**Understanding the New Requirements**

**Programs and Structure**

* **Programs**:
  + 3-year LLB (6 semesters).
  + 5-year BALLB (10 semesters).
* **Subjects**:
  + 30 subjects common to both programs (law subjects).
  + 20 additional non-law subjects for BALLB (Years 1–3).
* **Sections**:
  + Year 1 (BALLB and LLB): Two sections (A and B).
  + Years 2–5 (BALLB) and Years 2–3 (LLB): One section each.
* **Lectures**:
  + 5 daily lectures: 08:30 AM, 09:30 AM, 10:30 AM, 12:00 PM, 01:00 PM.
  + Each lecture is 1 hour (implied).

**Attendance Requirements**

* **Default**: Absent, marked Present during attendance filling.
* **Display**: Consolidated attendance by roll number (Month/Day/Week-wise).
  + Example: G240001 - P P A P A (P = Present, A = Absent).
* **Stats**: Number of students present per class/hour and consolidated statement.
* **Roll Numbers**:
  + BALLB: G24xxxx (e.g., G240001).
  + LLB: NG24xxxx (e.g., NG240012).

**Timetable and Subjects**

* **Timetable**: Fixed daily slots (5 lectures) for each course/section.
* **Subjects**: Listed per semester for both programs (30 law + 20 non-law for BALLB).

**Rewritten Django ORM (core/models.py)**

I’ll design the ORM to handle programs, sections, subjects, timetable, sessions, and attendance, with authentication linked to Django’s User model. Here’s the fresh implementation:

python

CollapseWrapCopy

from django.db import models

from django.contrib.auth.models import User

from django.core.exceptions import ValidationError

from django.core.validators import RegexValidator

*# Validate lecture time slots*

LECTURE\_SLOTS = [

('08:30:00', '08:30 AM - 09:30 AM'),

('09:30:00', '09:30 AM - 10:30 AM'),

('10:30:00', '10:30 AM - 11:30 AM'),

('12:00:00', '12:00 PM - 01:00 PM'),

('13:00:00', '01:00 PM - 02:00 PM'),

]

class Program(models.Model):

name = models.CharField(max\_length=50, unique=True) *# e.g., "BALLB 5 Yr", "LLB 3 Yr"*

duration\_years = models.PositiveIntegerField() *# 3 or 5*

def \_\_str\_\_(self):

return self.name

class Meta:

verbose\_name = "Program"

verbose\_name\_plural = "Programs"

class Section(models.Model):

program = models.ForeignKey(Program, on\_delete=models.CASCADE, related\_name="sections")

name = models.CharField(max\_length=10) *# e.g., "Section A", "Section B"*

year = models.PositiveIntegerField() *# 1, 2, 3, 4, 5*

def \_\_str\_\_(self):

return f"{self.program} - Year {self.year} - {self.name}"

class Meta:

unique\_together = ('program', 'year', 'name')

verbose\_name = "Section"

verbose\_name\_plural = "Sections"

class Subject(models.Model):

name = models.CharField(max\_length=100, unique=True) *# e.g., "Law of Contract I"*

is\_law\_subject = models.BooleanField(default=True) *# True for law, False for non-law*

semester = models.PositiveIntegerField() *# 1 to 10 (depending on program)*

def \_\_str\_\_(self):

return self.name

class Meta:

verbose\_name = "Subject"

verbose\_name\_plural = "Subjects"

class Teacher(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE) *# Authentication*

first\_name = models.CharField(max\_length=50)

last\_name = models.CharField(max\_length=50)

email = models.EmailField(unique=True)

phone = models.CharField(max\_length=15, null=True, blank=True)

def \_\_str\_\_(self):

return f"{self.first\_name} {self.last\_name}"

class Meta:

verbose\_name = "Teacher"

verbose\_name\_plural = "Teachers"

class Student(models.Model):

roll\_number = models.CharField(

max\_length=10,

unique=True,

validators=[RegexValidator(r'^(G|NG)24\d{4}$', 'Roll number must be G24xxxx or NG24xxxx')]

) *# e.g., G240001, NG240012*

first\_name = models.CharField(max\_length=50)

last\_name = models.CharField(max\_length=50)

email = models.EmailField(unique=True)

section = models.ForeignKey(Section, on\_delete=models.CASCADE, related\_name="students")

subjects = models.ManyToManyField(Subject, related\_name="students", blank=True)

def \_\_str\_\_(self):

return f"{self.roll\_number} - {self.first\_name} {self.last\_name}"

class Meta:

verbose\_name = "Student"

verbose\_name\_plural = "Students"

class Timetable(models.Model):

DAY\_CHOICES = [

('Monday', 'Monday'),

('Tuesday', 'Tuesday'),

('Wednesday', 'Wednesday'),

('Thursday', 'Thursday'),

('Friday', 'Friday'),

('Saturday', 'Saturday'),

]

section = models.ForeignKey(Section, on\_delete=models.CASCADE, related\_name="timetable")

subject = models.ForeignKey(Subject, on\_delete=models.CASCADE, related\_name="timetable")

teacher = models.ForeignKey(Teacher, on\_delete=models.SET\_NULL, null=True, related\_name="timetable")

day\_of\_week = models.CharField(max\_length=9, choices=DAY\_CHOICES)

start\_time = models.TimeField(choices=LECTURE\_SLOTS) *# Fixed lecture slots*

semester\_start\_date = models.DateField()

semester\_end\_date = models.DateField()

def \_\_str\_\_(self):

return f"{self.section} - {self.subject} ({self.day\_of\_week} {self.start\_time})"

class Meta:

unique\_together = ('section', 'subject', 'day\_of\_week', 'start\_time')

verbose\_name = "Timetable"

verbose\_name\_plural = "Timetables"

class Session(models.Model):

