In the Name of God

Web Programming

Homework 1

Lecturer
M. Nazari

Authors
Hirad Davari | 99106136
Mahdi Saadatbakht | 99105475
Soheil Nazari Mendejin | 99102412

Sharif University of Technology
October 2023

University Website Models Implementation Guide

This guide will walk you through the implementation of the Django models for a university website. The code defines various models such as Student, Department, Professor, Course, Enrollment, Classroom, Schedule, and Assignment along with their relationships.

1. Setting Up Django

Make sure you have Django installed. If not, you can install it using pip:

```
pip install django
```

2. Creating a Django Project

If you don't have an existing Django project, create one using the following command:

```
django-admin startproject web_hw1
```

3. Creating a Django App

Inside your project directory, create a new app to house the university models:

```
cd web_hw1
python manage.py startapp university
```

4. Add university app to INSTALLED_APPS

Inside the web_hw1 directory, there is a `settings.py` file. Inside that file, there is a variable named `INSTALLED_APPS`. Add university app to this variable:

```
INSTALLED_APPS = [
    # ...
    "university",
]
```

5. Define Models

5.1 Student Model

```
class Student(models.Model):
    MAJOR_CHOICES = (
        ('CE', 'Computer Engineering'), ('CS', 'Computer Science'),
        ('EE', 'Electrical Engineering'), ('MSE', 'Material Science Engineering'),
        ('AE', 'Aerospace Engineering'), ('ME', 'Mechanical Engineering'),
        ('CHE', 'Chemical Engineering'), ('IE', 'Industrial Engineering'),
        ('AM', 'Applied Mathematics'), ('P', 'Physics'), ('E', 'Economics')
    )
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    student_number = models.CharField(max_length=20, unique=True)
    major = models.CharField(max_length=20, choices=MAJOR_CHOICES)
    enrollment_year = models.PositiveIntegerField()

    def __str__(self):
        return f"{self.first_name} {self.last_name} - {self.student_number}"
```

In the Student model, we define several fields for student information. Here's why we use these options:

- `first_name` and `last_name` are defined as character fields to store the first and last names of the student.
- `student_number` is defined as a character field with `unique=True` to ensure that each student has a unique identifier.
- `major` is a character field to store the student's major.
- `enrollment_year` is an integer field to store the year when the student enrolled.

5.2 Department Model

```
class Department(models.Model):
    DEPARTMENT_CHOICES = (
        ('CE', 'Computer Engineering'), ('EE', 'Electrical Engineering'),
        ('MSE', 'Material Science Engineering'), ('AE', 'Aerospace Engineering'),
        ('ME', 'Mechanical Engineering'), ('CHE', 'Chemical Engineering'),
        ('IE', 'Industrial Engineering'), ('MS', 'Mathematical Science'),
        ('P', 'Physics'), ('E', 'Economics')
    )
    name = models.CharField(max_length=100, choices=DEPARTMENT_CHOICES)
    head_of_department = models.ForeignKey(to:'Professor', on_delete=models.SET_NULL, null=True,
                                           blank=True, related_name='department_head')

    def __str__(self):
        return self.name
```

In the Department model, we define fields to represent a department. Here's why we use these options:

- `name` is a character field to store the name of the department.
- `head_of_department` is a `ForeignKey` field that establishes a relationship with the `Professor` model. We use `on_delete=models.SET_NULL` to specify that if a professor (head of department) is deleted, their reference in the department should be set to `NULL`. We also add `null=True` to allow for cases where a department might not have a head.
- `related_name='department_head'` is used to provide a custom reverse query name for the professor who is the head of the department. This helps prevent a naming clash with the related field in the `Professor` model.

