

---

# **University of Nebraska - Computer Science Tutoring Portal**

***Release 0.0.1***

**Nolan Gregory**

**Dec 03, 2023**



# CSLC PORTAL BACKEND

<b>1</b>	<b>About</b>	<b>1</b>
<b>2</b>	<b>API Endpoints</b>	<b>3</b>
2.1	Course . . . . .	3
2.2	Issues . . . . .	3
2.3	Professor . . . . .	4
2.4	Section . . . . .	5
2.5	Ticket . . . . .	5
2.6	User . . . . .	6
<b>3</b>	<b>Database Models</b>	<b>7</b>
3.1	Course . . . . .	7
3.2	Issues . . . . .	9
3.3	Professor . . . . .	10
3.4	Section . . . . .	12
3.5	Ticket . . . . .	12
3.6	User . . . . .	15
<b>4</b>	<b>API Scripts</b>	<b>19</b>
<b>5</b>	<b>API Config</b>	<b>21</b>
5.1	Settings . . . . .	21
5.2	base.asgi module . . . . .	21
5.3	base.settings module . . . . .	21
5.4	base.urls module . . . . .	21
5.5	base.wsgi module . . . . .	22
5.6	Module contents . . . . .	22
<b>6</b>	<b>Indices and tables</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>
	<b>Index</b>	<b>27</b>



