University of Nebraska - Computer Science Tutoring Portal

Release 0.0.1

Nolan Gregory

Dec 03, 2023

CSLC PORTAL BACKEND

CHAPTER

ONE

ABOUT

CHAPTER

TWO

API ENDPOINTS

2.1 Course

class api.endpoints.course.**APICourseList**(***kwargs*)

Bases: APIView

get(*request: Request*) → Response

get_querystring(*request: Request*) → QueryDict | Any

post(*request: Request*) → Response

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

sanitize(*querystring: str*) → str

serializer_class

alias of CourseSerializer

2.2 Issues

class api.endpoints.issue.**APIIssueDetail**(***kwargs*)

Bases: APIView

get(*request: Request*, *pk: str | Any = Ellipsis*) → Response

put(*request: Request*, *pk: str | Any = Ellipsis*) → Response

query_obj(*pk: str*) → QuerySet | Any

class api.endpoints.issue.**APIIssueView**(***kwargs*)

Bases: APIView

get(*request: Request*) → Response

post(*request: Request*) → Response

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

serializer_class

alias of IssueSerializer

2.3 Professor

class api.endpoints.professor.**APIProfessorDetail**(***kwargs*)

Bases: APIView

get(*request: Request*, *professor_pk: str*) → Response

put(*request: Request*, *professor_pk: str*) → Response

query_obj(*pk: str*) → QuerySet

Add more fish or shrimp to the tank.

Parameters

- **inhabitant** – The type of inhabitant, either shrimp of fish
- **quantity** – The number of fish or shrimp to be added

Raises

TankIsFullError – if the tank is already full

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

serializer_class

alias of ProfessorSerializer

class api.endpoints.professor.**APIProfessorView**(***kwargs*)

Bases: APIView

get(*request: Request*) → Response

get_querystring(*request: Request*) → QueryDict | Any

post(*request: Request*, *search: str | None = None*) → Response

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

sanitize(*querystring: str*) → str

serializer_class

alias of ProfessorSerializer

2.4 Section

class api.endpoints.section.**APISectionDetail**(***kwargs*)

Bases: APIView

get(*request: Request*, *section_id: str*) → Response

put(*request: Request*, *section_id: str*) → Response

query_obj(*pk: str*) → QuerySet | None

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

serializer_class

alias of SectionSerializer

class api.endpoints.section.**APISectionView**(***kwargs*)

Bases: APIView

get(*request: Request*) → Response

get_querystring(*request: Request*) → QueryDict | Any

post(*request: Request*) → Response

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

sanitize(*querystring: str*) → str

serializer_class

alias of SectionSerializer

2.5 Ticket

class api.endpoints.ticket.**APITicketDetail**(***kwargs*)

Bases: APIView

get(*request: Request*, *ticket_id: int*) → Response

patch(*request: Request*, *ticket_id: int*) → Response

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

serializer_class

alias of TicketGetSerializer

class api.endpoints.ticket.**APITicketView**(***kwargs*)

Bases: APIView

get(*request: Request*) → Response

get_querystring(*request: Request*) → Any

post(*request: Request*) → Response

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

sanitize(*querystring: str*) → str

serializer_class

alias of TicketSerializer

2.6 User

class api.endpoints.user.**APIUserDetail**(***kwargs*)

Bases: APIView

get(*request: Request*, *user_id: str*) → Response

put(*request: Request*, *user_id: str*) → Response

query_obj(*pk: str*) → QueryDict | Any

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

serializer_class

alias of UserSerializer

class api.endpoints.user.**APIUserView**(***kwargs*)

Bases: APIView

get(*request: Request*) → Response

get_querystring(*request: Request*) → QueryDict | Any

post(*request: Request*) → Response

renderer_classes = (<class 'rest_framework.renderers.BrowsableAPIRenderer'>, <class 'rest_framework.renderers.JSONRenderer'>, <class 'rest_framework.renderers.HTMLFormRenderer'>)

sanitize(*querystring: str*) → str

serializer_class

alias of UserSerializer

CHAPTER

THREE

DATABASE MODELS

3.1 Course

class api.models.course.**Course**(**args*, ***kwargs*)

Bases: Model

The course table holds all information about a specific course.

