ICode

Versión 1.0**

Andres Guevara Mastretta - Miguel Angel Herrera Islas

06 de junio de 2023

Contenido:

En esta archivo de planea documentar todos los aspectos técnicos de la plataforma ICode Se organizara por elementos dentro de cada una de nuestras aplicaciones de Django, permitiendo mostrar los modulos de cada una de ellas de manera general, para despues de manera particular, poder observar dentro de ellas las funciones, su código, y como es que funcionan y se relacionan dentro de nuestro desarrollo.

CAPÍTULO 1

Vistas App Miembros

members.views.**is_student**(*email_address*)

La funcion is_student dentro de nuestro modulo de vistas se encarga de realizar la comprobacion con funciones RegEx de si un correo enviado en una solicitud de registro cumple con las condiciones para ser considerado email de alumno, comenzar con la letra A y seguir de 8 numeros.

param [email_address]
[String que contiene el email del usuario]

members.views.**login_user**(*request*)

La funcion login_user dentro de nuestro modulo de vistas tiene la funcionalidad de realizar el proceso de inicio de sesion para los usuarios. Primero requiere conocer que tipo de metodo el usuario esta enviando desde la request, si es POST se procede a obtener tanto su email como su correo, que son los elementos que lo identifican en nuestra base de datos, se autentica al usuario con el metodo de autenticacion de Django, y si no existen problemas con su solicitud, se inicia sesion de manera ordinaria y se redirecciona a la pagina principal.

En caso de exista algun error, se envía un mensaje para que lo intente nuevamente. En caso de que sea la primera vez que el usuario entra a la pagina de login, simplemente se procese a mandar a renderizar el archivo html para que lo llene.

Parámetros
[request] – [Request es el nombre comun en Django para mencionar a un HttpRequest que es enviado con todos los metadatos de esta solicitud, se obtiene y se procesa]

members.views.**logout_user**(*request*)

La funcion de logout_user dentro de nuestro modulo de vistas se encarga de realizar el cierre de sesion de un usuario. Se utilizan los modulos por defecto de Django para poder realizar esto. Se obtiene desde la request y con el modulo logout un cierre de sesion del usuario, se da un mensaje de exito y se redirecciona a la pagina principal.

Parámetros
[request] – [Request es el nombre comun en Django para mencionar a un HttpRequest que es enviado con todos los metadatos de esta solicitud, se obtiene y se procesa]

members.views.**register_user**(*request*)

La funcion de register_user dentro de nuestro modulo de vistas se encarga de realizar el registro de nuevos usuarios a la plataforma. Primero se comprueba el metodo con el que se envia la solicitud, en caso de set POST se obtienen los elementos identificadores del usuario, su email y su contraseña. Despues se utilizan las funciones por defecto de Django de autenticacion para despues registrar al usuario.

Es en este punto donde se realiza el analisis de los email de los usuarios nuevos, y se determina su rol en la pagina, en caso de ser alumnos, usando la funcion «is_student» podemos obtener el rol del usuario, y modificarlo para que en nuestra base de datos se vea reflejado, ya sea de un estudiante o de un profesor.

Parámetros

[request] – [Request es el nombre comun en Django para mencionar a un HttpRequest que es enviado con todos los metadatos de esta solicitud, se obtiene y se procesa]

members.views.**validate_email_address**(*email_address*)

La funcion de validate_email_address dentro de nuestro modulo de vistas se encarga de realizar la comprobacion con funciones RegEx de si un correo enviado en una solicitud de registro cumple con las condiciones para ser considerado un email valido.

param [email_address]

[String que contiene el email del usuario]

CAPÍTULO 2

Vistas App Core

class core.views.**GraphView**(***kwargs*)

Vista utilizada para poder desplegar en el template main_path.html la ruta general de aprendizaje mostrada en su relacion de nodos. Aquí se tiene en cuenta los objetos del modelo SubjectRelation para poder, haciendo uso de la libreria Digraph, crear los nodos del grafo, darles un estilo y darles funcionalidad tambien. Cada uno al interactuar con el te envia a su propia pagina donde se deberan resolver sus ejercicios correspondientes.

get_context_data(***kwargs: Any*)

template_name = 'core/main_path.html'

test_func()

class core.views.**HomeView**(***kwargs*)

Esta es la vista que se encarga de controlar el uso de código por parte del usuario. Estamos utilizando el servicio de Judge0, montando en una maquina virtual con el cual nosotros nos encargamos de poder tener llamadas a la API de este servicio con el código del usuario, obtener un Token único, y poder posteriormente desplegar el resultado de ese código en pantalla.