---

CHAPTER  
**ONE**

---

**ABOUT**



## 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 or 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
```

## 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 guaranteed 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.

**API SCRIPTS**





## 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

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### a

- `api.endpoints.course`, 3
- `api.endpoints.issue`, 3
- `api.endpoints.professor`, 4
- `api.endpoints.section`, 5
- `api.endpoints.ticket`, 5
- `api.endpoints.user`, 6
- `api.models.course`, 7
- `api.models.issue`, 9
- `api.models.professor`, 10
- `api.models.ticket`, 12
- `api.models.user`, 15

### b

- `base`, 22
- `base.asgi`, 21
- `base.settings`, 21
- `base.urls`, 21
- `base.wsgi`, 22



## A

admin (*api.models.user.User* attribute), 16  
 AdminManager (*class in api.models.user*), 15  
 api.endpoints.course  
     module, 3  
 api.endpoints.issue  
     module, 3  
 api.endpoints.professor  
     module, 4  
 api.endpoints.section  
     module, 5  
 api.endpoints.ticket  
     module, 5  
 api.endpoints.user  
     module, 6  
 api.models.course  
     module, 7  
 api.models.issue  
     module, 9  
 api.models.professor  
     module, 10  
 api.models.ticket  
     module, 12  
 api.models.user  
     module, 15  
 APICourseList (*class in api.endpoints.course*), 3  
 APIIssueDetail (*class in api.endpoints.issue*), 3  
 APIIssueView (*class in api.endpoints.issue*), 3  
 APIProfessorDetail (*class in api.endpoints.professor*), 4  
 APIProfessorView (*class in api.endpoints.professor*), 4  
 APISectionDetail (*class in api.endpoints.section*), 5  
 APISectionView (*class in api.endpoints.section*), 5  
 APITicketDetail (*class in api.endpoints.ticket*), 5  
 APITicketView (*class in api.endpoints.ticket*), 5  
 APIUserDetail (*class in api.endpoints.user*), 6  
 APIUIView (*class in api.endpoints.user*), 6

## B

base  
     module, 22  
 base.asgi

    module, 21  
 base.settings  
     module, 21  
 base.urls  
     module, 21  
 base.wsgi  
     module, 22

## C

CLOSED (*api.models.ticket.Ticket* attribute), 12  
 closed\_at (*api.models.ticket.Ticket* attribute), 12  
 course (*api.models.ticket.Ticket* attribute), 12  
 Course (*class in api.models.course*), 7  
 Course.DoesNotExist, 8  
 Course.MultipleObjectsReturned, 8  
 course\_code (*api.models.course.Course* attribute), 7, 8  
 course\_department (*api.models.course.Course* attribute), 7, 8  
 course\_id (*api.models.course.Course* attribute), 7, 8  
 course\_id (*api.models.ticket.Ticket* attribute), 12  
 course\_name (*api.models.course.Course* attribute), 7, 8  
 courses\_taken (*api.models.user.User* attribute), 16  
 courses\_tutoring (*api.models.user.User* attribute), 16  
 created\_at (*api.models.ticket.Ticket* attribute), 12

## D

description (*api.models.ticket.Ticket* attribute), 12

## E

email (*api.models.professor.Professor* attribute), 10  
 email (*api.models.user.User* attribute), 16

## F

first\_name (*api.models.professor.Professor* attribute), 10  
 full\_name (*api.models.professor.Professor* attribute), 10

## G

generic (*api.models.course.Course* attribute), 8  
 generic (*api.models.issue.Issues* attribute), 9  
 generic (*api.models.professor.Professor* attribute), 11

generic (*api.models.ticket.Ticket* attribute), 12  
 generic (*api.models.user.User* attribute), 16  
 get() (*api.endpoints.course.APICourseList* method), 3  
 get() (*api.endpoints.issue.APIIssueDetail* method), 3  
 get() (*api.endpoints.issue.APIIssueView* method), 3  
 get() (*api.endpoints.professor.APIProfessorDetail* method), 4  
 get() (*api.endpoints.professor.APIProfessorView* method), 4  
 get() (*api.endpoints.section.APISectionDetail* method), 5  
 get() (*api.endpoints.section.APISectionView* method), 5  
 get() (*api.endpoints.ticket.APITicketDetail* method), 5  
 get() (*api.endpoints.ticket.APITicketView* method), 5  
 get() (*api.endpoints.user.APIUserDetail* method), 6  
 get() (*api.endpoints.user.APIUserView* method), 6  
 get\_active() (*api.models.ticket.TicketManager* method), 14  
 get\_admins() (*api.models.user.AdminManager* method), 15  
 get\_all() (*api.models.ticket.TicketManager* method), 15  
 get\_completed() (*api.models.ticket.TicketManager* method), 15  
 get\_course() (*api.models.ticket.TicketManager* method), 15  
 get\_issues() (*api.models.issue.IssueManager* method), 9  
