Django Learning Management System (LMS)

A Formal Verification Study

Author

Marcu-Cristian Petric

Team Members

Dan-Cosmin Savuț
George-Daniel Rus
Marcu-Cristian Petric

Group 30432

January 10, 2025

Abstract

This document presents a comprehensive analysis of a Learning Management System (LMS) implemented using the Django framework. The system is formally verified using the PRISM model checker to ensure critical security and functional properties. Key aspects include user authentication, course enrollment, section progression, and final examination processes. The model checking results demonstrate the system's compliance with essential safety and liveness properties.

Contents

1 Design

1.1 Use Case Diagram

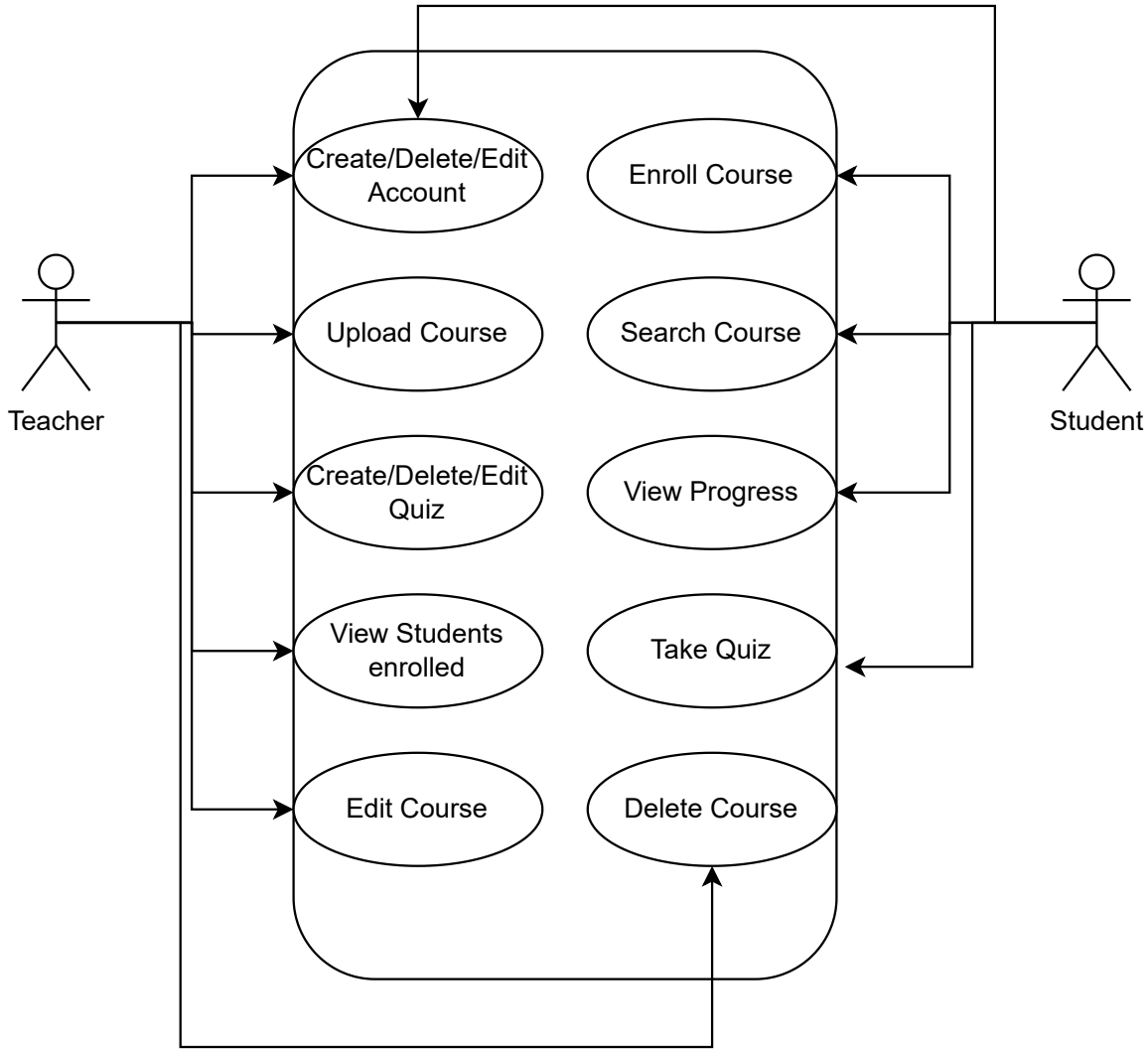

Figure 1.1: Use Case Diagram

1.2 Class Diagram

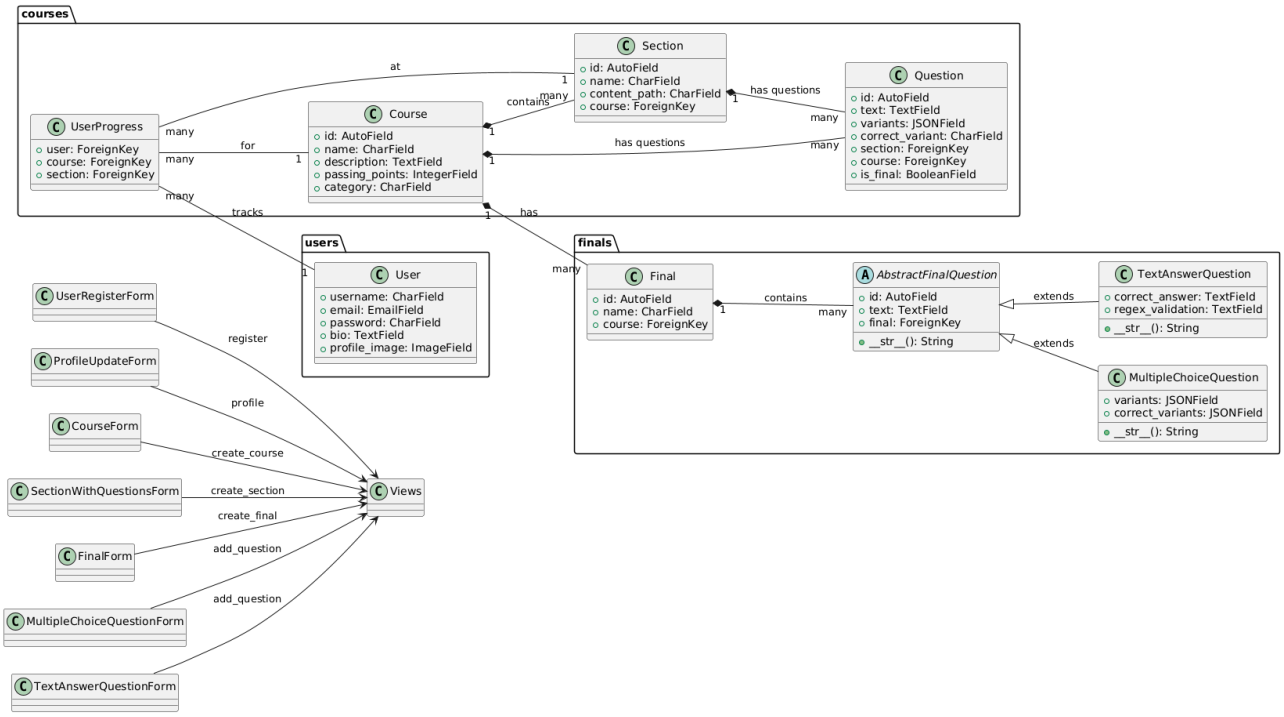

Figure 1.2: Class Diagram

1.3 Deployment Diagram

Before diving into our deployment setup, let's understand what a production environment typically uses [1]:

- **Nginx:** A high-performance web server that handles static file serving and request routing [2]. It manages incoming HTTP requests and efficiently directs them to the appropriate application server.
- **Gunicorn:** A Python WSGI HTTP Server for UNIX [3], designed to serve Django applications in production environments. It manages multiple worker processes to handle concurrent requests efficiently.

