ShiftPilot Scheduling Rules Analysis Report

Document: Scheduling Rules Implementation Analysis
Date: January 17, 2025
Status: Gap Analysis Complete
System: ShiftPilot v2.0

Executive Summary

This report analyzes the scheduling rules and guidelines document against the existing ShiftPilot system implementation. The analysis reveals that ShiftPilot already has a sophisticated foundation for handling most scheduling requirements, with specific gaps identified in institutional rule enforcement and specialized vacation handling.

Key Findings: - [COMPLETE] **Core scheduling system**: Fully implemented with constraint engine, fairness algorithms, and vacation optimization - [COMPLETE] **Priority system**: P1/P2/P3 vacation preferences with fairness-based tie-breaking - [COMPLETE] **Institutional blackouts**: Christmas/New Year and summer blackout periods implemented - [GAPS IDENTIFIED] **Gaps identified**: 7 specific areas requiring implementation or enhancement

---

Original Scheduling Rules

1. Holiday Week Allocation

- **Rule**: 11 holiday weeks available per partner
- **Priority System**: P1 (high), P2 (intermediate), P3 (low), P4 (remaining)
- **Processing**: Initial submission, subsequent rounds by priority

2. RSNA Conference Rules

- **Academic requests**: Must rank as P1 or P2
- **Personal vacation**: Should be avoided, requires discussion if necessary

3. March Break Rules

- **Constraint**: First OR second week (cannot split)
- **Priority**: Must rank as P1 or P2 (P1 recommended)
- **Restriction**: Partners without school-age children should avoid if possible
- **PT/Admin**: May not be available during March break

4. Summer Holidays

- **Limit**: 3 weeks at P1 maximum

- **Definition**: Start/end weeks defined by scheduler

5. **Part-time/Admin Days**
   - **Priority**: P1-P3 vacation takes precedence
   - **Accommodation**: Best efforts, no guarantee before schedule release
6. **Protected Weekends**
   - **Limit**: Maximum 5 weekends per partner
   - **Guidance**: Keep to minimum
7. **Christmas/New Year Scheduling**
   - **Formula**: 3 working days off before Christmas, before New Year's, after Christmas
   - **Restriction**: No holiday weeks during Christmas Holiday session
   - **Exceptions**: Additional time requires group consideration and approval
8. **Emergency Coverage Procedures**
   - **Notification**: Email scheduler MNC with cc to appropriate contacts
   - **Coverage**: Proactive contact of colleagues for coverage
   - **Escalation**: Text/call if no rapid response

Current ShiftPilot Implementation Analysis

[COMPLETE] Fully Implemented Features

Priority System & Vacation Management

```
// From: lib/scheduling/core/schedule-generator.ts
const sortedPreferences = pendingPreferences.sort((a, b) => {
  // First priority: rank (P1 > P2 > P3)
  if (a.rank !== b.rank) {
    return a.rank - b.rank
  }

  // Second priority: fairness score (higher fairness debt gets preference)
  const aScore = userFairnessScores.get(a.userId) || 0
  const bScore = userFairnessScores.get(b.userId) || 0
  return bScore - aScore // Higher score first
})
```

Features: - [COMPLETE] P1/P2/P3 priority system with fairness-based tie-breaking - [COMPLETE] 11 holiday weeks tracking per user - [COMPLETE] Vacation preference optimization algorithm - [COMPLETE] Approval/rejection workflow (PENDING/APPROVED/REJECTED) - [COMPLETE] Fairness debt tracking for long-term equity

Constraint Engine

```
// From: lib/scheduling/constraints/constraint-engine.ts
class VacationPreferenceConstraintValidator implements ConstraintValidator {
  validate(candidate: AssignmentCandidate, context: GenerationContext): ConstraintResult {
    // Penalty based on preference rank (P1 = highest penalty)
    const penaltyByRank = { 1: -50, 2: -30, 3: -10 }
    const penalty = penaltyByRank[conflictingPref!.rank] || 0
    // ... validation logic
  }
}
```

Hard Constraints: - [COMPLETE] HC-001: Shift Coverage Requirement - [COMPLETE] HC-002: Subspecialty Eligibility - [COMPLETE] HC-003: Named Allowlist (Procedure Shifts) - [COMPLETE] HC-004: Single Assignment Per Day - [COMPLETE] HC-005: Vacation Block Compliance - [COMPLETE] HC-006: FTE Compliance

