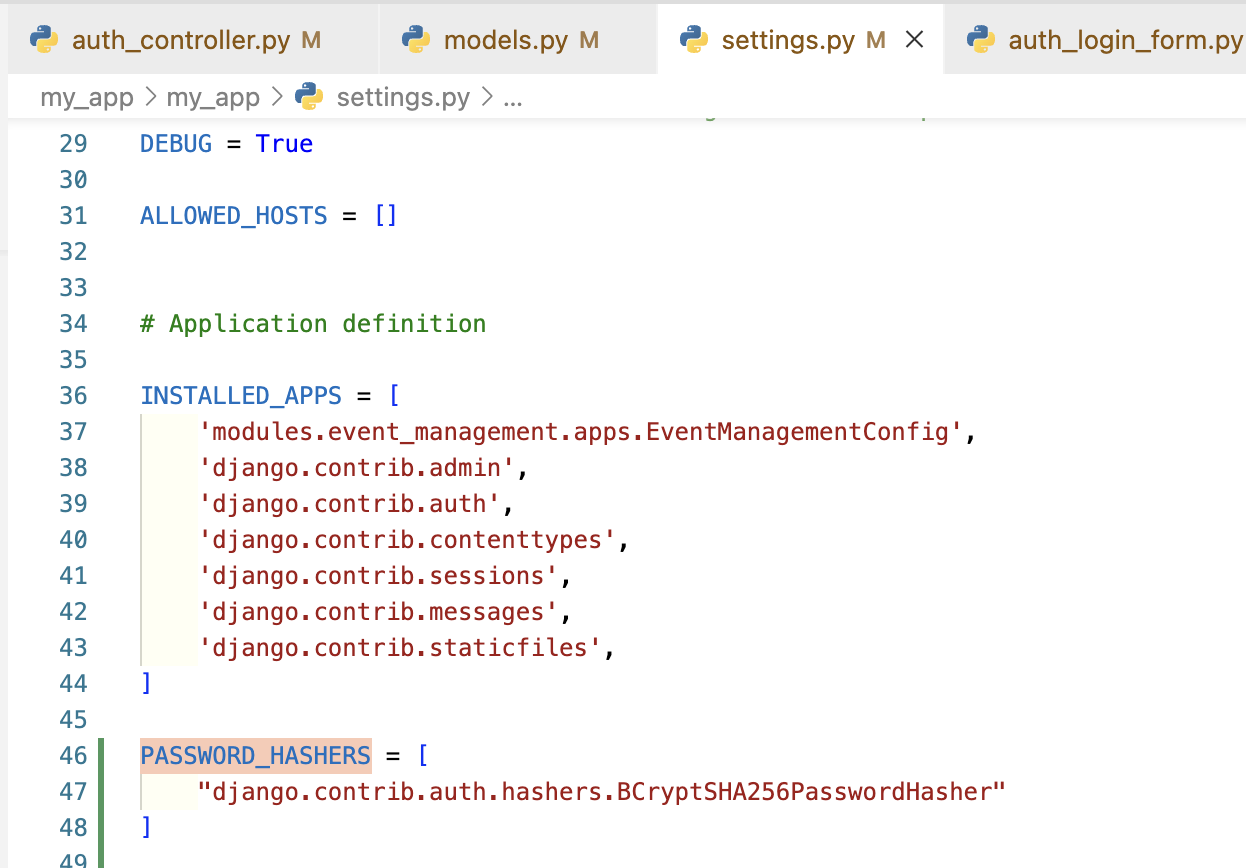


Figure 14: Thêm bcrypt Hasher vào settings.py

Mở **event\_management/models.py** và thêm hàm save(), hàm make\_password() nhận algorithm hasher là **bcrypt-sha256** để mã hóa mật khẩu

|  |
| --- |
| from django.db import models  import bcrypt  class AuthUserApi(models.Model):  id = models.AutoField(primary\_key=True)  email = models.CharField(max\_length=150, null=False, unique=True)  password = models.CharField(max\_length=100, null=False)    class Meta:  db\_table = 'api\_users'    # Calling save() directly on model to hash password  def save(self, \*args, \*\*kwargs):  if self.password:  self.password = make\_password(password=self.password)  super().save(args, kwargs) |