Decidimos utilizar Judge0 por las facilidades que da en cuestiones de seguridad que otras funciones como Exec, asi como la libertad que da el poder tenerlo montado en un servidor propio.

Ademas, se encarga de procesar los test-cases de los ejercicios de codigo de la ruta general, se encarga de comprobar input y output y otorgar el resultado correspondiente al codigo del usuario. Esto es util ya que nosotros controlamos los inputs y outputs del codigo.

form_class

alias de *CodeInputForm*

post(*request*)

Handle POST requests: instantiate a form instance with the passed POST variables and then check if it's valid.

success_url

template_name = 'core/home.html'

class core.views.**SubjectDetailView**(***kwargs*)

Muestra los ejercicios correspondientes al tema seleccionado por el alumno, de igual manera determina el nivel actual del usuario y se encarga de filtrar los ejercicios en base a ese nivel y que al mismo tiempo sean ejercicios nuevos para el usuario

code = ''

form_class

alias de *CodingExerciseInputForm*

get_context_data(***kwargs: Any*)

Insert the single object into the context dict.

levelUp(*request*)

model

alias de *Subject*

post(*request*, **args*, ***kwargs*)

template_name = 'core/subject_detail.html'

core.views.**actualizar_skips**(*request*)

Funcion que se encarga de recibir las llamadas desde el template de los ejercicios de la Ruta General y que administra el numero de skips que tiene el usuario actualmente y les añade uno hasta llegar a un maximo de 3 niveles por dificultad y por tema.

CAPÍTULO 3

Vistas Profesor

class professor.views.**CodingExerciseCreateView**(***kwargs*)

Permite a un profesor crear un nuevo ejercicio de codificación desde una interfaz gráfica o cargándolo desde un JSON.

clean_test_cases()

form_class

alias de CodingExerciseForm

form_valid(*form*)

If the form is valid, save the associated model.

model

alias de *CodingExercise*

success_url = '/professor/new_coding/'

test_func()

class professor.views.**CodingExerciseUpdateView**(***kwargs*)

Permite a un profesor actualizar un nuevo ejercicio de codificación desde una interfaz gráfica o cargándolo desde un JSON.

form_class

alias de CodingExerciseForm

form_valid(*form*)

If the form is valid, save the associated model.

model

alias de *CodingExercise*

success_url = '/professor/my_exercises/'

test_func()

class professor.views.CourseCreateView(**kwargs)

Permite a un profesor crear un nuevo curso al que se pueden inscribir los estudiantes.

fields = ['name']

form_valid(form)

If the form is valid, save the associated model.

model

alias de Course

success_url = '/professor/courses/'

test_func()

class professor.views.CourseDeleteView(*args, **kwargs)

Permite a un profesor eliminar un curso que este haya creado.

model

alias de Course

success_url = '/professor/courses/'

test_func()

class professor.views.CourseListView(**kwargs)

Muestra a un profesor una lista de todos los cursos en los que este es administrador (los que hayan sido creados por este profesor).

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias de Course

test_func()

class professor.views.CourseStudentsView(**kwargs)

Muestra al profesor una lista de los estudiantes inscritos en este curso. También muestra los botones para eliminar el curso o eliminar a un estudiante del curso y un botón para copiar el id del curso y poder compartirlo con los estudiantes.

model

alias de Course

template_name = 'professor/course_students.html'

test_func()

class professor.views.EnrollmentDeleteView(*args, **kwargs)

Permite a un profesor eliminar a un estudiante de un curso que este haya creado.

get_success_url()

Return the URL to redirect to after processing a valid form.

model

alias de *Enrollment*

test_func()

class professor.views.**ExerciseListView**(***kwargs*)

Muestra los ejercicios creados por este profesor en una tabla que muestra el título del ejercicio, estado de aceptación y mensaje del administrador.