Soft Constraints: - [COMPLETE] SC-001: Workload Distribution Fairness - [COMPLETE] SC-002: Fairness Ledger - [COMPLETE] SC-003: Vacation Preference Satisfaction - [COMPLETE] SC-004: Desirability Balance

Institutional Vacation Rules

```
// From: lib/scheduling/constraints/constraint-engine.ts
private getInstitutionalBlackouts(year: number): Array<{ start: Date; end: Date; reason: st
  return [
    // Christmas/New Year blackout
    {
      start: new Date(year, 11, 20), // Dec 20
      end: new Date(year + 1, 0, 5), // Jan 5
      reason: 'Christmas/New Year blackout'
    },
    // Summer blackout periods
    {
      start: new Date(year, 6, 1),    // July 1
      end: new Date(year, 6, 14),     // July 14
      reason: 'Summer blackout period 1'
    }
  ]
}
```

Features: - [COMPLETE] Christmas/New Year blackout periods - [COMPLETE] Summer blackout periods (configurable) - [COMPLETE] Heavy penalty for vacation requests during blackouts - [COMPLETE] Prioritization of non-vacation radiologists during blackouts

Data Model & Infrastructure

```
-- From: prisma/schema.prisma
model VacationPreference {
  id              String   @id @default(cuid())
  userId          String   @map("user_id")
  year            Int
  month           Int      // 1-12
  weekNumber      Int      // Week number in month
  rank            Int      // 1=first choice, 2=second, 3=third
  weekStartDate   DateTime // Monday of the week
  weekEndDate     DateTime // Sunday of the week
  status          PreferenceStatus @default(PENDING)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt

  @@unique([userId, year, month, rank])
  @@map("vacation_preferences")
}
```

Features: - [COMPLETE] Complete vacation preferences schema - [COMPLETE] Notification system framework - [COMPLETE] Manual vacation selector UI - [COMPLETE] Auto-generation of preferences - [COMPLETE] Preference management interface

---

[GAPS IDENTIFIED] Identified Gaps & Implementation Requirements

1. RSNA Conference Handling

Current State: No distinction between academic vs personal vacation requests

Required Implementation:

```
// Extend VacationPreference model
interface VacationPreferenceData {
  // ... existing fields
  reason?: 'ACADEMIC' | 'PERSONAL' | 'FAMILY' | 'CONFERENCE'
  conferenceType?: 'RSNA' | 'OTHER'
  requiresDiscussion?: boolean
}

// Add validation logic
```

```
class RSNAConstraintValidator implements ConstraintValidator {
  validate(candidate: AssignmentCandidate, context: GenerationContext): ConstraintResult {
    // Ensure academic RSNA requests get P1/P2 priority
    // Flag personal RSNA requests for discussion
  }
}
```

Implementation Tasks: - [ ] Add reason field to vacation preferences schema - [ ] Create RSNA-specific validation logic - [ ] Update UI to capture reason when submitting preferences - [ ] Add discussion workflow for personal RSNA requests

2. March Break Specific Rules

Current State: No enforcement of "first or second week" rule or family status tracking

Required Implementation:

```
// Extend user profile
interface RadiologistProfile {
  // ... existing fields
  hasSchoolAgeChildren?: boolean
  spouseInTeachingProfession?: boolean
  familyStatus?: 'WITH_CHILDREN' | 'TEACHING_SPOUSE' | 'NONE'
}

// Add March Break validation
class MarchBreakConstraintValidator implements ConstraintValidator {
  validate(candidate: AssignmentCandidate, context: GenerationContext): ConstraintResult {
    // Enforce first OR second week (no splitting)
    // Validate P1/P2 priority requirement
    // Check family status for priority validation
  }
}
```

Implementation Tasks: - [ ] Add family status fields to user profiles - [ ] Create March Break week validation logic - [ ] Add family status tracking in database - [ ] Update preference UI to show March Break rules

3. Summer Holiday Limitations

Current State: No enforcement of 3-week P1 summer limit

Required Implementation:

```
// Add summer tracking
interface SummerVacationTracker {
  userId: string
  year: number
  p1WeeksUsed: number
  maxP1Weeks: number
  summerSeasonStart: Date
  summerSeasonEnd: Date
}

// Add validation
class SummerLimitConstraintValidator implements ConstraintValidator {
  validate(candidate: AssignmentCandidate, context: GenerationContext): ConstraintResult {
    // Check if user has exceeded 3-week P1 summer limit
    // Validate against summer season dates
  }
}
```