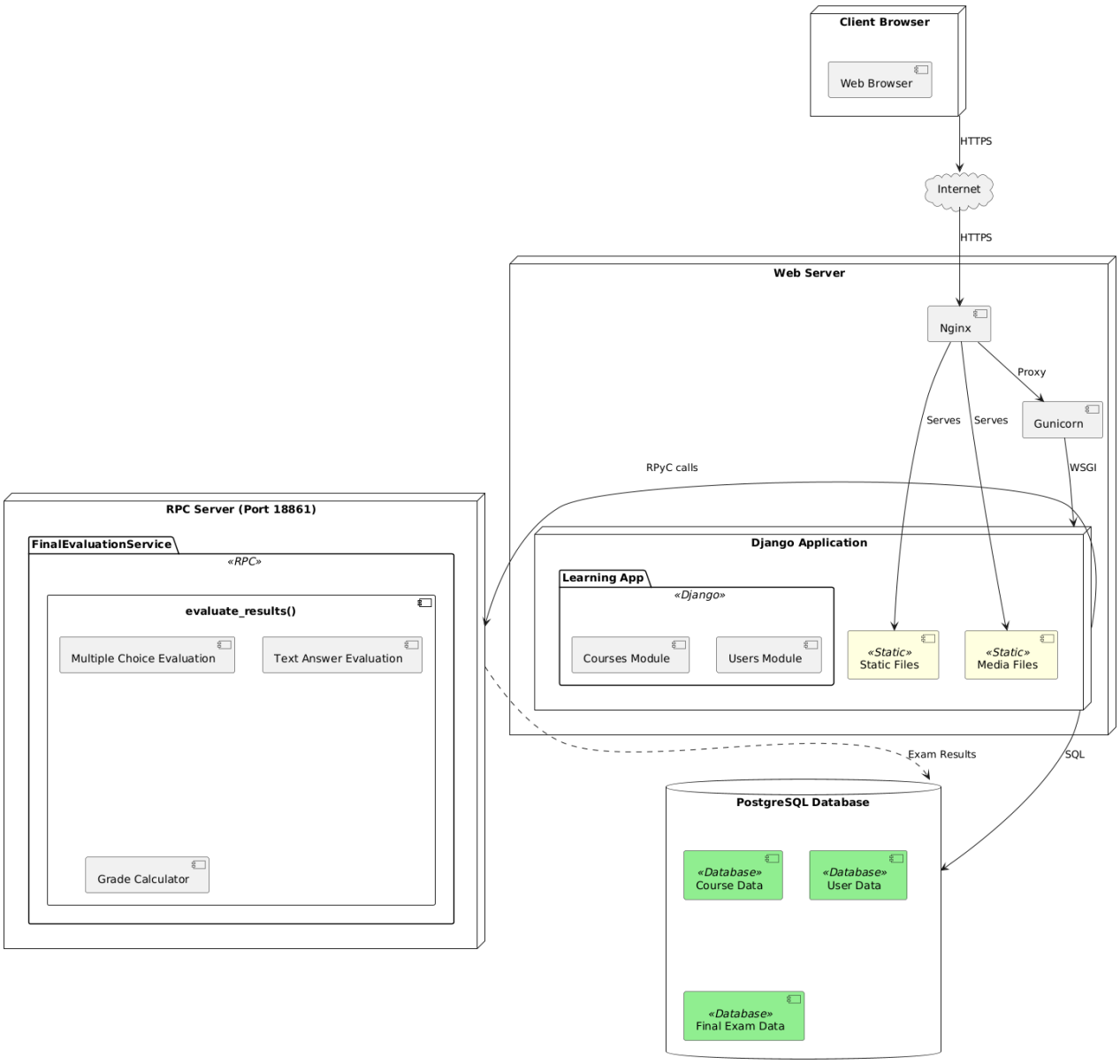

Figure 1.3: Deployment Diagram

2 PRISM model

2.1 Design

2.1.1 Model

Our PRISM model represents a simple learning system where students can take courses [4]. The model checking approach follows established methodologies for verifying concurrent systems [5], particularly focusing on educational platforms [6].

Students start by logging in (authentication) and then can join a course (enrollment). Each course has multiple sections, and to complete a section, students need to answer questions correctly [7]. The progression is strictly sequential, enforcing educational prerequisites. Students can also create and upload their own courses, sharing their knowledge with others.

After finishing all sections, there's a final exam with multiple questions. To pass the course, students need to achieve a minimum score set by the teacher [4]. This structured approach allows formal verification of critical system properties [5].

The model keeps track of:

- Whether a student is logged in
- If they're enrolled in a course
- Which section they're currently in
- How many questions they've answered correctly
- Their final exam score

We made sure students can't cheat by:

- Preventing double enrollment
- Making sure they complete each section before moving on
- Not letting them see the final exam until they're ready
- Keeping their progress safe when they log out

It's basically like having a virtual teaching assistant that makes sure everyone follows the rules and completes the course properly!

2.1.2 States and Transitions

The model is defined as a Discrete-Time Markov Chain (DTMC) with state space S where each state $s \in S$ is a tuple:

- $s = (auth,$
  $enrolled,$
  $course\_created,$
  $has\_sections,$
  $current\_section,$
  $correct\_answers,$

$final\_points$,
$answers\_submitted$,
$final\_exam\_in\_progress$,
$progress\_saved$,
$previous\_section$)

- $auth, enrolled, course\_created, has\_sections \in \{0, 1\}$
- $current\_section \in \{0, ..., 4\}$
- $correct\_answers \in \{0, 1, 2\}$
- $final\_points \in \{0, ..., 10\}$
- $answers\_submitted, final\_exam\_in\_progress, progress\_saved \in \{0, 1\}$
- $previous\_section \in \{0, ..., 3\}$