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias de *Exercise*

template_name = 'professor/exercise_list.html'

test_func()

class professor.views.**MultipleChoiceExerciseCreateView**(***kwargs*)

Permite a un profesor crear un nuevo ejercicio de opción múltiple desde una interfaz gráfica o cargándolo desde un JSON.

clean_options()

form_class

alias de MultipleChoiceExerciseForm

form_valid(*form*)

If the form is valid, save the associated model.

model

alias de *MultipleChoiceExercise*

success_url = '/professor/new_multiple_choice/'

test_func()

class professor.views.**MultipleChoiceExerciseUpdateView**(***kwargs*)

Permite a un profesor actualizar un nuevo ejercicio de opción múltiple desde una interfaz gráfica o cargándolo desde un JSON.

form_class

alias de MultipleChoiceExerciseForm

form_valid(*form*)

If the form is valid, save the associated model.

model

alias de *MultipleChoiceExercise*

success_url = '/professor/my_exercises/'

test_func()

CAPÍTULO 4

Vistas Estudiantes

class student.views.**CourseListView**(***kwargs*)

Esta vista sirve para mostrarle a un estudiante los cursos en los que está inscrito. Incluye un campo de entrada de texto mediante «CourseJoinForm()» que permite al estudiante inscribirse a un curso ingresando el id del curso

get_context_data(***kwargs*)

Get the context for this view.

get_queryset()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

model

alias de *Course*

template_name = 'student/course_list.html'

test_func()

student.views.**join_course**(*request*)

Vista utilizada para manejar la inscripción de un estudiante a un curso. Busca un curso con un id igual al ingresado por el estudiante. En caso de encontrarlo, agrega al estudiante a la lista de estudiantes inscritos en el curso. De lo contrario, redirige a la página anterior con un mensaje de error.

CAPÍTULO 5

Modelos App Members

class members.models.**CustomUser**(**args*, ***kwargs*)

La clase CustomUser es la que se encarga de crear nuestra tabla de usuario en la base de datos de Django, es donde nosotros modificamos al modelo de usuario por defecto de Django para poder añadirle los elementos que para nuestro desarrollo son importantes, como su email, su rol, etc. Este modelo es utilizado en otros modulos como modelo base para los usuarios, ya sea que inicien sesion o se registren por primera vez»

Parámetros

[AbstractUser] – [AbstractUser modelo de usuario por defecto de Django, este es modificado por nosotros añadiendole los elementos extras que son utiles para nuestro proyecto.]

exception DoesNotExist

exception MultipleObjectsReturned

PROFESSOR = 1

REQUIRED_FIELDS = []

ROLE_CHOICES = ((0, 'STUDENT'), (1, 'PROFESSOR'))

STUDENT = 0

USERNAME_FIELD = 'email'

completedlevelsperuser_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.

course_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.

courses

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.

date_joined

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

email

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

enrollment_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.

exercise_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.

exerciseinstanceingeneralpath_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.

first_name

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

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

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

get_role_display(*, *field=<django.db.models.fields.PositiveSmallIntegerField: role>*)

groups

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.

id

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

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_staff

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

is_superuser

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

last_login

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.

logentry_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.

objects = <members.manager.CustomUserManager object>

password

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

role

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

user_permissions

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.

username = None

CAPÍTULO 6

Modelos Ruta General Aprendizaje

class core.models.**CodingExercise**(**args*, ***kwargs*)

Representa un ejercicio de codificación en donde «test_cases» son los casos de prueba que tiene que pasar el código del estudiante para completar el ejercicio exitosamente.

exception DoesNotExist

exception MultipleObjectsReturned

exercise_ptr

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

exercise_ptr_id

test_cases

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

class core.models.**CompletedLevelsPerUser**(**args*, ***kwargs*)

Modelo que se encarga de controlar los niveles completados por el usuario y de subirlo de nivel al completar el requisito deseado

exception DoesNotExist

LEVEL = (('facil', 'Facil'), ('media', 'Intermedio'), ('dificil', 'Dificil'), ('Completado', 'Completado'))

exception MultipleObjectsReturned

get_level_display(*, *field=<django.db.models.fields.CharField: level>*)

id

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

level

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

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

skips

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

subject

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.

subject_id

user

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.

user_id

class core.models.**Exercise**(**args*, ***kwargs*)

Representa los campos comunes de un ejercicio que puede ser subido a la plataforma por un profesor y resuelto por estudiantes en la ruta general de aprendizaje o en assignments específicos de un curso.