Implementation Tasks: - [ ] Add summer vacation tracking system - [ ] Create summer P1 limit validation - [ ] Add configurable summer season dates - [ ] Update UI to show summer limits

4. Part-time/Admin Day Scheduling

Current State: No PT/admin day request system

Required Implementation:

```
// New model for PT/Admin requests
interface PTAdminDayRequest {
  id: string
  userId: string
  requestedDate: Date
  reason: string
  priority: 'LOW' | 'MEDIUM' | 'HIGH'
  status: 'PENDING' | 'APPROVED' | 'REJECTED'
  conflictsWithVacation: boolean
}

// Add PT/Admin constraint
class PTAdminConstraintValidator implements ConstraintValidator {
  validate(candidate: AssignmentCandidate, context: GenerationContext): ConstraintResult {
    // Check PT/Admin requests against P1-P3 vacation
    // Prioritize vacation over PT/Admin requests
  }
}
```

Implementation Tasks: - [ ] Create PT/Admin day request system - [ ] Add PT/Admin request UI - [ ] Implement priority handling against vacation - [ ] Add notification system for PT/Admin conflicts

5. Protected Weekend Management

Current State: No weekend request tracking or limits

Required Implementation:

```
// Add weekend tracking
interface WeekendRequest {
  id: string
  userId: string
  weekendDate: Date
  reason: string
  status: 'PENDING' | 'APPROVED' | 'REJECTED'
}

// Add weekend limit validation
class WeekendLimitConstraintValidator implements ConstraintValidator {
  validate(candidate: AssignmentCandidate, context: GenerationContext): ConstraintResult {
    // Check if user has exceeded 5-weekend limit
    // Track weekend requests per user per year
  }
}
```

Implementation Tasks: - [ ] Create weekend request system - [ ] Add 5-weekend limit tracking - [ ] Create weekend request UI - [ ] Add exception handling for additional weekends

6. Christmas/New Year Formula

Current State: Blackout periods exist but no "3 working days off" formula

Required Implementation:

```
// Add Christmas formula logic
interface ChristmasFormula {
  year: number
  workingDaysOff: {
    beforeChristmas: Date[]
    beforeNewYear: Date[]
    afterChristmas: Date[]
  }
  additionalRequests: ChristmasAdditionalRequest[]
}

// Add Christmas constraint
class ChristmasFormulaConstraintValidator implements ConstraintValidator {
  validate(candidate: AssignmentCandidate, context: GenerationContext): ConstraintResult {
    // Enforce 3 working days off formula
    // Handle additional holiday time requests
  }
}
```

Implementation Tasks: - [ ] Implement "3 working days off" calculation - [ ] Create group approval workflow for additional time - [ ] Add Christmas formula validation - [ ] Integrate with existing blackout system

7. Emergency Coverage Procedures

Current State: Notification framework exists but no emergency coverage system

Required Implementation:

```
// Add emergency coverage system
interface EmergencyCoverage {
  id: string
  radiologistId: string
  shiftInstanceId: string
  reason: 'ILLNESS' | 'EMERGENCY' | 'LATE'
  notificationSent: Date
  coverageArranged: boolean
  coveringRadiologistId?: string
  escalationLevel: number
}

// Add coverage workflow
class EmergencyCoverageService {
  async handleEmergencyAbsence(absence: EmergencyCoverage): Promise<void> {
    // Send notifications to scheduler and site chief
    // Find available covering radiologist
    // Update schedule assignments
  }
}
```

Implementation Tasks: - [ ] Create emergency coverage system - [ ] Implement notification escalation - [ ] Add coverage arrangement tracking - [ ] Create emergency absence UI - [ ] Integrate with schedule update system

---

Implementation Roadmap

Phase 1: Data Model Extensions (2-3 weeks)

1. Extend vacation preferences schema with reason field
2. Add family status fields to user profiles
3. Create PT/Admin day request tables
4. Add weekend request tracking tables

5. Create emergency coverage tables

Phase 2: Business Logic Implementation (3-4 weeks)

1. Implement RSNA constraint validation
2. Add March Break week validation
3. Create summer P1 limit enforcement
4. Implement PT/Admin priority handling
5. Add weekend limit validation

Phase 3: UI/UX Enhancements (2-3 weeks)