Important: A course can have many sections, while a section can only be attributed to a single course. The course represents what you would see in a catalog (e.g., CIST-1400 & CSCI-1620), and sections are specific instances of the course (online, in person, etc.). Unlike sections, a course does not have a directly assigned professor; this data is stored in sections.

---

course_department

The department code of the course (e.g., "CSCI").

Type
CharField

course_name

The name of the course as it appears

Type
CharField

in the catalog

Type
e.g., "Operating Systems"

course_id

The unique identifier for the course.

Type
IntegerField

course_code

The code associated with the course (e.g., "4500").

Type
CharField

Manager:
generic (Manager): The default manager for the *Course* model.

Example

Examples can be given using either the `Example` or `Examples` sections. Sections support any reStructuredText formatting, including literal blocks:

```
$ python example_google.py
```

Todo:

- For module TODOs
- You have to also use `sphinx.ext.todo` extension

---

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

course_code

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

course_department

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

course_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

course_name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

generic: Manager = <django.db.models.manager.Manager object>

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

section_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ticket_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

usertocoursetaken

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

usertocoursetutored

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

3.2 Issues

class api.models.issue.**IssueManager**(**args*, ***kwargs*)

Bases: `Manager`

get_issues() → QuerySet

class api.models.issue.**Issues**(**args*, ***kwargs*)

Bases: `Model`

Issues are a field in the ticket that a user will choose to describe the issue they are having. This allows for ticket issues to be comprehensively studied, and provides cleaner data as opposed to users entering their issue type manually. With this style of ticket issue, data analysis can be done more rapidly and with a higher confidence than previous methods allowed. The fields in the table are the problem type (i.e. homework), the severity (used for data collection purposes). The admin is able to quickly add and modify the issues with no consequence to the attributed tickets.

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

generic: Manager = <django.db.models.manager.Manager object>

get_severity_display(*, *field=<django.db.models.fields.CharField: severity>*)

issue_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

problem_type

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

severity

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

ticket_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

3.3 Professor

class api.models.professor.**Professor**(**args*, ***kwargs*)

Bases: `Model`

The professor table holds all details attributed to a professor. This table contains a professors first name, their last name, their full name, their email address, whether they are currently active (see "semester" for more detail on that), and their prof. ID. Since the professor ID is guranteed to be unique, we have opted to use this as the primary key for the table. As such, professors are searchable by their unique ID, and the serializer will likewise return the str representation as well.

exception DoesNotExist

Bases: `ObjectDoesNotExist`

exception MultipleObjectsReturned

Bases: `MultipleObjectsReturned`

email

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

first_name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

full_name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

generic = <django.db.models.manager.Manager object>

is_active

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

last_name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

professor = <api.models.professor.ProfessorManager object>

professor_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

section_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

ticket_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

class api.models.professor.**ProfessorManager**(**args*, ***kwargs*)

Bases: `Manager`

get_professor(*professor: str*) → QuerySet

get_professors() → QuerySet

3.4 Section

3.5 Ticket

class api.models.ticket.**Ticket**(**args*, ***kwargs*)

Bases: Model

The base class for tickets in the database. This is where the meat of the application purpose lies, as this table will hold all fields associated with a specific ticket. These fields are the professor, the section, the semester, the issue, the student who submitted the ticket, the tutor who primarily helped with the ticket, the tutor(s) who assisted the primary tutor, the name of the student, whether it was a successful ticket, the time the ticket was created, the date the ticket was created, the time the ticket was claimed (different than created), the time the ticket was closed, additional tutor comments, and if the ticket was reopened after it was closed.