ACCEPTED = 2

DIFFICULTY_CHOICES = (('facil', 'Fácil'), ('media', 'Media'), ('dificil', 'Difícil'))

exception DoesNotExist

IN_REVIEW = 1

exception MultipleObjectsReturned

REJECTED = 0

STATUS_CHOICES = ((0, 'REJECTED'), (1, 'IN REVIEW'), (2, 'ACCEPTED'))

TOPICS = (('1.1 Uso de programación para la solución de problemas', '1.1 Uso de programación para la solución de problemas'), ('1.2 lenguajes de programacion', '1.2 Lenguajes de programación'), ('1.3 fases de desarrollo de un programa', '1.3 Fases de desarrollo de un programa'), ('1.4 ambientes de programacion', '1.4 Ambientes de programación'), ('2.1 estructura basica de un programa secuencial.', '2.1 Estructura básica de un programa secuencial.'), ('2.2 variables, constantes y tipos de datos', '2.2 Variables, constantes y tipos de datos'), ('2.3 expresiones aritmeticas', '2.3 Expresiones aritméticas'), ('2.4 funciones predefinidas', '2.4 Funciones predefinidas'), ('2.5 solucion de problemas con formulas matematicas', '2.5 Solución de problemas con fórmulas matemáticas'), ('3.1 programacion modular', '3.1 Programación modular'), ('3.2 funciones', '3.2 Funciones'), ('3.3 solucion de problemas con funciones', '3.3 Solución de problemas con funciones'), ('4.1 expresiones logicas', '4.1 Expresiones lógicas'), ('4.2 estatus de decision', '4.2 Estatus de decisión'), ('4.3 estatutos de decision anidados', '4.3 Estatutos de decisión anidados'), ('4.4 solucion de problemas con estatutos de decision', '4.4 Solución de problemas con estatutos de decisión'), ('5.1 while', '5.1 While'), ('5.2 for', '5.2 For'), ('5.3 ciclos anidados', '5.3 Ciclos anidados'), ('5.4 solucion de problemas con estatutos de repeticion', '5.4 Solución de problemas con estatutos de repetición'), ('6.1 listas', '6.1 Listas'), ('6.2 recorridos de listas', '6.2 Recorridos de listas'), ('6.3 matrices', '6.3 Matrices'), ('6.4 strings', '6.4 Strings'), ('6.5 solucion de problemas con listas', '6.5 Solución de problemas con listas'), ('6.6 solucion de problemas con matrices', '6.6 Solución de problemas con matrices'), ('6.7 solucion de problemas con strings', '6.7 Solución de problemas con strings'), ('7.7 creacion y uso de archivos', '7.7 Creación y uso de archivos'), ('7.8 solucion de problemas con archivos', '7.8 Solución de problemas con archivos'))

author

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

codingexercise

Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

created_by

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.

created_by_id

description

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

difficulty

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

exerciseinstanceingeneralpath_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.

get_difficulty_display(*, *field=<django.db.models.fields.CharField: difficulty>*)

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

id

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

multiplechoiceexercise

Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Place.restaurant` is a `ReverseOneToOneDescriptor` instance.

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

status

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

status_message

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

title

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

topic

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.

topic_id

class core.models.**ExerciseInstanceInGeneralPath**(*id*, *user*, *exercise*, *solution*, *passed*)

exception DoesNotExist

exception MultipleObjectsReturned

exercise

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.

exercise_id

id

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

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

passed

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

solution

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

user

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.

user_id

class core.models.**MultipleChoiceExercise**(**args*, ***kwargs*)

Representa un ejercicio de opción múltiple en donde «answer» es la respuesta correcta del ejercicio y «options» son las posibles respuestas a la pregunta.

