Settlement Schedule Reflow - Technical Test

Overview

You are tasked with building a **settlement schedule reflow system** for a financial operations platform. When disruptions occur (delayed trades, counterparty failures, regulatory holds), the system must intelligently reschedule settlement tasks while respecting various constraints and dependencies.

6-hour timebox: If you run out of time, no worries! Just add `@upgrade` comments on incomplete parts explaining what you'd build next. We want to see how you think and prioritize.

AI Tools: Feel free to use an AI assistant to help with test data generation, debugging, algorithm exploration, or edge case analysis. If you use AI for key decisions (e.g., analyzing constraints, designing the reflow logic), save those prompts in a markdown file — we'd love to see your problem-solving process!

Example of settlement schedule view: Think of a Gantt-style timeline where each row is a settlement desk/channel and each block is a settlement task with its processing window.

At a Glance

What you're building: A settlement scheduler that reschedules settlement tasks when disruptions occur

Must handle:

- Dependencies (Fund transfer must clear before disbursement)
- Channel conflicts (no overlapping tasks on the same settlement channel)
- Market hours (processing pauses outside operating windows)
- Regulatory blackout windows (blocked time)

Required deliverables:

1. Working algorithm (TypeScript)
2. Sample data (3+ scenarios)
3. Loom demo (5–10 min)

4. GitHub repo with README

Bonus (optional):

- Automated test suite
- More sample data
- DAG implementation
- Optimization metrics
- AI prompts documentation

The Problem

Capital33 operates a financial operations platform that processes trade settlements, fund transfers, and compliance checks across multiple settlement channels. Settlement tasks are scheduled across these channels, but real-world disruptions constantly require rescheduling:

- A counterparty's fund transfer arrives late, delaying everything downstream
- A settlement channel goes offline for system maintenance (planned or unplanned)
- Settlement tasks have dependencies (a disbursement can't begin until the corresponding margin check clears)
- Different settlement channels operate during different market hours (e.g., domestic vs. international wire windows)
- Some tasks are regulatory compliance holds that cannot be moved

Your job is to create a **reflow algorithm**. A reflow algorithm will look at all the settlement tasks from a time interval perspective and will update those times in order to produce a valid schedule that respects all constraints.

Core Requirements

1. The Reflow Algorithm

Takes:

- Settlement tasks with start/end dates
- Settlement channels with operating hours and maintenance windows
- Dependencies between tasks

Produces:

- Valid schedule with updated dates
- List of changes (what moved, by how much)

- Explanation (why it changed)

Key Rules:

- Settlement tasks take their full `durationMinutes` to complete
- Processing pauses outside operating hours, resumes in the next operating window
- No processing during maintenance/blackout windows
- All upstream dependencies must complete before a downstream task starts

Example: A settlement task needs 120 min, starts Mon 3PM, channel operating hours end at 4PM (Mon–Fri 8AM–4PM EST) → Processes 60 min Mon (3PM–4PM) → Pauses → Resumes Tue 8AM → Completes 9AM

2. Constraints (Hard Requirements)

Your algorithm MUST respect:

Settlement Channel:

- Only one task at a time per channel (no overlaps)
- Respect operating hours (market windows)
- No processing during maintenance/blackout windows

Dependencies:

- Multiple upstream dependencies allowed (all must complete first)
- Can form chains (Margin Check → Fund Transfer → Disbursement)

Time:

- Processing only during operating hours
- Maintenance/blackout windows cannot be changed
- Regulatory hold tasks cannot be rescheduled

3. Goal

Produce a valid schedule where:

- No settlement channel conflicts
- All dependencies satisfied
- All processing within operating hours
- Maintenance/blackout windows respected

Data Structures

All documents follow this structure:

```typescript
{
  docId: string;          // Unique identifier
  docType: string;        // Document type
  data: {
    // Document-specific fields
  }
}
```

Settlement Task

```typescript
{
  docId: string;
  docType: "settlementTask";
  data: {
    taskReference: string;          // e.g., "STL-20240115-001"
    tradeOrderId: string;           // Parent trade order
    settlementChannelId: string;    // Which channel processes
this

    // Timing
    startDate: string;              // ISO 8601
    endDate: string;                // ISO 8601
    durationMinutes: number;        // Total processing time
required

    // Constraints
    isRegulatoryHold: boolean;      // Cannot be rescheduled if
true

    // Dependencies (can have multiple upstream tasks)
    dependsOnTaskIds: string[];     // All must complete before
this starts
```

```
    // Task classification
    taskType: "marginCheck" | "fundTransfer" | "disbursement" |
              "complianceScreen" | "reconciliation" |
"regulatoryHold";
  }
}
```