5.3 Professor Model

```python
class Professor(models.Model):
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    staff_number = models.CharField(max_length=20, unique=True)
    hiring_date = models.DateField()
    department = models.ForeignKey( to: 'Department', on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.first_name} {self.last_name} - {self.staff_number}"
```

In the `Professor` model, we define fields for professor information. Here's why we use these options:

- `first_name` and `last_name` are character fields to store the first and last names of the professor.
- `staff_number` is a character field with `unique=True` to ensure each professor has a unique identifier.
- `hiring_date` is a `DateField` to store the date when the professor was hired.
- `department` is a `ForeignKey` field that establishes a relationship with the `Department` model. We use `on_delete=models.CASCADE` to specify that if a department is deleted, the associated professor(s) should also be deleted.

5.4 Course Model

```python
class Course(models.Model):
    course_name = models.CharField(max_length=255)
    course_code = models.CharField(max_length=20, unique=True)
    unit_count = models.PositiveIntegerField()
    offered_by = models.ForeignKey( to: 'Professor', on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.course_name} ({self.course_code}-{self.unit_count})"
```

In the Course model, we define fields to represent a course. Here's why we use these options:

- `course_name` is a character field to store the name of the course.
- `course_code` is a character field with `unique=True` to ensure each course has a unique code.
- `unit_count` is an integer field to store the number of units for the course.
- `offered_by` is a `ForeignKey` field that establishes a relationship with the `Professor` model, indicating which professor teaches the course. We use `on_delete=models.CASCADE` to specify that if a professor is deleted, the associated courses should also be deleted.

5.5 Enrollment Model

```
class Enrollment(models.Model):
    SEMESTER_CHOICES = [('Spring', 'Spring'), ('Fall', 'Fall')]
    enrolled_student = models.ForeignKey( to: 'Student', on_delete=models.CASCADE)
    enrolled_course = models.ForeignKey( to: 'Course', on_delete=models.CASCADE)
    semester = models.CharField(max_length=20, choices=SEMESTER_CHOICES)

    def __str__(self):
        return f"{self.enrolled_student} - {self.enrolled_course} ({self.semester})"
```

In the Enrollment model, we define fields to represent a student's enrollment in a course. Here's why we use these options:

- `student` is a `ForeignKey` field that establishes a relationship with the `Student` model, representing the student who is enrolled in the course. We use `on_delete=models.CASCADE` to specify that if a student or course is deleted, the enrollment record should be deleted.
- `course` is a `ForeignKey` field that establishes a relationship with the `Course` model, indicating which course the student is enrolled in. We use `on_delete=models.CASCADE` for the same reasons as above.
- `semester` is a character field with `choices` to store the semester when the enrollment occurs.

5.6 Classroom

```
class Classroom(models.Model):
    building = models.CharField(max_length=30)
    class_number = models.PositiveIntegerField()
    capacity = models.PositiveIntegerField()
    projector_available = models.BooleanField(default=False)
    whiteboard_available = models.BooleanField(default=True)
    wheelchair_accessible = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.building} - Room {self.class_number}"
```

In the `Classroom` model, we define fields to represent a classroom. Here's why we use these options:

- `building` is a character field to store the name of the building where the classroom is located.
- `class_number` is an integer field to store the classroom number.
- `capacity` is an integer field to store the maximum seating capacity of the classroom.
- `projector_available`, `whiteboard_available`, and `wheelchair_accessible` are boolean fields that indicate whether specific amenities are available in the classroom. We set default values to `False` for `projector_available and wheelchair_accessible`, and `True` for `whiteboard_available`.

5.7 Schedule

```
class Schedule(models.Model):
    DAY_OF_WEEK_CHOICES = (
        ('Saturday', 'Saturday'), ('Sunday', 'Sunday'), ('Monday', 'Monday'),
        ('Tuesday', 'Tuesday'), ('Wednesday', 'Wednesday'), ('Thursday', 'Thursday'),
    )
    course = models.ForeignKey( to: 'Course', on_delete=models.CASCADE)
    professor = models.ForeignKey( to: 'Professor', on_delete=models.CASCADE)
    day_of_week = models.CharField(max_length=10, choices=DAY_OF_WEEK_CHOICES)
    start_time = models.TimeField()
    end_time = models.TimeField()
    location = models.ForeignKey( to: 'Classroom', on_delete=models.SET_NULL)

    def get_day_of_week_display(self):
        return f"On {self.day_of_week.capitalize()}"

    def __str__(self):
        return f"{self.course} | {self.professor} | {self.get_day_of_week_display()} | " \
               f"{self.start_time}-{self.end_time}"
```