Luồng validate được xử lí như sau

Kết quả thu được với dữ liệu POSTMAN sử dụng dữ lieu như sau

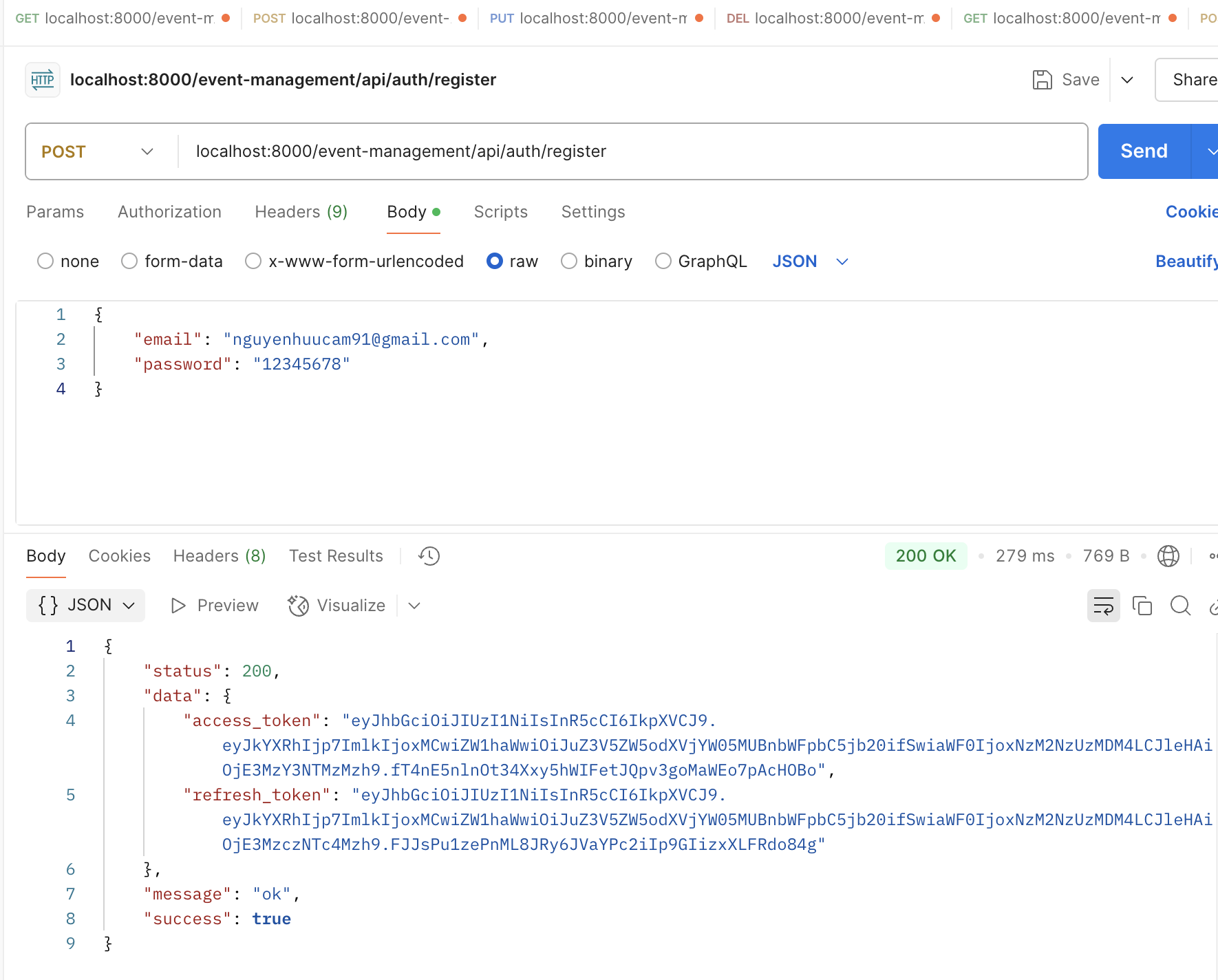


Figure 15: Data gửi lên

Kết quả khi lưu vào csdl

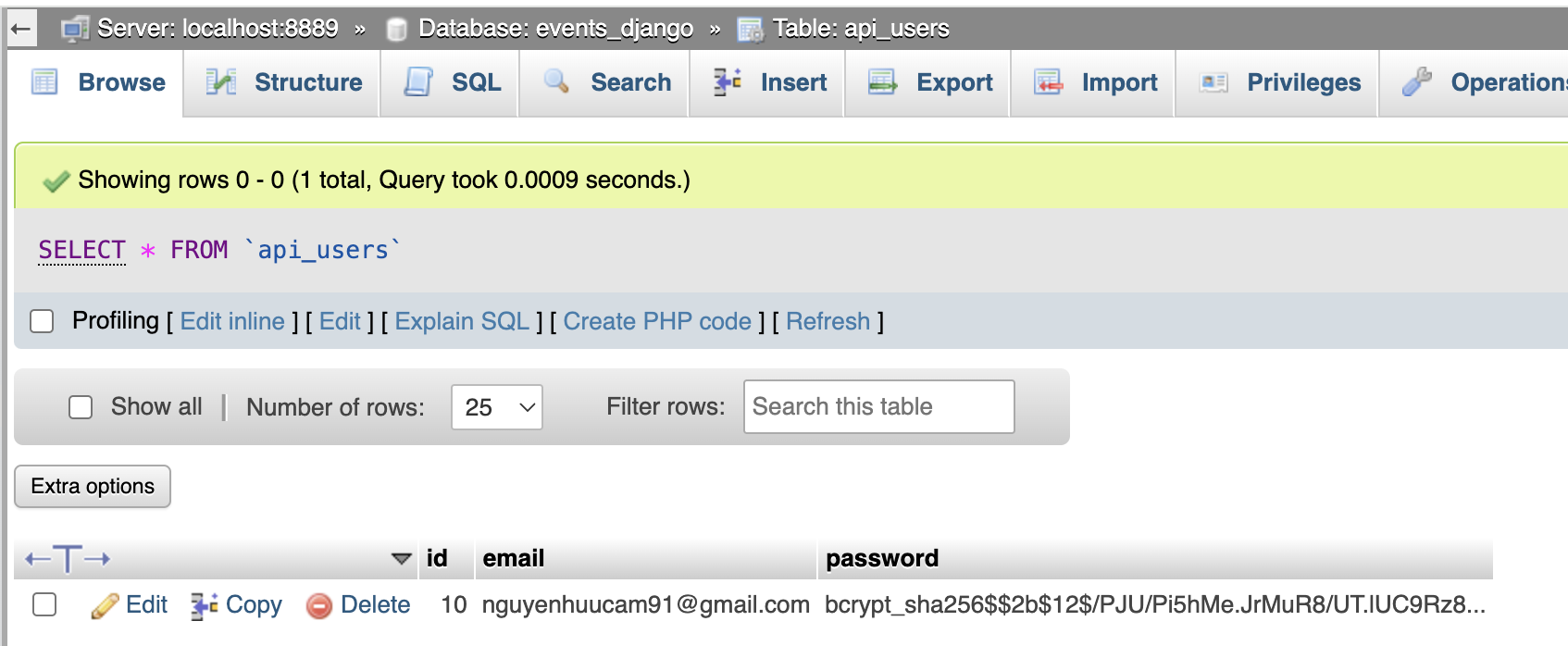


Figure 16: Mật khẩu được mã hóa bằng Bcrypt sha256

Đăng nhập chỉ cần kiểm tra mật khẩu đã nhập sau khi mã hóa và so sánh với mật khẩu trên CSDL thì tạo token. Để thuận tiện thì truy vấn ở đây sử dụng Manager class, được thể hiện qua hàm AuthUserApi.objects.authenticate(). Hàm này cho phép tự viết 1 custom query dưới dạng 1 hàm mà không viết trực tiếp truy vấn vào controller. Tính năng này cũng tương tự như Laravel Eloquent Query Scope.

