High-Level Design
Django-Based Online Judge Platform

Srivardhan Ginjala

June 19, 2025

Contents

1 Purpose

The Online Judge platform enables users to practice programming problems by submitting code solutions and receiving automated feedback. Built with Django and SQLite, this system addresss core challenges like secure code execution, submission scalability, and educational value through AI-enhanced features.

2 Technology Stack

Component	**Technology**
Backend Framework	Django + Django REST Framework
Database	SQLite (Django default)
Frontend	Django Templates + JavaScript
Code Execution	Docker Containers
Task Queue	Celery + Redis
AI Integration	OpenAI API
Deployment	AWS EC2

3 System Architecture

The platform follows Django's MVT (Model-View-Template) architecture with asynchronous task processing:

```
User Interface (Django Templates)
                ↓
   Django Views & URL Routing
                ↓
    Models & SQLite Database
                ↓
 Celery + Docker Execution Engine
```

4 Database Schema (Task 1)

Following the three-table requirement from the step breakdown:

```
# models.py - Core three tables as specified
from django.db import models
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    full_name = models.CharField(max_length=200)
    created_at = models.DateTimeField(auto_now_add=True)

class Problem(models.Model):
    DIFFICULTY_CHOICES = [
        ('E', 'Easy'), ('M', 'Medium'), ('H', 'Hard')
    ]
    statement = models.TextField()  # CharField as per requirement
    name = models.CharField(max_length=255)  # CharField as per requirement
    code = models.CharField(max_length=50)  # Problem code identifier
    difficulty = models.CharField(max_length=1, choices=DIFFICULTY_CHOICES)
    time_limit = models.IntegerField(default=1)
    memory_limit = models.IntegerField(default=256)

class TestCase(models.Model):
    input = models.TextField()  # CharField as per requirement
    output = models.TextField()  # CharField as per requirement
    problem = models.ForeignKey(Problem, on_delete=models.CASCADE)

class Submission(models.Model):
    problem = models.ForeignKey(Problem, on_delete=models.CASCADE)
    verdict = models.CharField(max_length=50)  # CharField as per requirement
    submitted_at = models.DateTimeField(auto_now_add=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    code = models.TextField()
    language = models.CharField(max_length=20)
```

5 Web Server Design (Task 2)

Following the maximum 3 UI screens requirement, implementing exactly 2 main screens:

5.1 Screen 1: Problem List & Individual Problem

URL: `/problems/`, `/problems/<id>/`

- **Template:** Simple list of problem names linking to individual pages
- **View:** GET request fetching all problems from Problem table
- **Individual Problem:** Shows problem statement with submission box
- **Submission:** POST request handling code evaluation workflow

5.2 Screen 2: Leaderboard

URL: `/leaderboard/`

- **Template:** List showing verdict of last 10 submissions
- **View:** GET request fetching solutions with verdicts from submission table

6 Code Evaluation System (Task 3)

6.1 Docker Integration

Following the step breakdown requirements for secure execution:

```
# Celery task for asynchronous processing
import docker
from celery import shared_task

@shared_task
def execute_submission(submission_id):
    submission = Submission.objects.get(id=submission_id)
    client = docker.from_env()

    # Docker container setup as per requirements
    container_config = {
        'image': 'gcc',  # Using GCC container as specified
        'mem_limit': f'{submission.problem.memory_limit}m',
        'network_mode': 'none',  # Network isolation
        'read_only': True
    }

    try:
        # Get test cases for the problem
        test_cases = TestCase.objects.filter(problem=submission.problem)
        results = []

        for test_case in test_cases:
            result = client.containers.run(
                **container_config,
                command=['timeout', '5s', 'python3', '/app/solution.py'],
                stdin=test_case.input,
                capture_output=True,
                remove=True
            )

            if result.stdout.strip() == test_case.output.strip():
                results.append('AC')
            else:
                results.append('WA')

        # Save verdict in database
        verdict = 'Accepted' if all(r == 'AC' for r in results) else 'Wrong Answer'
        submission.verdict = verdict
        submission.save()

        return verdict
    except Exception as e:
        submission.verdict = 'Runtime Error'
        submission.save()
        return 'Runtime Error'
```

7 Unique Features (USPs)

7.1 Tab Switching Detection

Enhances contest integrity by monitoring user behavior:

- JavaScript Visibility API integration for real-time detection
- Server-side logging of suspicious activity
- Configurable penalty system for violations
- Admin dashboard for monitoring contest integrity

7.2 AI-Powered Debugging Assistant

Provides intelligent feedback to improve learning outcomes:

- OpenAI API integration for error analysis
- Natural language explanations of compilation errors
- Contextual hints based on common error patterns
- Performance optimization suggestions
- Personalized feedback based on submission history

8 Security Measures

Addressing the three core challenges from requirements:

8.1 Thundering Herd Solution

- Celery task queue for asynchronous submission processing
- Redis as message broker to handle thousands of concurrent submissions
- Rate limiting on submission endpoints

8.2 Malicious Code Protection

- Docker containerization with strict resource limits
- Network isolation (network_mode=’none’)
- Memory and CPU constraints
- Process timeout enforcement

8.3 Unauthorized Access Prevention

- Django's built-in authentication system
- CSRF protection for all forms
- Input validation and sanitization
- Role-based access control

9 Implementation Timeline (2 Weeks) (Hopefully)

9.1 Week 1: Core Foundation (Tasks 0-1)

Day	Task	Deliverable
1-2	Django skeleton setup	Project structure, basic models
3-4	Database design	Three tables implemented
5-6	User authentication	Login/registration system
7	Problem management	CRUD operations for problems

9.2 Week 2: Advanced Features (Tasks 2-3)

Day	Task	Deliverable
8-9	UI screens implementation	Problem list & leaderboard
10-11	Docker execution system	Secure code evaluation
12-13	USP features	Tab detection & AI debugging
14	Testing & deployment	Production-ready system

10 URL Structure

Django URL patterns following the functional requirements:

URL Pattern	View Function	Purpose
/	home	Landing page
/register/	register_user	User registration
/login/	login_user	Authentication
/problems/	problem_list	Browse problems (Screen 1)
/problems/¡int:id¿/	problem_detail	Individual problem view
/submit/	submit_solution	Code submission handler
/leaderboard/	leaderboard	Submission results (Screen 2)

11 Conclusion

This Django-based Online Judge platform provides a practical solution for competitive programming practice. By leveraging Django's built-in features and focusing on the three core tables and two main screens as specified, the system addresses all fundamental requirements while incorporating unique differentiators.

The design successfully tackles the three major challenges through proven solutions: message queues for scalability, Docker containers for security, and Django's authentication for access control. The addition of tab switching detection and AI-powered debugging assistance positions this platform competitively in the online judge marketplace while maintaining implementation feasibility for beginner developers like myself.