Settlement Channel

```typescript
{
  docId: string;
  docType: "settlementChannel";
  data: {
    name: string;                   // e.g., "Domestic Wire
Desk", "FX Settlement"

    // Operating hours (market windows)
    operatingHours: Array<{
      dayOfWeek: number;           // 0–6, Sunday = 0
      startHour: number;           // 0–23
      endHour: number;             // 0–23
    }>;

    // Maintenance / blackout windows (blocked time periods)
    blackoutWindows: Array<{
      startDate: string;           // ISO 8601
      endDate: string;             // ISO 8601
      reason?: string;             // e.g., "Fed settlement
system maintenance"
    }>;
  }
}
```

Trade Order

```
TypeScript
{
  docId: string;
  docType: "tradeOrder";
  data: {
    tradeOrderNumber: string;       // e.g., "TRD-20240115-042"
    instrumentId: string;           // Security or asset
identifier
    quantity: number;
    settlementDate: string;         // Target settlement date
(T+1, T+2, etc.)
  }
}
```

What You Need to Deliver

1. Working Algorithm (Required)

Build the reflow scheduler that handles:

- Dependencies (multiple upstream tasks)
- Settlement channel conflicts
- Operating hour boundaries (pause/resume)
- Blackout windows

Suggested structure:

```
None
src/
├── reflow/
│   ├── reflow.service.ts          # Main algorithm
│   ├── constraint-checker.ts      # Validation logic
│   └── types.ts                   # TypeScript types
└── utils/
    └── date-utils.ts              # Date helpers (Luxon)
```

2. Sample Data (Required)

Create sample data for at least **2 scenarios**:

1. **Delay Cascade**: A counterparty's fund transfer is delayed → affects all downstream settlement tasks (margin verification, disbursement, reconciliation)
2. **Market Hours or Blackout**: A settlement task spans operating windows OR conflicts with a system blackout (e.g., Fedwire maintenance)

Can be hardcoded or in JSON files. Just needs to demonstrate your algorithm works.

3. Documentation (Required)

- **Code comments** explaining key logic
- **README.md** with:
    - How to run your code
    - High-level algorithm approach
    - Setup instructions

4. Demo Video (Required)

5–10-minute Loom video showing:

- Your code running with sample data
- Output for both scenarios (what changed, why)
- Walkthrough of your algorithm approach

5. Public Repository (Required)

GitHub/GitLab repo with:

- Working code (runnable)
- Sample data
- README
- Clean commit history

Bonus Points (Optional — Show Off Your Skills!)

If you finish early or want to showcase advanced capabilities:

Bonus Features:

- **DAG (Directed Acyclic Graph)** implementation for dependency management

    - Topological sort for optimal ordering
    - Cycle detection (circular dependencies)

- **Automated test suite**: Write formal tests (Jest/Vitest) covering:
  - 3+ scenarios with proper assertions
  - Edge cases (circular dependencies, impossible schedules)
  - Constraint validation tests
  - This is highly valued but not required due to time constraints
- **Setup/pre-processing time handling**: Settlement tasks may have `prepTimeMinutes` before actual processing starts (e.g., document verification before wire initiation)
  - Add to Settlement Task: `prepTimeMinutes?: number`
  - Prep time counts as working time within operating hours
- **Additional scenarios** (3+ beyond the required 2):
  - Complex multi-constraint scenarios (dependencies + blackouts + market hours)
  - Channel conflict resolution with multiple competing settlement tasks
  - Impossible schedule detection (explain why constraints cannot be satisfied)
- **Optimization metrics**:
  - Calculate total settlement delay introduced: `Σ (new_end_date - original_end_date)`
  - Minimize number of settlement tasks affected
  - Track utilization metrics: `(total processing minutes) / (total available operating minutes)`
  - Settlement channel idle time analysis
  - SLA breach detection: flag tasks that would miss their trade order's target settlement date
- **Document your AI prompts**: Save prompts you used in markdown files (shows collaboration approach)
  - Example: `prompts/algorithm-design.md`, `prompts/market-hours-calculation.md`
- **Enhanced documentation**:
  - Trade-offs you considered (why this approach over alternatives?)
  - Known limitations (what doesn't work yet?)
  - Tag future improvements with `@upgrade` comments in code
- **Clean git history**:
  - Meaningful commit messages
  - Logical progression of features
  - Separate commits for major features