In the Schedule model, we define fields to represent the schedule of courses. Here's why we use these options:

- `DAY_OF_WEEK_CHOICES` is a tuple of choices for the days of the week.
- `course` is a `ForeignKey` field that establishes a relationship with the `Course` model, indicating which course is scheduled.
- `professor` is a `ForeignKey` field that establishes a relationship with the `Professor` model, indicating which professor is responsible for the course.
- `day_of_week` is a character field with choices to specify the day of the week when the course is scheduled.
- `start_time` and `end_time` are `TimeField` to specify the start and end times of the course.
- `location` is a `ForeignKey` field that establishes a relationship with the `Classroom` model to indicate the location of the course. We use `on_delete=models.SET_NULL` to allow for cases where a location might change or become unavailable. The `get_day_of_week_display` method provides a more readable display for the day of the week, capitalizing the day's name.

The `__str__` method provides a custom string representation for each schedule entry, displaying the course, professor, day of the week, and the time range.

5.8 Assignment

```python
class Assignment(models.Model):
    course = models.ForeignKey( to: 'Course', on_delete=models.CASCADE)
    number_of_assignment = models.PositiveIntegerField()
    topic = models.CharField(max_length=30)
    deadline = models.DateTimeField()

    def get_ordinal_number_of_assignment(self):
        if self.number_of_assignment % 10 == 1:
            return f"{self.number_of_assignment}st"
        elif self.number_of_assignment % 10 == 2:
            return f"{self.number_of_assignment}nd"
        elif self.number_of_assignment % 10 == 3:
            return f"{self.number_of_assignment}rd"
        else:
            return f"{self.number_of_assignment}th"

    def __str__(self):
        return f"{self.get_ordinal_number_of_assignment()} Assignment from {self.course} Course, " \
               f"with topic of {self.topic} and deadline of {self.deadline}"
```

In the `Assignment` model, we define fields to represent assignments. Here's why we use these options:

- `course` is a `ForeignKey` field that establishes a relationship with the `Course` model, indicating which course the assignment belongs to.
- `number_of_assignment` is an integer field to specify the order of the assignment within the course.
- `topic` is a character field to store the topic or title of the assignment.
- `deadline` is a `DateTimeField` to specify the due date and time for the assignment.

The `get_ordinal_number_of_assignment` method is used to generate a human-readable ordinal representation for the assignment number (e.g., 1st, 2nd, 3rd, 4th, etc.).

6. Admin Panel

To manage the data easily, register the models in the admin panel. In the `admin.py` file within the university app, add the following code:

```
from django.contrib import admin
from .models import Student, Department, Professor, Course,
Enrollment, Classroom, Schedule, Assignment

admin.site.register(Student)
admin.site.register(Department)
admin.site.register(Professor)
admin.site.register(Course)
admin.site.register(Enrollment)
admin.site.register(Classroom)
admin.site.register(Schedule)
admin.site.register(Assignment)
```

7. Migrations

Create and apply migrations for the models using the following commands:

```
python manage.py makemigrations
python manage.py migrate
```

8. Create a super user

Run the command below to create a super user for accessing the admin panel:

```
python manage.py createsuperuser
```

This command asks for a username, email, and password. Remember the username and password. We want to use it later on.

9. Run the Development Server

Start the development server to run the Django application:

```
python manage.py runserver
```

You can now access the Django admin panel by visiting http://127.0.0.1:8000/admin/ in your web browser. Log in with your superuser credentials to add, update, and manage data on the university website. Now you can add and manage students, departments, professors, courses, enrollments, classrooms, schedules, and assignments through the admin panel.