exception DoesNotExist

exception MultipleObjectsReturned

answer

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

exercise_ptr

Accessor to the related object on the forward side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`Restaurant.place` is a `ForwardOneToOneDescriptor` instance.

exercise_ptr_id

options

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

class core.models.**Subject**(**args*, ***kwargs*)

El modelo Subject son los temas que forman parte de la ruta general de aprendizaje. El modelo cuenta con un campo nombre y una funcion que regresa el nombre del tema.

exception DoesNotExist

exception MultipleObjectsReturned

completedlevelsperuser_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.

exercise_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.

from_relation

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.

id

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

name

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

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

to_relation

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 core.models.**SubjectRelation**(**args*, ***kwargs*)

El modelo SubjectRelation representa la relación que existe entre los temas de la ruta general. Al ser un grafo se decidió utilizar un sistema de llaves foraneas que ayuden a identificar el nodo fuente del nodo destino y asi poder representarlos posteriormente

exception DoesNotExist

exception MultipleObjectsReturned

destination

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.

destination_id

id

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

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

source

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.

source_id

core.models.**options_default**()

core.models.**tests_default**()

CAPÍTULO 7

Modelos Profesor

class professor.models.**Course**(**args*, ***kwargs*)

Este modelo representa un curso al cual se pueden inscribir los estudiantes. Está asociado a un profesor mediante el campo «admin» y a múltiples estudiantes mediante el campo «students»

exception DoesNotExist

exception MultipleObjectsReturned

created_by

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.

created_by_id

enrollment_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.

id

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

name

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

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

students

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.

class professor.models.**Enrollment**(**args*, ***kwargs*)

Este modelo representa la inscripción de un estudiante en un curso. Es una tabla intermedia que se utiliza para conectar los modelos «Course» y «CustomUser»

exception DoesNotExist

exception MultipleObjectsReturned

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

id

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

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

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

CAPÍTULO 8

Forms App Miembros

class members.forms.**CustomUserChangeForm**(**args*, ***kwargs*)

La clase CustomUserChangeForm es la clase que nosotros creamos para poder hacer uso de nuestro propio formulario con el objetivo de editar elementos de los usuarios dentro de la plataforma. Este hereda del formulario por defecto de Django que es UserChangeForm, y nosotros lo modificamos para que reconozca con que modelo estara trabajando de ahora en adelante»

Parámetros

[UserChangeForm] – [UserChangeForm es el formulario por defecto de Django, este se hereda en esta clase para que pueda ser modificado y vinculado a nuestro modelo personalizado]

class Meta

fields = '__all__'

model

alias de *CustomUser*

base_fields = {'date_joined': <django.forms.fields.DateTimeField object>, 'email': <django.forms.fields.EmailField object>, 'first_name': <django.forms.fields.CharField object>, 'groups': <django.forms.models.ModelMultipleChoiceField object>, 'is_active': <django.forms.fields.BooleanField object>, 'is_staff': <django.forms.fields.BooleanField object>, 'is_superuser': <django.forms.fields.BooleanField object>, 'last_login': <django.forms.fields.DateTimeField object>, 'last_name': <django.forms.fields.CharField object>, 'name': <django.forms.fields.CharField object>, 'password': <django.contrib.auth.forms.ReadOnlyPasswordHashField object>, 'role': <django.forms.fields.TypedChoiceField object>, 'user_permissions': <django.forms.models.ModelMultipleChoiceField object>}

declared_fields = {'password': <django.contrib.auth.forms.ReadOnlyPasswordHashField object>}

property media

Return all media required to render the widgets on this form.

class members.forms.**CustomUserCreationForm**(**args*, ***kwargs*)

La clase CustomUserCreationForm es la clase que nosotros creamos para poder hacer uso de nuestro propio formulario con el objetivo de registrar usuarios en la plataforma. Este hereda del formulario por defecto de Django que es UserCreationForm, y nosotros lo modificamos para que reconozca con que modelo estara trabajando de ahora en adelante»