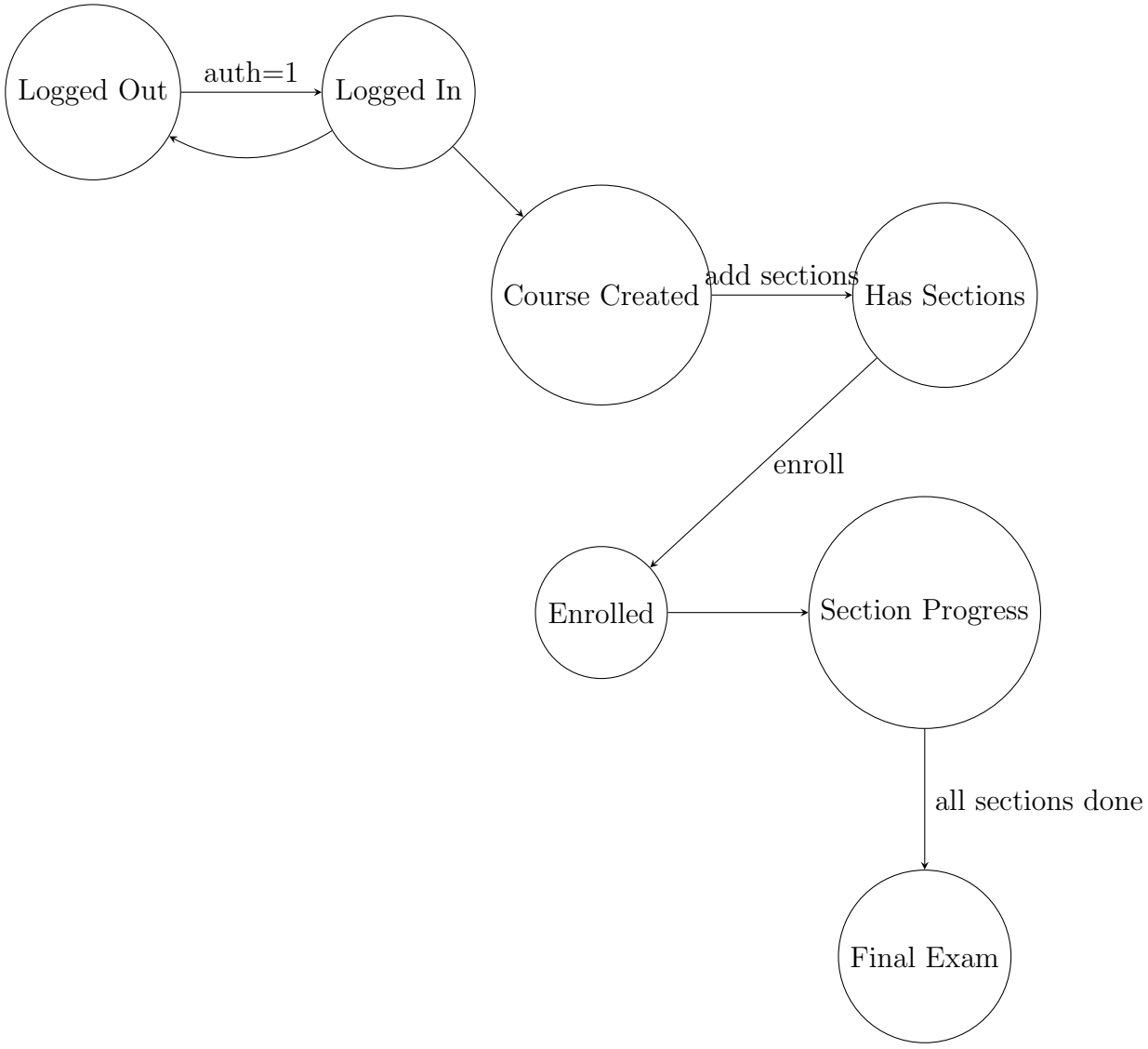

Figure 2.1: State Transition Diagram for Learning System

2.1.3 Properties

The properties are expressed in PCTL (Probabilistic Computation Tree Logic):

- Enrollment Properties:
  - $P > 0\big[F(\mathit{enrolled} = 1 \land \mathit{previous\_section} > 0)\big]$
  - $P \leq 0\big[F(\mathit{enrolled} = 1 \land \mathit{auth} = 0)\big]$
  - $P \leq 0\big[F(\mathit{enrolled} = 0 \land \mathit{current\_section} > 0)\big]$
  - $P \leq 0\big[F(\mathit{has\_sections} = 0 \land \mathit{enrolled} = 1)\big]$
- Course Management:
  - $P \leq 0\big[F(\mathit{course\_created} = 1 \land \mathit{auth} = 0)\big]$
  - $P \leq 0\big[F(\mathit{has\_sections} = 1 \land \mathit{course\_created} = 0)\big]$
- Progress Properties:
  - $P \leq 0\big[F(\mathit{current\_section} > \mathit{previous\_section} + 1)\big]$
  - $$P \leq 0\big[F(\mathit{auth} = 0 \land (\mathit{correct\_answers} > 0 \lor \mathit{final\_exam\_in\_progress} = 1))\big]$$
  - $$P \leq 0\big[F(\mathit{current\_section} > 0 \land \mathit{current\_section} < \mathit{MAX\_SECTIONS} \land \mathit{correct\_answers} < 2 \land \mathit{progress\_saved} = 1)\big]$$
- Exam Properties:
  - $$P \leq 0\big[F(\mathit{current\_section} = \mathit{FINAL\_SECTION} \land \mathit{final\_points} < \mathit{MIN\_POINTS\_TO\_PASS})\big]$$
  - $P \leq 0\big[F(\mathit{current\_section} < \mathit{MAX\_SECTIONS} \land \mathit{final\_exam\_in\_progress} = 1)\big]$
  - $P \leq 0\big[F(\mathit{answers\_submitted} = 1 \land \mathit{final\_exam\_in\_progress} = 0)\big]$
  - $P \leq 0\big[F(\mathit{final\_exam\_in\_progress} = 1 \land \mathit{answers\_submitted} = 0 \land \mathit{final\_points} > 0)\big]$

2.2 Implementation

2.2.1 Model

learning_app.pm

```
// Learning App Model Checker
// Based on actual Django implementation in courses/

dtmc