Evaluation Criteria

Your submission will be evaluated on:

Algorithm Correctness (80%)

- **Constraint Satisfaction (40%)**: Respects ALL hard constraints
  - Settlement channel conflicts (no overlaps)
  - Dependencies (all upstream tasks complete before downstream)
  - Operating hours (processing only during market windows)
  - Blackout windows (no processing during maintenance/regulatory blackouts)
- **Valid Schedule Generation (30%)**: Produces schedules that actually work
  - Correct start/end date calculations
  - Proper operating hour boundary handling
  - Blackout windows properly avoided
  - **Demonstrated working in Loom video**
- **Handles Required Scenarios (10%)**: At least 2 scenarios demonstrated

Problem Solving (20%)

- **Code Quality (10%)**:
  - Clean, readable TypeScript with proper types
  - Good separation of concerns
  - Proper error handling
- **Algorithm Design (5%)**:
  - Logical approach to solving the scheduling problem
  - Efficient constraint checking
- **Communication (5%)**:
  - Clear explanation in Loom demo
  - Understandable code comments
  - Good documentation

Bonus Points (Extra Credit)

- Automated test suite (highly valued)
- DAG implementation with cycle detection
- Prep time handling
- Additional scenarios (3+)
- Optimization features (delay metrics, utilization, SLA breach detection)
- AI prompt documentation
- Enhanced documentation (trade-offs, complexity analysis)
- Git commit quality

Hints & Tips

Start simple, add complexity gradually:

1. Get basic reflow working (ignore operating hours)
2. Add dependencies
3. Add settlement channel conflicts
4. Add operating hour logic (hardest part!)
5. Add blackout windows

Operating hour logic is the trickiest part:

- Use **Luxon** for date manipulation (strongly recommended)
- Create helper: `calculateEndDateWithOperatingHours(startDate, duration, operatingHours)`
- Remember: processing pauses outside operating hours, resumes in next window

Suggested constraint checking order: Dependencies → Channel Conflicts → Operating Hours → Blackout Windows

Key considerations:

- All dates in UTC
- Track processing minutes, not elapsed time
- Blackout windows = blocked time on settlement channels

Common Questions

Q: What if there's no valid solution?
A: Throw an error explaining which constraints can't be satisfied (e.g., "Task STL-001 cannot meet T+2 settlement deadline due to upstream dependency chain delay").

Q: Can I use libraries like Luxon?
A: Yes! Luxon is highly recommended for date handling.

Q: How should settlement tasks span across operating windows?
A: They pause during non-operating hours and resume in the next window.

- Example: 120-min task starts Mon 3PM, channel closes 4PM (Mon–Fri 8AM–4PM)
- Processes 60 min Monday → pauses → resumes Tue 8AM → completes 9AM

Q: Can settlement tasks have multiple upstream dependencies?
A: Yes! ALL upstream tasks must complete before the downstream task starts.

Q: What if I don't finish in 6 hours?
A: Submit what you have! A working core solution is better than an incomplete one with bonuses.

Submission

Provide:

1. **Public GitHub/GitLab repository** with:
- Working code (TypeScript)
- Sample data for 3+ scenarios
- README with setup and approach
- Clean commit history
2. **Loom video** (max 10 minutes) demonstrating your solution
3. **(Bonus)** Automated test suite
4. **(Bonus)** Markdown files with AI prompts you used

Example Usage (Not Required, Just for Context)

Here's what using your reflow service might look like:

```TypeScript
const reflowService = new ReflowService();

const result = reflowService.reflow({
  settlementTasks: [...],       // Current settlement tasks
  settlementChannels: [...],    // Channels with operating hours
and blackouts
  tradeOrders: [...]            // Trade orders for context
(target settlement dates)
});

console.log(result.updatedTasks);    // New schedule
console.log(result.changes);         // What changed
console.log(result.explanation);     // Why it changed
```

Good luck! We're excited to see your solution. 🚀

Capital33 Team