TECHNICAL ASSESSMENT

Software Development Intern Position

Position:	Software Development Intern
Duration:	3-4 hours
Focus:	Problem Solving & Code Quality
Tech Stack:	Python, Django, JavaScript, HTML, CSS

Important Instructions:

- Read the entire document before starting
- Complete the assignment within the given time frame
- Focus on code quality and problem-solving approach over feature completeness
- Document your design decisions in the README
- Commit your code to a GitHub repository with clear commit messages

Overview

You will build a **Smart Task Analyzer** - a mini-application that intelligently scores and prioritizes tasks based on multiple factors. This assignment tests your problem-solving ability, algorithmic thinking, and clean code practices.

What We're Evaluating:

- **Algorithm Design (40%):** Can you translate business requirements into logical code?
- **Code Quality (30%):** Clean, maintainable, well-structured code
- **Critical Thinking (20%):** Handling edge cases and ambiguous requirements
- **Frontend Skills (10%):** Functional user interface with good UX

Problem Statement

Create a task management system that intelligently scores and sorts tasks based on multiple factors. Your system should help users identify which tasks they should work on first.

Part 1: Backend Development (Python/Django)

Estimated Time: 2 hours

Task Model Structure

Each task should have the following properties:

```
{    "title": "Fix login bug",     "due_date": "2025-11-30",     "estimated_hours": 3,
"importance": 8,  // 1-10 scale     "dependencies": []  // list of task IDs }
```

The Core Challenge: Design a Priority Algorithm

Create a scoring function that calculates task priority by considering multiple factors. Your algorithm should intelligently weigh:

- **Urgency:** How soon is the task due? Past-due tasks should be weighted appropriately.
- **Importance:** User-provided rating (1-10 scale)
- **Effort:** Lower effort tasks might be "quick wins" worth prioritizing
- **Dependencies:** Tasks that block other tasks should rank higher

Required API Endpoints

POST /api/tasks/analyze/	Accept a list of tasks and return them sorted by priority score. Each task should include its calculated score.
GET /api/tasks/suggest/	Return the top 3 tasks the user should work on today, with explanations for why each was chosen.

Critical Considerations

- How do you handle tasks with due dates in the past?
- What if a task has missing or invalid data?
- How do you detect circular dependencies?
- Should your algorithm be configurable? (e.g., user preferences for weighting)
- How do you balance competing priorities (urgent vs important)?

Part 2: Frontend Development (HTML/CSS/JavaScript)

Estimated Time: 1.5 hours

Required Interface Components

1. Input Section

- Form to add individual tasks with all required fields
- Option to paste JSON array of tasks for bulk input
- Button to "Analyze Tasks" that calls your API

2. Output Section

- Display sorted tasks with their calculated priority scores
- Visual priority indicators (e.g., color coding: High/Medium/Low)
- Show a brief explanation of why each task received its score
- Display the task details (title, due date, effort, importance)

3. Critical Thinking Element (Important!)

Add a toggle or dropdown to switch between different sorting strategies:

- **"Fastest Wins":** Prioritize low-effort tasks
- **"High Impact":** Prioritize importance over everything
- **"Deadline Driven":** Prioritize based on due date
- **"Smart Balance":** Your custom algorithm that balances all factors

Frontend Requirements

- Functional interface that successfully communicates with your API
- Clean, readable code with proper event handling
- Basic form validation before API calls
- Error handling and user feedback (loading states, error messages)
- Responsive design (should work on different screen sizes)

Bonus Challenges (Optional)

If you complete the core assignment early, consider implementing one or more of these features:

- **Dependency Graph Visualization:** Detect and visually flag circular dependencies (30-45 min)
- **Date Intelligence:** Consider weekends/holidays when calculating urgency (30 min)
- **Eisenhower Matrix View:** Display tasks on a 2D grid (Urgent vs Important) (45 min)
- **Learning System:** Allow users to mark if suggested tasks were helpful and adjust algorithm (1 hour)
- **Unit Tests:** Write comprehensive tests for your scoring algorithm (45 min)

Note: Bonus challenges help us understand your technical depth, but quality on the core assignment is more important than attempting bonuses.

Evaluation Criteria

Algorithm Quality (40%)	• Does your scoring logic make sense? • How do you handle edge cases? • Is the algorithm flexible/configurable? • Clear documentation of approach
Code Quality (30%)	• Clean, readable code structure • Proper error handling • Good variable/function naming • Logical organization
Critical Thinking (20%)	• Handling ambiguous requirements • Trade-off decisions • Edge case awareness • Creative problem-solving
Frontend (10%)	• Functional interface • Good user experience • Proper API integration • Basic styling and responsiveness

Submission Requirements

1. GitHub Repository

Your repository should contain:

- Complete Django backend code
- Frontend files (HTML, CSS, JavaScript)
- requirements.txt or Pipfile for Python dependencies
- README.md with setup instructions
- At least 3 unit tests for your scoring algorithm
- Clean commit history with meaningful commit messages

2. README.md Content

Your README should include:

- **Setup Instructions:** How to install dependencies and run the application
- **Algorithm Explanation:** How your priority scoring works (300-500 words)
- **Design Decisions:** Trade-offs you made and why
- **Time Breakdown:** Approximate time spent on each section
- **Bonus Challenges:** Which ones you attempted (if any)
- **Future Improvements:** What you'd do with more time

3. Technical Requirements

- Python 3.8+ and Django 4.0+
- Database: SQLite is fine (default Django setup)
- No deployment required - just needs to run locally
- No authentication system needed for this assignment

Submission Deadline: Please submit your GitHub repository link within 48 hours of receiving this assignment.

Suggested Project Structure

```
task-analyzer/ ├── backend/ │   ├── manage.py │   ├── task_analyzer/ │   │   ├──
__init__.py │   │   ├── settings.py │   │   ├── urls.py │   │   └── wsgi.py │   ├── tasks/ │
│   ├── __init__.py │   │   ├── models.py │   │   ├── views.py │   │   ├── serializers.py │
│   ├── scoring.py │   │   ├── urls.py │   │   └── tests.py │   └── requirements.txt ├──
frontend/ │   ├── index.html │   ├── styles.css │   └── script.js └── README.md
```

Tips for Success

- **Start with the algorithm:** Spend time thinking through your scoring logic before coding
- **Test edge cases:** Make sure your code handles missing data, invalid inputs, and unusual scenarios
- **Keep it simple first:** Get a working solution, then refine and improve
- **Document as you go:** Write comments explaining your reasoning, especially for the algorithm
- **Commit frequently:** Make meaningful commits that show your development process
- **Don't over-engineer:** Focus on solving the problem well rather than adding unnecessary complexity
- **Ask questions:** If requirements are unclear, document your assumptions in the README

Final Notes

This assignment is designed to see how you approach problems, structure your code, and think through trade-offs. There's no single "correct" answer - we're interested in your thought process and execution.
Focus on demonstrating clean code practices, solid problem-solving skills, and good communication through your documentation. Good luck, and we look forward to reviewing your submission!