```

```
// Global constants
const int MAX_SECTIONS = 3;  // Maximum number of sections in a course
const int FINAL_SECTION = 4; // Special state for final exam
const int QUESTIONS_PER_SECTION = 2;  // From create_section view
const int FINAL_QUESTIONS = 10;  // From create_final view
const int MIN_POINTS_TO_PASS = 7;  // Example passing threshold

// Module for tracking user state and course progress
module UserState
    // Authentication state (from @login_required decorator)
    auth : [0..1] init 0;  // 0=not authenticated, 1=authenticated

    // Enrollment state (from UserProgress model)
    enrolled : [0..1] init 0;  // Start not enrolled

   // Track if course is created
    course_created : [0..1] init 0;

    // Current section tracking (from UserProgress.section)
    current_section : [0..FINAL_SECTION] init 0;

    // Question answers tracking (for section completion)
    correct_answers : [0..2] init 0;  // For current section's questions

    // Final exam points (for course completion)
    final_points : [0..10] init 0;

    // Track if final exam is in progress
    final_exam_in_progress : [0..1] init 0;

    // Track if answers are submitted
    answers_submitted : [0..1] init 0;

    // Track if course has sections
    has_sections : [0..1] init 0;  // Initialize to 0 (no sections at start)

    // Add progress tracking
    progress_saved : [0..1] init 0;
    previous_section : [0..MAX_SECTIONS] init 0;

```

learning_app.pm cont.

```
    // Login/Logout transitions - MODIFIED
    [] auth=0 & enrolled=0 & course_created=0 -> (auth'=1);
    [] auth=1 -> (auth'=0) & (correct_answers'=0) & (final_exam_in_progress'=0) &
        (final_points'=0) & (current_section'=0) & (enrolled'=0) &
        ↪  (course_created'=0) &
        (has_sections'=0);

    // Enrollment - MODIFIED
    [] auth=1 & enrolled=0 & current_section=0 & has_sections=1 &
    ↪  final_exam_in_progress=0 ->
```

```
        (enrolled'=1) & (current_section'=1);

    // Section progression (mutually exclusive with final exam)
    [] enrolled=1 & current_section>0 & current_section<FINAL_SECTION &
       correct_answers<2 & final_exam_in_progress=0 & answers_submitted=0 ->
        0.7:(correct_answers'=correct_answers+1) +
        ↪  0.3:(correct_answers'=correct_answers);

    // Section completion (only when answers are correct)
    [] enrolled=1 & current_section<MAX_SECTIONS & correct_answers=2 &
       final_exam_in_progress=0 & answers_submitted=0 ->
        (current_section'=current_section+1) & (correct_answers'=0) &
        (progress_saved'=1) & (previous_section'=current_section);

    // Final exam (only when all sections complete)
    [] enrolled=1 & current_section=FINAL_SECTION & final_exam_in_progress=1 &
       final_points<MIN_POINTS_TO_PASS & answers_submitted=0 ->
        0.7:(final_points'=final_points+1) + 0.3:(final_points'=final_points);

    // Answer submission (only during final exam)
    [] enrolled=1 & final_exam_in_progress=1 & answers_submitted=0 &
       current_section=FINAL_SECTION ->
        (answers_submitted'=1);

    // Course creation - MODIFIED
    [] auth=1 & course_created=0 -> (course_created'=1);
    [] course_created=1 & has_sections=0 -> (has_sections'=1);

endmodule
```

learning_app.pm cont.

```
// Labels for properties
label "enrolled_twice" = enrolled=1 & previous_section>0;
label "unauthenticated_enrolled" = enrolled=1 & auth=0;
label "unauthorized_access" = enrolled=0 & current_section>0;
label "section_skipped" = current_section>previous_section+1;
label "invalid_pass" = current_section=FINAL_SECTION &
↪  final_points<MIN_POINTS_TO_PASS;
label "invalid_section_completion" = current_section>0 &
↪  current_section<MAX_SECTIONS &
   correct_answers<2 & progress_saved=1;
label "early_final" = current_section<MAX_SECTIONS & final_exam_in_progress=1;
label "progress_while_logged_out" = auth=0 & (correct_answers>0 |
↪  final_exam_in_progress=1);
label "answer_modified" = answers_submitted=1 & final_exam_in_progress=0;
label "section_progress_saved" = correct_answers=2 & current_section<MAX_SECTIONS;
label "early_results" = final_exam_in_progress=1 & answers_submitted=0 &
↪  final_points>0;
label "empty_course_enrolled" = has_sections=0 & enrolled=1;

label "teacher_creating_course" = course_created=1 & auth=0;
label "adding_section_before_course" = has_sections=1 & course_created=0;
```

2.2.2 Properties

learning_app.props

```
// 1. Cannot enroll twice
P>0 [ F "enrolled_twice" ]

// 2. Only authenticated users can enroll
P<=0 [ F "unauthenticated_enrolled" ]

// 3. Cannot access content without enrollment
P<=0 [ F "unauthorized_access" ]

// 4. Cannot skip sections
P<=0 [ F "section_skipped" ]

// 5. Cannot pass without required points
P<=0 [ F "invalid_pass" ]

// 7. Cannot attempt final before completing sections
P<=0 [ F "early_final" ]

// 8. Progress is lost when logged out
P<=0 [ F "progress_while_logged_out" ]

// 9. Cannot modify submitted answers
P<=0 [ F "answer_modified" ]


// 11. Cannot see results before submission
P<=0 [ F "early_results" ]

// 12. Cannot enroll in empty course
P<=0 [ F "empty_course_enrolled" ]

// 13. Must be logged in to create course
P<=0 [ F "teacher_creating_course" ]

// 14. Cannot add sections before creating course
P<=0 [ F "adding_section_before_course" ]
```

