CREATING AN ONLINE LEARNING SYSTEM WITH DJANGO 3

Lobar Burhonovna
Badalova Dehqonova
Dehqonova Hayotxon Muhtorjon qizi
Niyozov Bekzod Davlatali o`gli
Master's Students at the National University of Uzbekistan

Abstract: *This article provides a comprehensive guide on building an online learning system using Django 3. While the website is still under development, the article offers valuable insights and practical guidance for developers and designers working on e-learning platforms.*

Keywords: *Django 3, web development, e-learning platform, database design, Django models, data normalization, database relationships, constraints in Django, online education, quizzes, multimedia content, user authentication, user roles, student enrollment, advanced Django features, educational technology, children's education.*

INTRODUCTION:

In Uzbekistan, challenges like outdated curricula and insufficient resources hinder the delivery of high-quality education to young children. This article focuses on leveraging Django 3 to create an e-learning platform, emphasizing not only the database design but also the development of views, templates, and the application of advanced features. The goal is to empower developers to contribute to the transformation of the education landscape by providing a modern, accessible, and engaging online learning experience.

MAIN PART:

1. Setting Up Django Project:

To setup we should install Django and create a new project. The following commands are used to do them:

```
pip install django
django-admin startproject elearning_platform
```

2. Planning Database Models:

After drawing database schema, we should represent these entities in django models, keeping in mind the relationships between them. For instance, creating models for courses, students, teachers, enrollments, payment, assignments, quizzes, and more. The power of Django's ORM can be utilized to define these models concisely and intuitively.

```
# models.py
from django.db import models
class Course(models.Model):
    title = models.CharField(max_length=100)
    description = models.TextField()
```

```
class Student(models.Model):
    name = models.CharField(max_length=50)
    contact_info = models.CharField(max_length=20)
class Quiz(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    title = models.CharField(max_length=100)
class Question(models.Model):
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE)
    text = models.TextField()
class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    text = models.CharField(max_length=100)
    is_correct = models.BooleanField(default=False)
# ... Additional models for teachers, enrollments...
```

3. Database Migration:

After defining models, we will create and apply database migrations to set up the initial database schema. Django's migration system allows you to version-control your database schema, making it easy to evolve your models over time.

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

4. Creating Views and Templates:

After that we'll develop views to render templates for different aspects of the e-learning platform, such as courses, student profiles, quizzes, assignments, etc. Django follows the MVC (Model-View-Controller) architectural pattern, where views handle the logic and templates handle the presentation.

```
# views.py
from django.shortcuts import render, get_object_or_404
from .models import Course, Quiz, Question
def course_list(request):
    courses = Course.objects.all()
    return render(request, 'courses/course_list.html', {'courses': courses})

def quiz_detail(request, quiz_id):
    quiz = get_object_or_404(Quiz, pk=quiz_id)
    questions = Question.objects.filter(quiz=quiz)
    return render(request, 'quizzes/quiz_detail.html', {'quiz': quiz, 'questions': questions})

# ... Additional views for students, teachers, enrollments, payment, assignments
```

Corresponding templates should be created in the 'templates' folder. Django's template language allows us to embed dynamic content seamlessly within HTML, making it easy to create dynamic and interactive pages.

5. Normalization and Relationships:

Data normalization should be ensured by establishing relationships between models. We have to use ForeignKey for relationships like enrollments. Django's ORM simplifies the creation of relationships between models, handling the underlying SQL queries transparently.

```
# models.py
class Enrollment(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    # ... Additional fields for enrollment details

class Question(models.Model):
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE)
    text = models.TextField()

# ... Additional models for relationships
```

We can use ManyToManyField for more complex relationships.

6. Adding Constraints:

We have to implement constraints using Django model options. For example, enforce unique constraints on student enrollments. Constraints help maintain data integrity, ensuring that the data in our database follows predefined rules.

```
# models.py
class Enrollment(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.CASCADE, unique=True)
    # ... Additional fields for enrollment details

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    text = models.CharField(max_length=100)
    is_correct = models.BooleanField(default=False)
```

```
    class Meta:
        unique_together = ('question', 'is_correct')
```

7. Implementing Advanced Features:

We can enhance the platform by adding features like user authentication, user roles (student, teacher, admin), and content delivery through multimedia. Django provides a robust authentication system out of the box, allowing you to secure your application effortlessly.

```
# views.py
from django.contrib.auth.decorators import login_required

@login_required
def my_courses(request):
    student = request.user.student
    courses = Enrollment.objects.filter(student=student).select_related('course')
    return render(request, 'courses/my_courses.html', {'courses': courses})
```

8. Introducing Quizzes:

We can also extend the e-learning platform's interactivity by introducing quizzes. Develop views, templates, and models for quizzes, questions, and choices. Allow students to submit answers and provide immediate feedback on quiz completion.

CONCLUSION:

In conclusion, creating an e-learning platform with Django 3 involves setting up the project, planning and implementing database models, migrating the database, creating views and templates, ensuring normalization, relationships, and constraints, implementing advanced features, and introducing quizzes. This comprehensive approach ensures a robust, scalable, and feature-rich e-learning platform.

REFERENCES:

1.Django Documentation. (n.d.). https://docs.djangoproject.com/

2.W3Schools. (n.d.). Django Models. https://www.w3schools.com/django/django_models.asp

3. Mozilla Developer Network. (n.d.). Django ForeignKey. [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Models#adding-a-