EMPLOYEE SHIFT SCHEDULING SYSTEM

Introduction

- Managing intramural sports staff scheduling is a complex challenge faced by university recreation departments. Coordinators must balance numerous constraints: employee availability varies by semester, individuals have different sport expertise levels, weekly hour limits must been forced, and each game requires specific numbers of supervisors and referees. Manual scheduling is time-consuming and error-prone, often resulting in double-bookings, unqualified staff assignments, or violation of labor regulations.
- We have attempted to develop an automated scheduling system that uses intelligent recommendation algorithms to suggest optimal staff assignments while respecting all constraints.
- The system features dashboards for administrators to keep track of supervisor/employee assignments and the schedule.
- By attempting to automate the tedious constraint checking and providing smart recommendations based on expertise, experience andother factors, we enable admins to complete scheduling in minutes rather than hours, while ensuring compliance with all regulations

Problem Description

The Challenge of Manual Scheduling

- **Time-Consuming Manual Process**
- Administrators spend 5-8 hours weekly creating schedules
- Checking individual employee availability manually
- Cross-referencing qualifications and certifications
- Managing last-minute changes and conflicts
- **High Error Rate**
- Double-booking employees across multiple games
- Forgetting approved time-off requests
- Exceeding maximum work hours (labor law violations)
- Assigning unqualified staff to specialized sports

- **Communication Breakdown**
- Information scattered across emails, spreadsheets, and text messages
- Employees unaware of schedule changes
- No centralized system for time-off requests
- Difficulty tracking worked hours for payroll

Analysis

- The Challenge:
- We discovered three critical user personas: administrators who create schedules and assign staff, supervisors who oversee games, and staff members who submit availability and view assignments. Through workflow analysis, we identified key pain points: tracking which employees can work which sports, managing time-off requests, preventing double-bookings, and ensuring weekly hour limits aren't exceeded.
- Core Concept:
- When we began designing our scheduling system, we faced a fundamental choice between two approaches:
- **Automated Scheduling** uses specialized constraint-solving software that examines the entire schedule at once and automatically determines the optimal assignment for every position. Think of it as a super-intelligent assistant that considers every possible arrangement and guarantees perfect results.
- **Manual Scheduling** provides smart recommendations to administrators who make the final decisions. The system analyzes candidates, eliminates ineligible options, ranks the remaining choices, and presents the top two for administrator selection.

Automated Scheduling

- **Automated Scheduling**: Automated scheduling employs what's called a "constraint solver"
- These systems treat scheduling as one interconnected optimization problem. Over 30-60 seconds, they evaluate millions of assignment combinations, guaranteeing mathematically that no rules are violated - no double-booking, no hour limit violations, no constraint failures possible.
- **Guaranteed Rule Enforcement:** This is the key advantage. The software mathematically guarantees it will never violate hard constraints. Double-booking becomes impossible. Exceeding the 20-hour limit becomes impossible. It's built into the fundamental design - the system literally cannot produce an invalid schedule.
- **Built-In Fairness:** The algorithm actively optimizes for even workload distribution. Instead of always choosing your best employees, it balances quality with fairness, ensuring everyone receives approximately equal opportunities and hours.
- **Why We Didn't Choose It:** Implementation requires 6-10 weeks with specialized optimization frameworks. Our team lacked experience with constraint-solving technology, and our 4-week project timeline made this approach impractical. The learning curve and implementation complexity outweighed the benefits for our scale.

Manual Scheduling

- **Manual Scheduling**:
- **Filtering Logic:** When an administrator selects a position to fill, we automatically eliminate ineligible candidates - those lacking required sport expertise, already assigned to overlapping games, exceeding hour limits, or having scheduling conflicts.
- **Scoring Algorithm:** We rank remaining candidates using weighted criteria: sport expertise (30%),performance rating (25%), hour balance (20%), recency fairness (15%), and preference matching (10%).These weights reflect our stakeholders' priorities - expertise matters most, followed by quality, fairness, and employee satisfaction.
- **Why we chose this?:** This method required experience in SQL databases. JAVA development and JavaFX UI design. Our limited experience in constraint-solving algorithms and frameworks made the manual scheduling approach our top choice in the limited 3-4 week project timeline.