 get\_next\_by\_created\_at() (*api.models.ticket.Ticket* method), 12  
 get\_next\_by\_updated\_at() (*api.models.ticket.Ticket* method), 13  
 get\_previous\_by\_created\_at() (*api.models.ticket.Ticket* method), 13  
 get\_previous\_by\_updated\_at() (*api.models.ticket.Ticket* method), 13  
 get\_professor() (*api.models.professor.ProfessorManager* method), 11  
 get\_professor() (*api.models.ticket.TicketManager* method), 15  
 get\_professors() (*api.models.professor.ProfessorManager* method), 11  
 get\_querystring() (*api.endpoints.course.APICourseList* method), 3  
 get\_querystring() (*api.endpoints.professor.APIProfessorView* method), 4  
 get\_querystring() (*api.endpoints.section.APISectionView* method), 5  
 get\_querystring() (*api.endpoints.ticket.APITicketView* method), 6  
 get\_querystring() (*api.endpoints.user.APIUserView* method), 6  
 get\_section() (*api.models.ticket.TicketManager* method), 15  
 get\_severity\_display() (*api.models.issue.Issues* method), 10  
 get\_status\_display() (*api.models.ticket.Ticket* method), 13  
 get\_student() (*api.models.ticket.TicketManager* method), 15  
 get\_student() (*api.models.user.StudentManager* method), 15  
 get\_students() (*api.models.user.StudentManager* method), 15  
 get\_successful() (*api.models.ticket.TicketManager* method), 15  
 get\_tutor() (*api.models.ticket.TicketManager* method), 15  
 get\_tutor() (*api.models.user.TutorManager* method), 15  
 get\_tutors() (*api.models.user.TutorManager* method), 15  
 get\_unclaimed() (*api.models.ticket.TicketManager* method), 15  
**H**  
 hour\_set (*api.models.user.User* attribute), 16  
**I**  
 id (*api.models.course.Course* attribute), 8  
 id (*api.models.ticket.Ticket* attribute), 13  
 is\_active (*api.models.professor.Professor* attribute), 11  
 is\_active (*api.models.user.User* attribute), 16  
 is\_admin (*api.models.user.User* attribute), 16  
 is\_tutor (*api.models.user.User* attribute), 16  
 is\_working (*api.models.user.User* attribute), 17  
 issue (*api.models.ticket.Ticket* attribute), 13  
 issue\_id (*api.models.issue.Issues* attribute), 10  
 issue\_id (*api.models.ticket.Ticket* attribute), 13  
 IssueManager (class in *api.models.issue*), 9  
 Issues (class in *api.models.issue*), 9  
 Issues.DoesNotExist, 9  
 Issues.MultipleObjectsReturned, 9  
 last\_name (*api.models.professor.Professor* attribute), 11  
**M**  
 messages\_set (*api.models.ticket.Ticket* attribute), 13  
 messages\_set (*api.models.user.User* attribute), 17  
 module  
   api.endpoints.course, 3  
   api.endpoints.issue, 3  
   api.endpoints.professor, 4  
   api.endpoints.section, 5  
   api.endpoints.ticket, 5  
   api.endpoints.user, 6



[api.models.course](#), 7  
[api.models.issue](#), 9  
[api.models.professor](#), 10  
[api.models.ticket](#), 12  
[api.models.user](#), 15  
[base](#), 22  
[base.asgi](#), 21  
[base.settings](#), 21  
[base.urls](#), 21  
[base.wsgi](#), 22

[MSOID](#) ([api.models.user.User](#) attribute), 15

## N

[name](#) ([api.models.ticket.Ticket](#) attribute), 13  
[name](#) ([api.models.user.User](#) attribute), 17  
[NEW](#) ([api.models.ticket.Ticket](#) attribute), 12

## O

[OPENED](#) ([api.models.ticket.Ticket](#) attribute), 12  
[opened\\_at](#) ([api.models.ticket.Ticket](#) attribute), 13

## P

[patch\(\)](#) ([api.endpoints.ticket.APITicketDetail](#) method), 5  
[post\(\)](#) ([api.endpoints.course.APICourseList](#) method), 3  
[post\(\)](#) ([api.endpoints.issue.APIIssueView](#) method), 3  
[post\(\)](#) ([api.endpoints.professor.APIProfessorView](#) method), 4  
[post\(\)](#) ([api.endpoints.section.APISectionView](#) method), 5  
[post\(\)](#) ([api.endpoints.ticket.APITicketView](#) method), 6  
[post\(\)](#) ([api.endpoints.user.APIUserView](#) method), 6  
[problem\\_type](#) ([api.models.issue.Issues](#) attribute), 10  
[professor](#) ([api.models.professor.Professor](#) attribute), 11  
[professor](#) ([api.models.ticket.Ticket](#) attribute), 13  
[Professor](#) (class in [api.models.professor](#)), 10  
[Professor.DoesNotExist](#), 10  
[Professor.MultipleObjectsReturned](#), 10  
[professor\\_id](#) ([api.models.professor.Professor](#) attribute), 11  
[professor\\_id](#) ([api.models.ticket.Ticket](#) attribute), 14  
[ProfessorManager](#) (class in [api.models.professor](#)), 11  
[put\(\)](#) ([api.endpoints.issue.APIIssueDetail](#) method), 3  