Code đăng nhập trong **auth\_controller.py** như sau

|  |
| --- |
| from django.http import JsonResponse  import jwt, json  from ..forms.auth\_form import AuthLoginForm, AuthRegisterForm  from my\_app import settings  import datetime  from ..models import AuthUserApi  def login(request):  form = AuthLoginForm(json.loads(request.body))  if form.is\_valid():  current\_time = datetime.datetime.now(datetime.timezone.utc)  # Get data after validation  email = form.cleaned\_data.get('email')  password = form.cleaned\_data.get('password')  # Check against db  user = AuthUserApi.objects.authenticate(email, password)  if not user:  return JsonResponse({  'success': False,  'message': 'Failed to authenticate'  }, status = 401)    jwt\_access\_token = jwt.encode(  payload={  'data': {  'id': user.id,  'email': user.email  },  'iat': current\_time,  'exp': current\_time + datetime.timedelta(minutes=5), # 5 minute  }, key=settings.JWT\_SECRET, algorithm="HS256")    jwt\_refresh\_token = jwt.encode(  payload={  'data': {  'id': user.id,  'email': user.email  },  'iat': current\_time,  'exp': current\_time + datetime.timedelta(days=7), # 5 minute  }, key=settings.JWT\_SECRET, algorithm="HS256")    return JsonResponse({  'status': 200,  'data': {  'access\_token': jwt\_access\_token,  'refresh\_token': jwt\_refresh\_token,  },  'message': 'ok',  'success': True  }, status=200)  return JsonResponse({  'data': {  'errors': form.errors  }  }) |

Code trong **AuthLoginForm.py** sử dụng forms.Form thay vì forms.ModelForm. Sự khác nhau của forms.Form và forms.ModelForm nằm tại cách xử lí validation. forms.Form sẽ lấy validation ngay trên Form, trong khi forms.ModelForm sẽ thực thi validation dựa trên Model rule được định nghĩa. Trong forms.Form các trường không có rule kiểm tra Unique. Do đó nếu muốn sử dụng cần định nghĩa phương thức tùy chỉnh clean\_<field>() để validate trường.

|  |
| --- |
| from django import forms  from ..models import AuthUserApi  class AuthLoginForm(forms.Form):  # Email, bắt buộc với số kí tự tối thiểu là 7  email = forms.EmailField(min\_length=7)  password = forms.CharField(min\_length=5)  class Meta:  # fillable  fields = ['email', 'password'] |

Model class được cập nhật có thêm sự bổ sung của Manager class, điều này giúp tập trung tất cả các vấn đề liên quan đến mã khóa mật khẩu ở một chỗ thay vì đặt rải rác phần mã hóa mật khẩu ở các trang khác nhau.

|  |
| --- |
| from django.db import models  import bcrypt  from django.contrib.auth.hashers import check\_password, make\_password  class AuthUserApiManager(models.Manager):  def authenticate(self, email, password):  try:  user = self.get(email=email) # Get the user by email  if check\_password(password, user.password): # Compare raw password with hashed  return user # Return the user if the password matches  else:  return None # Invalid password  except self.model.DoesNotExist:  return None # User not found  class AuthUserApi(models.Model):  id = models.AutoField(primary\_key=True)  # Sử dụng rule unique  email = models.EmailField(max\_length=150, null=False, unique=True)  password = models.CharField(max\_length=100, null=False)  # Sử dụng objects để truy cập vào phương thức trong Manager class  # ví dụ như AuthUserApi.objects.authenticate(), có  # có thể thay tên objects này bằng từ khác, sử dụng AuthUserApi.<custom>.authenticate()  objects = AuthUserApiManager()    class Meta:  db\_table = 'api\_users'    # Calling save() directly on model to hash password  def save(self, \*args, \*\*kwargs):  if self.password:  self.password = make\_password(password=self.password)  super().save(args, kwargs) |

### MIddleware