Design (Algorithms & System Architecture)

- **System Architecture:** We designed a three-tier MVC (Model-View-Controller) architecture. The Model layer contains nine core entities: Users, Employees, Sports, Game Schedules, Shifts, Seasonal Availability, Permanent Conflicts, Weekly Hours, and Employee Expertise. The Controller layer processes business logic and coordinates between views and data access objects. The View layer implements JavaFX interfaces for administrator and staff dashboards.
- **Data Flow Summary**
- User interactions flow from View → Controller → Service → DAO → Database for write operations, with results flowing back through the same layers to update the UI. For example, creating an employee flows:AddEmployeeModal → EmployeeManagementService → UserDAO/EmployeeDAO/AvailabilityDAO →Database → success confirmation → refresh AdminDashboard. Generating recommendations flows:AdminDashboard → SchedulingController → loads data via DAOs → SchedulingEngine scores candidates →ShiftDAO persists → UI displays options.

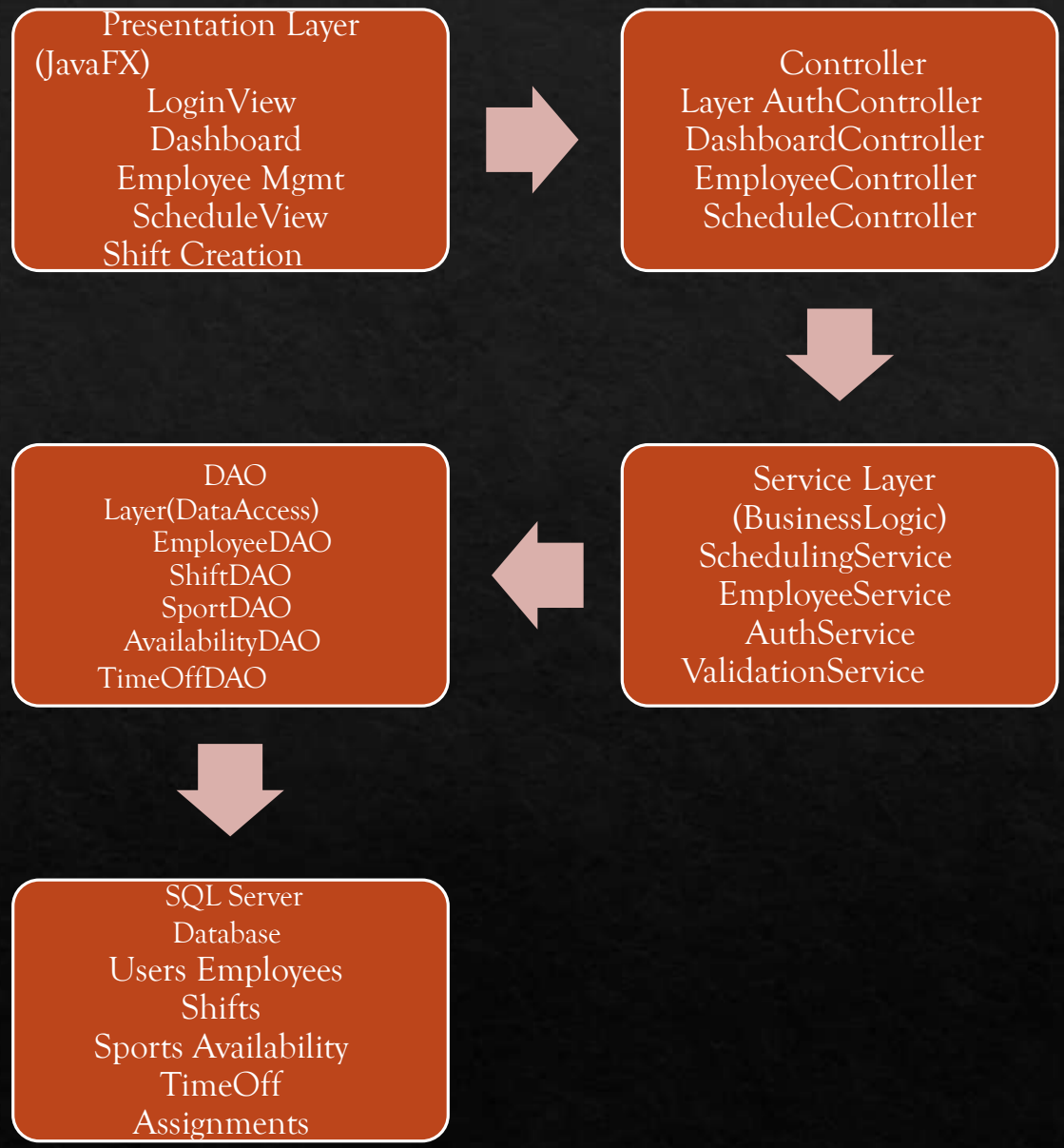
Design (Algorithms & System Architecture)