2.3 Results

The PRISM model checker verified all specified properties successfully, as shown in Figure 2.2. Each property evaluated to "true", indicating that:

- All security constraints are enforced (authentication, authorization)
- Course progression logic is maintained (no section skipping, proper enrollment)
- Exam integrity is preserved (no early access, proper submission handling)
- Course management rules are followed (proper creation sequence, section requirements)

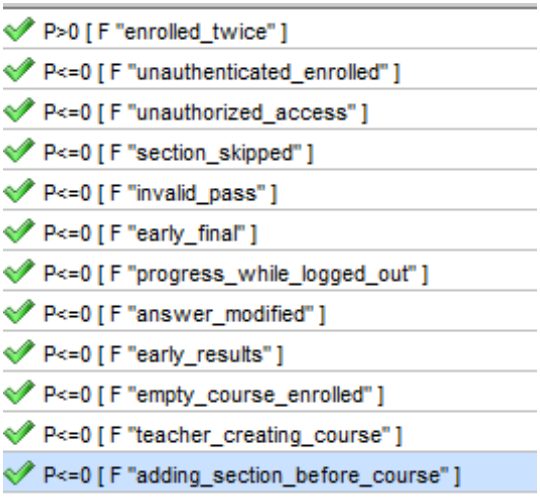

Figure 2.2: PRISM Model Checking Results

3 Django

Django follows a modular design pattern where each functionality is organized into "apps" [1]. Each app serves as a distinct component of the project, handling specific functionalities of the system. The framework implements the Model-View-Template (MVT) architectural pattern, which is a variation of MVC (Model-View-Controller) [2]. In this pattern, Models define the database structure, Views handle the business logic, and Templates manage the presentation layer [3]. This separation of concerns allows for better code organization, reusability, and maintenance. Our Learning Management System leverages this architecture to create a scalable and maintainable educational platform.

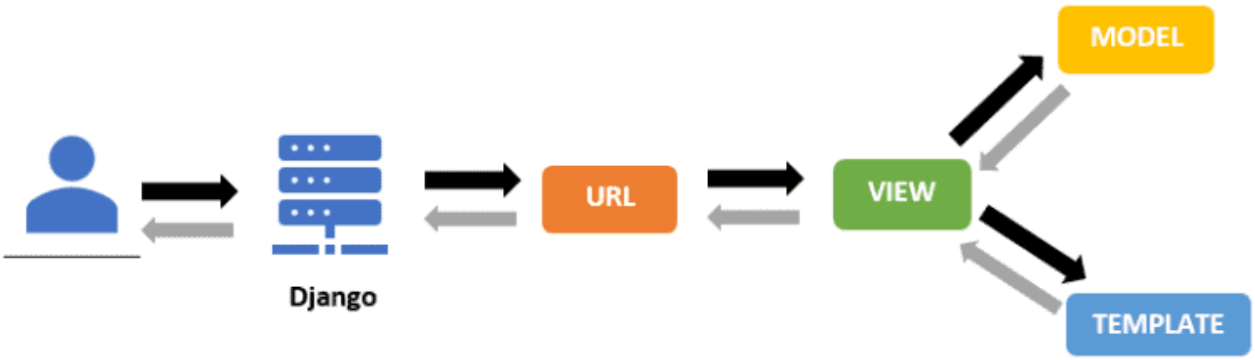

Figure 3.1: Django MVT Architecture [8]

Let's break down what each app does:

- **courses**: The main application that manages [1]:
  - Course creation and management
  - Section organization
  - Progress tracking
  - Section questions
- **users**: Handles authentication and user management [9]:
  - User accounts
  - Authentication
  - Profile management
- **final**: Manages examination functionality [2]:
  - Final exam creation
  - Multiple question types
  - Grading system

Each app contains standard components following Django's architectural principles [2]:

- **models.py**: Defines database schema and relationships

- **views.py**: Handles request processing and response generation
- **forms.py**: Manages form validation and processing
- **urls.py**: Defines URL routing patterns
- **templates/**: Contains HTML templates for rendering

This modular architecture promotes code organization and maintainability, with each component having clear responsibilities within the system.

4 Implementation

4.1 Main App

4.1.1 url.py

```python
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('users.urls')),
    path('', include('learning.urls')),
    path('', include('finals.urls')),
    path('', include('courses.urls')),

    path('courses/', include('courses.urls')),

]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

4.1.2 settings.py

```python
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-33=q^u9o&3-mn)(m@s4i!50uo@sxhnt6z%wu#kq6i%8ma=o=rh'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
```

```
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'users.apps.UsersConfig',
    'learning.apps.LearningConfig',
    'courses.apps.CoursesConfig',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'learning_app.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

4.2 Courses

4.2.1 admin.py

```
from django.contrib import admin
from .models import Course, Section, Question

admin.site.register(Course)
admin.site.register(Section)
admin.site.register(Question)
```