Parámetros

[UserCreationForm] – [UserCreationForm es el formulario por defecto de Django, este se hereda en esta clase para que pueda ser modificado y vinculado a nuestro modelo personalizado]

class Meta

fields = ('email', 'name', 'last_name')

labels = {'email': 'Correo Electrónico', 'last_name': 'Apellidos', 'name': 'Nombre'}

model

alias de *CustomUser*

base_fields = {'email': <django.forms.fields.EmailField object>, 'last_name': <django.forms.fields.CharField object>, 'name': <django.forms.fields.CharField object>, 'password1': <django.forms.fields.CharField object>, 'password2': <django.forms.fields.CharField object>}

declared_fields = {'password1': <django.forms.fields.CharField object>, 'password2': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

CAPÍTULO 9

Forms App Core

class core.forms.**CodeInputForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

La clase CodeInputForm es la clase utilizada para tener nuestro editor de codigo disponible en la pagina principal. Nosotros utilizamos el widget de Django, django-ace para hacer uso de este editor de codigo, donde podemos especificar para que lenguaje de programacion lo necesitamos y elementos de diseño para su dispocision en la pagina principal.

Parámetros

[forms.Form] – [Utilizamos forms.Form para poder darle un estilo a nuestro editor de codigo, ademas de darle un elemento de modo para que resalte la sintaxis, en este caso de Python]

base_fields = {'code_input': <django.forms.fields.CharField object>}

declared_fields = {'code_input': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

class core.forms.**CodingExerciseInputForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

class Meta

fields = ['solution']

labels = {'solution': 'Ingresa tu Código'}

model

alias de *ExerciseInstanceInGeneralPath*

widgets = {'solution': <django_ace.widgets.AceWidget object>}

base_fields = {'solution': <django.forms.fields.CharField object>}

declared_fields = {}

property media

Return all media required to render the widgets on this form.

CAPÍTULO 10

Forms Estudiantes

class student.forms.**CourseJoinForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Formulario utilizado para la inscripción de un estudiante en un curso mediante el id del curso.

base_fields = {'course_id': <django.forms.fields.CharField object>}

declared_fields = {'course_id': <django.forms.fields.CharField object>}

property media

Return all media required to render the widgets on this form.

CAPÍTULO 11

Admin App Miembros

class members.admin.**CustomUserAdmin**(*model*, *admin_site*)

La clase CustomUserAdmin utiliza al admin de Django para poder hacerle las modificaciones necesarias el modelo de usuario que nosotros creamos. Aqui se definen los formularios que Django debe de utilizar para este tipo de usuarios, asi como que campos podran ser visibles y editables desde la pagina de admin de Django. Al final, registramos los elementos para que la pagina de administrador de Django reconozca los cambios realizados

Parámetros

[UserAdmin] – [UserAdmin es el administrador por defecto de Django, nosotros lo heredamos para poder hacer las modificaciones necesarias para nuestro modelo custom]

add_fieldsets = (('Personal Info', {'classes': ('wide',), 'fields': ('name', 'last_name', 'email', 'role', 'password1', 'password2', 'is_staff', 'is_active')}),)

add_form

alias de *CustomUserCreationForm*

fieldsets = (('Personal Info', {'fields': ('email', 'password', 'role')}), ('Permissions', {'fields': ('is_staff', 'is_active', 'is_superuser', 'groups', 'user_permissions')}), ('Dates', {'fields': ('last_login', 'date_joined')}))

form

alias de *CustomUserChangeForm*

list_display = ('email', 'name', 'last_name', 'role')

property media

model

alias de *CustomUser*

ordering = ('email',)

search_fields = ('email',)

CAPÍTULO 12

Manager App Miembros

class members.manager.**CustomUserManager**(**args*, ***kwargs*)

La clase CustomUserManager es una clase de igual manera creada para poder hacer uso del modelo personalizado de usuario que creamos. En este caso en especifico esta hereda de la clase BaseUserManager, clase por defecto de Django para manejar a los usuarios. En esta clase nosotros modificamos que el campo unico que identifique a nuestros usuarios sea su email, no su nombre de usuario.»

Parámetros

[BaseUserManager] – [BaseUserManager es el manager por defecto de Django para su modelo de usuario, nosotros lo heredamos para poder modificarlo y especificar campos importantes para nosotros]

create_superuser(*email*, *password*, ***extra_fields*)

create_user(*email*, *password*, ***extra_fields*)

Índice de Módulos Python

Índice

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W