- **System Architecture Overview**
- We designed a three-tier MVC architecture separating data representation, business logic, and user interface into distinct layers with unidirectional dependencies flowing from View → Controller → Service → DAO → Database.
- **Layer Design**
- **Model Layer:** We created nine domain classes (User, Employee, Sport, Game Schedule, Shift, Seasonal Availability, Permanent Conflict, Weekly Hours, Time Off (Request) representing business entities with encapsulated behaviors.
- **DAO Layer:** We implemented nine Data Access Object classes handling all JDBC database interactions through CRUD operations andspecialized queries, isolating SQL from business logic.
- **Service Layer:** We developed five service classes (SchedulingEngine, ConflictChecker, HoursTracker, EmployeeManagementService,AuthenticationService) orchestrating complex multi-step operations and implementing business rules.
- **Util Layer:** We built utility classes (DateTimeUtil, ValidationUtil, TimeSlot) providing shared helper methods for date/time operations,input validation, and common calculations.

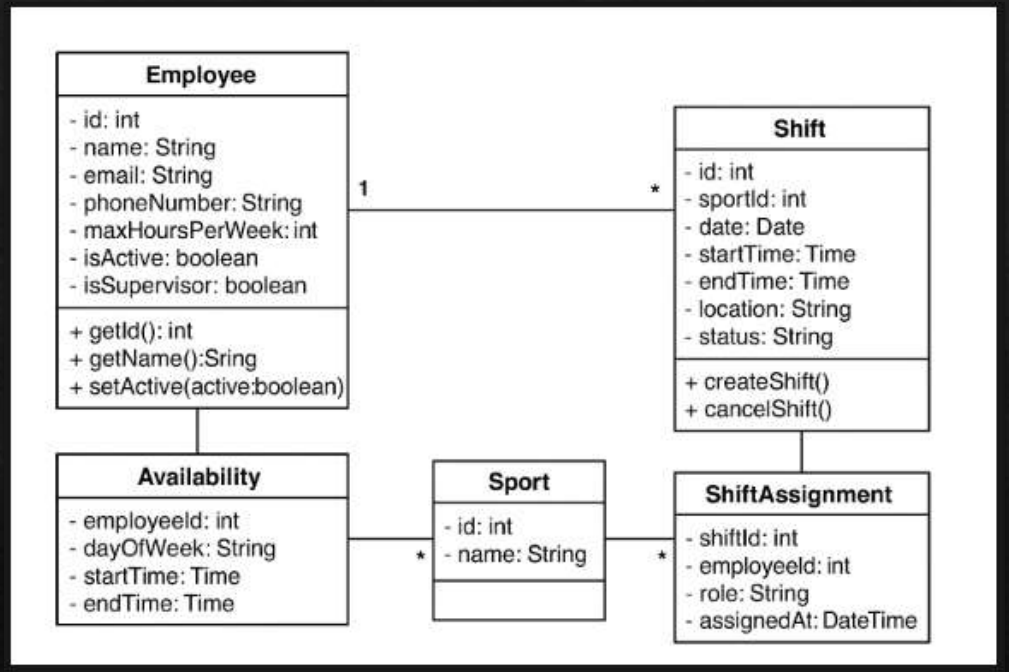
Design (Algorithms & System Architecture)