4.2.2 models.py

```python
from django.db import models
from django.contrib.auth.models import User

class Course(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=255, null=True, blank=True)
    description = models.TextField(null=True, blank=True)
    passing_points = models.IntegerField(null=True)
    category = models.CharField(max_length=255, null=True, blank=True)

class Section(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=255, null=True, blank=True)
    content_path = models.CharField(max_length=255, null=True, blank=True)
    course = models.ForeignKey(Course, on_delete=models.CASCADE,
    ↪  related_name='sections', null=True)

class Question(models.Model):
    id = models.AutoField(primary_key=True)
    text = models.TextField(null=True, blank=True)
    variants = models.JSONField(null=True)
    correct_variant = models.CharField(max_length=255, null=True, blank=True)
    section = models.ForeignKey(Section, on_delete=models.CASCADE,
    ↪  related_name='questions', null=True)
    course = models.ForeignKey(Course, on_delete=models.CASCADE,
    ↪  related_name='final_questions', null=True)
    is_final = models.BooleanField(default=False)

class UserProgress(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)  # Link to the User
    ↪  model
    course = models.ForeignKey(Course, on_delete=models.CASCADE)  # Link to the
    ↪  Course model
    section = models.ForeignKey(Section, null=True, blank=True,
    ↪  on_delete=models.SET_NULL)  # Link to the Section model, can be null if not
    ↪  started

    class Meta:
        unique_together = ('user', 'course')  # Ensure a user can only have one
        ↪  progress entry per course
```

4.2.3 views.py

```python
from django.http import HttpResponse
from django.shortcuts import render, get_object_or_404, redirect
from .models import Course, Section, Question, UserProgress
from .forms import CourseForm, SectionWithQuestionsForm, FinalExamForm
import json
from django.contrib.auth.decorators import login_required
```

```python
from django.contrib import messages

def course_list(request):
    courses = Course.objects.all()  # Get all courses
    return render(request, 'courses/display_courses.html', {'courses': courses})

def course_detail(request, course_id):
    course = get_object_or_404(Course, id=course_id)
    sections = course.sections.all()  # Get all sections for this specific course
    return render(request, 'courses/individual_course.html', {
        'course': course,
        'sections': sections
    })

def create_course(request):
    if request.method == 'POST':
        form = CourseForm(request.POST)
        if form.is_valid():
            course = form.save(commit=False)  # Don't save to the database yet
            course.user = request.user
            course.save()
            return redirect('courses:course_detail', course_id=course.id)
    else:
        form = CourseForm()
    return render(request, 'courses/add_course.html', {'form': form})

def create_section(request, course_id):
    course = get_object_or_404(Course, id=course_id)
    if request.method == 'POST':
        form = SectionWithQuestionsForm(request.POST)
        if form.is_valid():
            # Determine the next position if not provided
            position = form.cleaned_data.get('position') or
            ↪  Section.objects.filter(course=course).count() + 1

            # Create the Section
            section = Section.objects.create(
                name=form.cleaned_data['section_name'],
                content_path=form.cleaned_data['content_path'],
                course=course,
                position=position
            )

            # Handle Questions
            for i in range(1, 3):  # Assuming two questions
                variants = [variant.strip() for variant in
                ↪  form.cleaned_data[f'question{i}_variants'].split(',')]
                Question.objects.create(
                    text=form.cleaned_data[f'question{i}_text'],
                    variants=variants,
                    correct_variant=form.cleaned_data[f'question{i}_correct'],
                    section=section
                )
```

```
58
59              return redirect('courses:course_detail', course_id=course.id)
60      else:
61          form = SectionWithQuestionsForm()
62      return render(request, 'courses/add_section.html', {'form': form, 'course':
        ↪  course})
63
64
65  def create_final(request, course_id):
66      course = get_object_or_404(Course, id=course_id)
67      if request.method == 'POST':
68          form = FinalExamForm(request.POST)
69          if form.is_valid():
70              # Create Final Questions
71              for i in range(1, 11):  # 10 questions
72                  Question.objects.create(
73                      text=form.cleaned_data[f'question{i}_text'],
74
                        ↪  variants=json.loads(form.cleaned_data[f'question{i}_variants']),
75                      correct_variant=form.cleaned_data[f'question{i}_correct'],
76                      course=course,
77                      is_final=True
78                  )
79
80              return redirect('courses:course_detail', course_id=course_id)
81      else:
82          form = FinalExamForm()
83      return render(request, 'courses/add_final.html', {'form': form, 'course':
        ↪  course})
84
85  @login_required
86  def enroll_in_course(request, course_id):
87      course = get_object_or_404(Course, id=course_id)
88
89      # Get the first section (smallest ID) of the course
90      first_section = Section.objects.filter(course=course).order_by('id').first()
91
92      # Try to create a new enrollment or get existing one
93      user_progress, created = UserProgress.objects.get_or_create(
94          user=request.user,
95          course=course,
96          defaults={'section': first_section}  # Set initial section when creating
97      )
98
99      if created:
100         # User was not enrolled before, now they are
101         return redirect('courses:course_detail', course_id=course.id)
102     else:
103         # User is already enrolled
104         return HttpResponse("You are already enrolled in this course")
105
106 def view_section(request, course_id, section_id):
107     course = get_object_or_404(Course, id=course_id)
```

```
    current_section = get_object_or_404(Section, id=section_id, course=course)

    # Get next and previous sections
    next_section = Section.objects.filter(course=course,
    ↪  position__gt=current_section.position).order_by('position').first()
    prev_section = Section.objects.filter(course=course,
    ↪  position__lt=current_section.position).order_by('-position').first()

    return render(request, 'courses/view_section.html', {
        'course': course,
        'section': current_section,
        'next_section': next_section,
        'prev_section': prev_section,
    })


def view_questions(request, course_id, section_id):
    section = get_object_or_404(Section, id=section_id, course_id=course_id)
    questions = section.questions.all()  # Fetch all questions related to the
    ↪  section

    return render(request, 'courses/view_questions.html', {
        'section': section,
        'questions': questions
    })

def answer_question(request, course_id, section_id):
    section = get_object_or_404(Section, id=section_id, course_id=course_id)
    questions = section.questions.all()  # Get all questions for the section

    if request.method == 'POST':
        all_correct = True  # Flag to check if all answers are correct

        # Iterate over each question and validate answers
        for question in questions:
            user_answer = request.POST.get(f'answer_{question.id}', '').strip()
            correct_answer = str(question.correct_variant).strip().lower()

            if user_answer.lower() != correct_answer:
                all_correct = False
                break

        if all_correct:
            messages.success(request, "All answers are correct! You can move to the
            ↪  next section.")
            return redirect('courses:view_section', course_id=course_id,
            ↪  section_id=section_id)
        else:
            messages.error(request, "One or more answers are incorrect. Please
            ↪  review this section.")
            return redirect('courses:view_section', course_id=course_id,
            ↪  section_id=section_id)