[put\(\)](#) ([api.endpoints.professor.APIProfessorDetail](#) method), 4  
[put\(\)](#) ([api.endpoints.section.APISectionDetail](#) method), 5  
[put\(\)](#) ([api.endpoints.user.APIUserDetail](#) method), 6

## Q

[query\\_obj\(\)](#) ([api.endpoints.issue.APIIssueDetail](#) method), 3  
[query\\_obj\(\)](#) ([api.endpoints.professor.APIProfessorDetail](#) method), 4

[query\\_obj\(\)](#) ([api.endpoints.section.APISectionDetail](#) method), 5  
[query\\_obj\(\)](#) ([api.endpoints.user.APIUserDetail](#) method), 6

## R

[renderer\\_classes](#) ([api.endpoints.course.APICourseList](#) attribute), 3  
[renderer\\_classes](#) ([api.endpoints.issue.APIIssueView](#) attribute), 3  
[renderer\\_classes](#) ([api.endpoints.professor.APIProfessorDetail](#) attribute), 4  
[renderer\\_classes](#) ([api.endpoints.professor.APIProfessorView](#) attribute), 4  
[renderer\\_classes](#) ([api.endpoints.section.APISectionDetail](#) attribute), 5  
[renderer\\_classes](#) ([api.endpoints.section.APISectionView](#) attribute), 5  
[renderer\\_classes](#) ([api.endpoints.ticket.APITicketDetail](#) attribute), 5  
[renderer\\_classes](#) ([api.endpoints.ticket.APITicketView](#) attribute), 6  
[renderer\\_classes](#) ([api.endpoints.user.APIUserDetail](#) attribute), 6  
[renderer\\_classes](#) ([api.endpoints.user.APIUserView](#) attribute), 6

## S

[sanitize\(\)](#) ([api.endpoints.course.APICourseList](#) method), 3  
[sanitize\(\)](#) ([api.endpoints.professor.APIProfessorView](#) method), 4  
[sanitize\(\)](#) ([api.endpoints.section.APISectionView](#) method), 5  
[sanitize\(\)](#) ([api.endpoints.ticket.APITicketView](#) method), 6  
[sanitize\(\)](#) ([api.endpoints.user.APIUserView](#) method), 6  
[section\\_set](#) ([api.models.course.Course](#) attribute), 8  
[section\\_set](#) ([api.models.professor.Professor](#) attribute), 11  
[serializer\\_class](#) ([api.endpoints.course.APICourseList](#) attribute), 3  
[serializer\\_class](#) ([api.endpoints.issue.APIIssueView](#) attribute), 3  
[serializer\\_class](#) ([api.endpoints.professor.APIProfessorDetail](#) attribute), 4  
[serializer\\_class](#) ([api.endpoints.professor.APIProfessorView](#) attribute), 4  
[serializer\\_class](#) ([api.endpoints.section.APISectionDetail](#) attribute), 5  
[serializer\\_class](#) ([api.endpoints.section.APISectionView](#) attribute), 5

`serializer_class` (*api.endpoints.ticket.APITicketDetail*  
    *attribute*), 5  
`serializer_class` (*api.endpoints.ticket.APITicketView*  
    *attribute*), 6  
`serializer_class` (*api.endpoints.user.APIUserDetail*  
    *attribute*), 6  
`serializer_class` (*api.endpoints.user.APIUserView*  
    *attribute*), 6  
`severity` (*api.models.issue.Issues attribute*), 10  
`status` (*api.models.ticket.Ticket attribute*), 14  
`STATUS_CHOICES` (*api.models.ticket.Ticket attribute*), 12  
`student` (*api.models.ticket.Ticket attribute*), 14  
`student` (*api.models.user.User attribute*), 17  
`student_id` (*api.models.ticket.Ticket attribute*), 14  
`student_nuid` (*api.models.user.User attribute*), 17  
`student_ticket` (*api.models.user.User attribute*), 17  
`StudentManager` (*class in api.models.user*), 15

## T

`ticket` (*api.models.ticket.Ticket attribute*), 14  
`Ticket` (*class in api.models.ticket*), 12  
`Ticket.DoesNotExist`, 12  
`Ticket.MultipleObjectsReturned`, 12  
`ticket_set` (*api.models.course.Course attribute*), 8  
`ticket_set` (*api.models.issue.Issues attribute*), 10  
`ticket_set` (*api.models.professor.Professor attribute*),  
    11  
`TicketManager` (*class in api.models.ticket*), 14  
`title` (*api.models.ticket.Ticket attribute*), 14  
`tutor` (*api.models.ticket.Ticket attribute*), 14  
`tutor` (*api.models.user.User attribute*), 17  
`tutor_id` (*api.models.ticket.Ticket attribute*), 14  
`tutor_ticket` (*api.models.user.User attribute*), 17  
`TutorManager` (*class in api.models.user*), 15

## U

`updated_at` (*api.models.ticket.Ticket attribute*), 14  
`User` (*class in api.models.user*), 15  
`User.DoesNotExist`, 15  
`User.MultipleObjectsReturned`, 15  
`user_bio` (*api.models.user.User attribute*), 17  
`usertocoursetaken` (*api.models.course.Course at-*  
    *tribute*), 9  
`usertocoursetutored` (*api.models.course.Course at-*  
    *tribute*), 9

## W

`was_reopened` (*api.models.ticket.Ticket attribute*), 14  
`was_successful` (*api.models.ticket.Ticket attribute*), 14  
`workinghours_set` (*api.models.user.User attribute*), 18