STATUS\_CHOICES = [

('Scheduled', 'Scheduled'),

('Completed', 'Completed'),

('Cancelled', 'Cancelled'),

]

timetable = models.ForeignKey(Timetable, on\_delete=models.CASCADE, related\_name="sessions")

date = models.DateField()

status = models.CharField(max\_length=10, choices=STATUS\_CHOICES, default='Scheduled')

def \_\_str\_\_(self):

return f"{self.timetable} on {self.date} ({self.status})"

class Meta:

unique\_together = ('timetable', 'date')

verbose\_name = "Session"

verbose\_name\_plural = "Sessions"

class Attendance(models.Model):

STATUS\_CHOICES = [

('Present', 'Present'),

('Absent', 'Absent'),

]

student = models.ForeignKey(Student, on\_delete=models.CASCADE, related\_name="attendance")

session = models.ForeignKey(Session, on\_delete=models.CASCADE, related\_name="attendance")

status = models.CharField(max\_length=7, choices=STATUS\_CHOICES, default='Absent')

timestamp = models.DateTimeField(auto\_now=True)

recorded\_by = models.ForeignKey(Teacher, on\_delete=models.SET\_NULL, null=True, related\_name="attendance\_records")

def \_\_str\_\_(self):

return f"{self.student} - {self.session} - {self.status}"

class Meta:

unique\_together = ('student', 'session')

verbose\_name = "Attendance"

verbose\_name\_plural = "Attendance"

class CalendarException(models.Model):

date = models.DateField(unique=True)

description = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return f"{self.date} - {self.description}"

class Meta:

verbose\_name = "Calendar Exception"

verbose\_name\_plural = "Calendar Exceptions"

**Explanation of Changes and Additions**

**1. Program**

* **Purpose**: Represents the 3-year LLB and 5-year BALLB programs.
* **Fields**: name (e.g., "BALLB 5 Yr"), duration\_years (3 or 5).
* **Why**: Tracks the two distinct programs.

**2. Section**

* **Purpose**: Manages sections (A, B, or single section) per year and program.
* **Fields**: program, name (e.g., "Section A"), year (1–5).
* **Constraints**: Unique per program and year.
* **Why**: Supports Year 1 (A & B) and Years 2–5 (single section) as specified.

**3. Subject**

* **Purpose**: Stores all subjects (50 total: 30 law + 20 non-law).
* **Fields**: name, is\_law\_subject (to differentiate), semester (1–10).
* **Why**: Tracks subjects across semesters and programs.

**4. Teacher**

* **Purpose**: Represents instructors with authentication.
* **Fields**: Linked to User for login, plus personal details.
* **Why**: Teachers need secure access to mark attendance.

**5. Student**

* **Purpose**: Represents students with program-specific roll numbers.
* **Fields**: roll\_number (regex for G24xxxx or NG24xxxx), section, subjects (many-to-many).
* **Why**: Links students to sections and subjects; roll number format matches your example.

**6. Timetable**

* **Purpose**: Schedules lectures for each section.
* **Fields**: section, subject, teacher, day\_of\_week, start\_time (fixed slots), semester\_dates.
* **Constraints**: Unique per section, subject, day, and time.
* **Why**: Supports 5 daily lectures (08:30 AM–02:00 PM) per section.

**7. Session**

* **Purpose**: Represents individual lecture instances.
* **Fields**: timetable, date, status.
* **Why**: Tracks each scheduled lecture for attendance.

**8. Attendance**

* **Purpose**: Records attendance per student per session.
* **Fields**: student, session, status (default Absent), timestamp, recorded\_by.
* **Why**: Default Absent, marked Present as per your request; supports hourly tracking.

**9. CalendarException**

* **Purpose**: Manages holidays to cancel sessions.
* **Fields**: date, description.
* **Why**: Ensures holiday-adjusted attendance.

**How It Meets Your Requirements**

* **Programs/Sections**: Program and Section handle BALLB (Years 1–5) and LLB (Years 1–3) with A/B sections in Year 1.
* **Subjects**: 50 subjects (30 law + 20 non-law) stored in Subject, assignable via Timetable.
* **Timetable**: Fixed 5 lecture slots daily, configurable per section.
* **Attendance**: Default Absent, consolidated display possible via queries (e.g., G240001 - P P A P A).
* **Stats**: Aggregatable via Django queries (e.g., present count per hour).

**Next Steps**

1. **Setup Django**:
   * python manage.py makemigrations
   * python manage.py migrate
2. **Sample Data**:

python

CollapseWrapCopy

from django.contrib.auth.models import User

from core.models import Program, Section, Subject, Teacher, Student, Timetable, Session

*# Programs*

ballb = Program(name="BALLB 5 Yr", duration\_years=5)

llb = Program(name="LLB 3 Yr", duration\_years=3)

ballb.save(); llb.save()

*# Sections*

Section(program=ballb, name="Section A", year=1).save()

Section(program=ballb, name="Section B", year=1).save()

Section(program=ballb, name="Section A", year=2).save()

Section(program=llb, name="Section A", year=1).save()

*# Subjects (example)*

Subject(name="Law of Contract I", is\_law\_subject=True, semester=1).save()

*# Teacher*

user = User.objects.create\_user(username="john.doe", password="password123")

teacher = Teacher(user=user, first\_name="John", last\_name="Doe", email="john.doe@example.com")

teacher.save()

1. **API Routes**: I’ll rewrite routes once you confirm the ORM is okay.

**Feedback**

* **Details Okay?**: Does this match your vision? Any fields or relationships to add/remove?
* **Next**: Shall I proceed with API routes or tweak the ORM further?

Let me know what you think!