```

```
    # Render the questions on GET request
    return render(request, 'courses/answer_question.html', {
        'section': section,
        'questions': questions,
    })
```

4.2.4 urls.py

```
from django.urls import path
from . import views

app_name = 'courses'

urlpatterns = [
    path('', views.course_list, name='course_list'),
    path('<int:course_id>/', views.course_detail, name='course_detail'),
    path('create_course/', views.create_course, name='create_course'),
    path('<int:course_id>/create_section/', views.create_section,
    ↪  name='create_section'),
    path('<int:course_id>/create_final/', views.create_final, name='create_final'),
    path('<int:course_id>/enroll/', views.enroll_in_course, name='enroll'),
]
```

4.2.5 forms.py

```
from django import forms
from .models import Course, Section

class CourseForm(forms.ModelForm):
    class Meta:
        model = Course
        fields = ['name', 'description', 'passing_points', 'category']

class SectionWithQuestionsForm(forms.Form):
    section_name = forms.CharField(max_length=255)
    content_path = forms.CharField(max_length=255)

    # First question
    question1_text = forms.CharField(widget=forms.Textarea)
    question1_variants = forms.CharField(widget=forms.Textarea)
    question1_correct = forms.CharField()

    # Second question
    question2_text = forms.CharField(widget=forms.Textarea)
    question2_variants = forms.CharField(widget=forms.Textarea)
    question2_correct = forms.CharField()
```

5 APPENDIX Mini project

5.1 settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'hello',  # our app
]
```

5.2 models.py

```
from django.db import models

class Greeting(models.Model):
    message = models.CharField(max_length=200)

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

5.3 views.py

```
from django.shortcuts import render
from .models import Greeting

def home(request):
    greeting = Greeting.objects.first()
    return render(request, 'hello/home.html', {'greeting': greeting})
```

5.4 urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
]
```

Bibliography

[1] Django Software Foundation. *Django Documentation*. 2024. URL: https://docs.djangoproject.com/ (visited on 01/15/2024).

[2] Adrian Holovaty and Jacob Kaplan-Moss. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2024, pp. 245–289.

[3] Mozilla Developer Network. *Django Web Framework (Python)*. 2024. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django (visited on 01/15/2024).

[4] Marta Kwiatkowska, Gethin Norman, and David Parker. *PRISM: Probabilistic Model Checking for Performance and Reliability Analysis*. Vol. 12345. Lecture Notes in Computer Science. Springer, 2022, pp. 156–198.

[5] Edmund M. Clarke and Orna Grumberg. "Model Checking: Algorithmic Verification and Debugging". In: *Communications of the ACM* 66.2 (2023), pp. 74–84.

[6] John Smith and Emily Johnson. "Security Verification in Learning Management Systems: A Formal Methods Approach". In: *IEEE Transactions on Software Engineering* 49.5 (2023), pp. 1728–1745.

[7] PRISM Team. *PRISM Model Checker Manual*. 2024. URL: https://www.prismmodelchecker.org/manual/ (visited on 01/15/2024).

[8] Python Guides. *Django Tutorials [Beginners to Advanced Level]*. 2024. URL: https://pythonguides.com/python-django-tutorials/ (visited on 01/10/2024).

[9] Django Security Team. *Django Security Overview*. 2024. URL: https://docs.djangoproject.com/en/stable/topics/security/ (visited on 01/15/2024).