1. Update vacation preference forms
2. Add family status management
3. Create PT/Admin request interface
4. Add weekend request interface
5. Create emergency coverage interface

Phase 4: Workflow Integration (2-3 weeks)

1. Implement group discussion workflows
2. Add Christmas formula calculation
3. Create emergency notification system
4. Integrate coverage arrangement tracking
5. Add approval workflows for exceptions

Phase 5: Testing & Validation (1-2 weeks)

1. Test all new constraint validations
2. Validate workflow integrations
3. Test emergency coverage procedures
4. Performance testing with new constraints
5. User acceptance testing

Technical Architecture Recommendations

Database Schema Extensions

```sql
-- Add reason field to vacation preferences
ALTER TABLE vacation_preferences
ADD COLUMN reason VARCHAR(50),
ADD COLUMN conference_type VARCHAR(50),
ADD COLUMN requires_discussion BOOLEAN DEFAULT FALSE;

-- Add family status to user profiles
ALTER TABLE radiology_profiles
```

```sql
ADD COLUMN has_school_age_children BOOLEAN DEFAULT FALSE,
ADD COLUMN spouse_in_teaching_profession BOOLEAN DEFAULT FALSE,
ADD COLUMN family_status VARCHAR(50) DEFAULT 'NONE';

-- Create PT/Admin day requests table
CREATE TABLE pt_admin_day_requests (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL,
  requested_date DATE NOT NULL,
  reason TEXT,
  priority VARCHAR(20) DEFAULT 'LOW',
  status VARCHAR(20) DEFAULT 'PENDING',
  conflicts_with_vacation BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Create weekend requests table
CREATE TABLE weekend_requests (
  id TEXT PRIMARY KEY,
  user_id TEXT NOT NULL,
  weekend_date DATE NOT NULL,
  reason TEXT,
  status VARCHAR(20) DEFAULT 'PENDING',
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Create emergency coverage table
CREATE TABLE emergency_coverage (
  id TEXT PRIMARY KEY,
  radiologist_id TEXT NOT NULL,
  shift_instance_id TEXT NOT NULL,
  reason VARCHAR(50) NOT NULL,
  notification_sent TIMESTAMP DEFAULT NOW(),
  coverage_arranged BOOLEAN DEFAULT FALSE,
  covering_radiologist_id TEXT,
  escalation_level INTEGER DEFAULT 1,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);
```

API Endpoints to Add

```
// PT/Admin day requests
POST /api/pt-admin-requests
```

```
GET /api/pt-admin-requests
PUT /api/pt-admin-requests/:id/approve
PUT /api/pt-admin-requests/:id/reject

// Weekend requests
POST /api/weekend-requests
GET /api/weekend-requests
PUT /api/weekend-requests/:id/approve
PUT /api/weekend-requests/:id/reject

// Emergency coverage
POST /api/emergency-coverage
GET /api/emergency-coverage
PUT /api/emergency-coverage/:id/arrange
PUT /api/emergency-coverage/:id/escalate

// Family status management
PUT /api/users/:id/family-status
GET /api/users/:id/family-status
```

Constraint Engine Extensions

```
// Add new constraint validators
export class RSNAConstraintValidator implements ConstraintValidator { }
export class MarchBreakConstraintValidator implements ConstraintValidator { }
export class SummerLimitConstraintValidator implements ConstraintValidator { }
export class PTAdminConstraintValidator implements ConstraintValidator { }
export class WeekendLimitConstraintValidator implements ConstraintValidator { }
export class ChristmasFormulaConstraintValidator implements ConstraintValidator { }
```

Conclusion

ShiftPilot already has an excellent foundation for implementing the scheduling rules. The core constraint engine, vacation preference system, and fairness algorithms are well-designed and can be extended to handle the specific institutional rules.

The main implementation work involves: 1. **Data model extensions** to capture additional information 2. **Business logic implementation** for specific rule enforcement 3. **UI/UX enhancements** for new request types 4. **Workflow integration** for approval processes

With the existing architecture, these enhancements can be implemented incrementally without disrupting the current system functionality.

Estimated Total Implementation Time: 10-15 weeks **Risk Level**: Low (building on existing solid foundation) **Business Impact**: High (complete compliance with institutional scheduling rules)

This analysis was conducted on January 17, 2025, based on ShiftPilot v2.0 codebase and the provided scheduling rules document.