CLOSED = 'CLOSED'

exception DoesNotExist

Bases: ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

NEW = 'NEW'

OPENED = 'OPENED'

STATUS_CHOICES = [('NEW', 'New'), ('OPENED', 'Opened'), ('CLOSED', 'Closed')]

closed_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

course

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Child.parent is a ForwardManyToOneDescriptor instance.

course_id

created_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

description

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

generic = <django.db.models.manager.Manager object>

get_next_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=True*, ***kwargs*)

get_next_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=True*, ***kwargs*)

get_previous_by_created_at(*, *field=<django.db.models.fields.DateTimeField: created_at>*, *is_next=False*, ***kwargs*)

get_previous_by_updated_at(*, *field=<django.db.models.fields.DateTimeField: updated_at>*, *is_next=False*, ***kwargs*)

get_status_display(*, *field=<django.db.models.fields.CharField: status>*)

id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

issue

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

issue_id

messages_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

opened_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

professor

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

professor_id

status

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

student

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

student_id

ticket = <api.models.ticket.TicketManager object>

title

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

tutor

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Child.parent` is a `ForwardManyToOneDescriptor` instance.

tutor_id

updated_at

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

was_reopened

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

was_successful

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

class api.models.ticket.**TicketManager**(**args*, ***kwargs*)

Bases: `Manager`

get_active() → QuerySet

get_all() → QuerySet

get_completed() → QuerySet

get_course(*course: str*) → QuerySet

get_professor(*professor: str*) → QuerySet

get_section(*section: str*) → QuerySet

get_student(*student: str*) → QuerySet

get_successful() → QuerySet

get_tutor(*tutor: str*) → QuerySet

get_unclaimed() → QuerySet

3.6 User

class api.models.user.**AdminManager**(**args*, ***kwargs*)

Bases: Manager

get_admins() → QuerySet

class api.models.user.**StudentManager**(**args*, ***kwargs*)

Bases: Manager

get_student(*name: str*) → QuerySet

get_students() → QuerySet

class api.models.user.**TutorManager**(**args*, ***kwargs*)

Bases: Manager

get_tutor(*name: str*) → QuerySet

get_tutors() → QuerySet

class api.models.user.**User**(**args*, ***kwargs*)

Bases: Model

Generic user model for all active users of the application. Flags are used to indicate whether a user is an admin, a tutor, or a regular student. This currently will hold all data, such as tutor and admin specific data, but this might lead to empty fields within many of the objects in the table. This will be updated in the future to allow specific roles to be relations in the database, thus preventing empty fields (i.e. empty cells).

exception DoesNotExist

Bases: ObjectDoesNotExist

MSOID

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

exception MultipleObjectsReturned

Bases: MultipleObjectsReturned

admin = <api.models.user.AdminManager object>

courses_taken

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

courses_tutoring

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

email

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

generic = <django.db.models.manager.Manager object>

hour_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager class built by create_forward_many_to_many_manager() defined below.

is_active

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

is_admin

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

is_tutor

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

is_working

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

messages_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

name

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

student = <api.models.user.StudentManager object>

student_nuid

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

student_ticket

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

tutor = <api.models.user.TutorManager object>

tutor_ticket

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

user_bio

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

workinghours_set

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

CHAPTER

FOUR

API SCRIPTS

CHAPTER

FIVE

API CONFIG

5.1 Settings

5.2 base.asgi module

ASGI config for University-Nebraska-Tutor-Portal project.

It exposes the ASGI callable as a module-level variable named `application`.

For more information on this file, see https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/

5.3 base.settings module

Settings for University-Nebraska-Tutor-Portal project.

5.4 base.urls module

URL configuration for University-Nebraska-Tutor-Portal project.

The *urlpatterns* list routes URLs to views. For more information please see:
https://docs.djangoproject.com/en/4.2/topics/http/urls/

Examples: Function views

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path(‘’, views.home, name=’home’)

Class-based views

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path(‘’, Home.as_view(), name=’home’)

Including another URLconf

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path(‘blog/’, include(‘blog.urls’))

5.5 base.wsgi module

WSGI config for University-Nebraska-Tutor-Portal project.

It exposes the WSGI callable as a module-level variable named `application`.

For more information on this file, see https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/

5.6 Module contents

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

b

INDEX

H

I

L

M

N

O

P

Q

R

S

T

U

W