- **Recommendation Algorithm**
- **Two-Phase Process:** We have attempted to design a filtering-then-scoring algorithm where Phase 1 eliminates ineligible candidates through hard constraint validation and Phase 2 ranks remaining candidates using weighted scoring.
- **Phase 1 - Filtering:** Conflict Checker validates five constraints - active status, sport expertise, no time conflicts, under 20-hour limit, no permanent conflicts - rejecting any employee violating these rules.
- **Phase 2 - Scoring:** We calculate score = (0.30 × expertise) + (0.25 × hour Balance) + (0.20 × performance) + (0.15 ×recency) + (0.10 × preference) based on stakeholder priorities.
- **Duplicate Prevention:** We are working on duplicate prevention and compliance with scheduling conflicts and Overlap detection.
- **Output:** The algorithm returns the top two scored candidates as Option A and Option B for administrator selection.

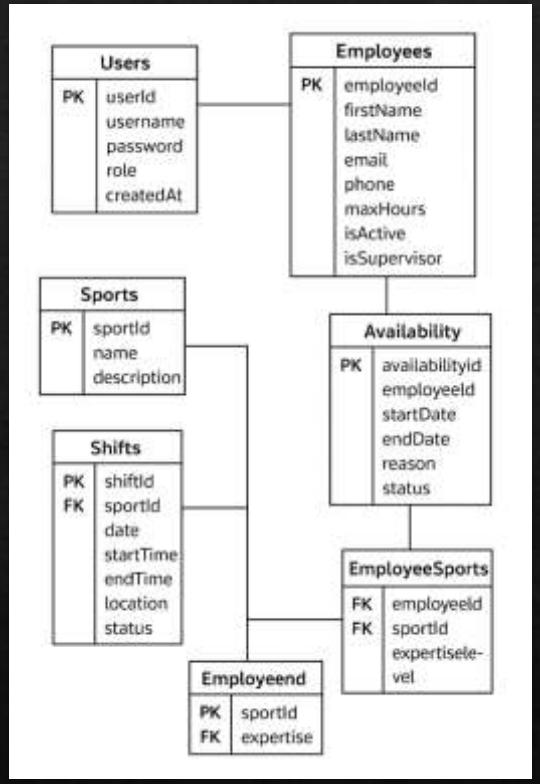
System Architecture



UML Diagram



ER Diagram

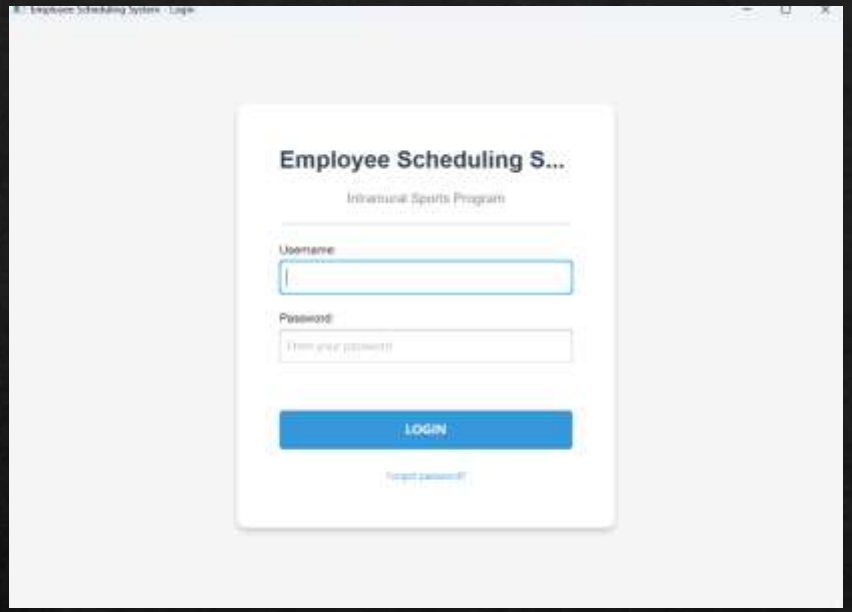


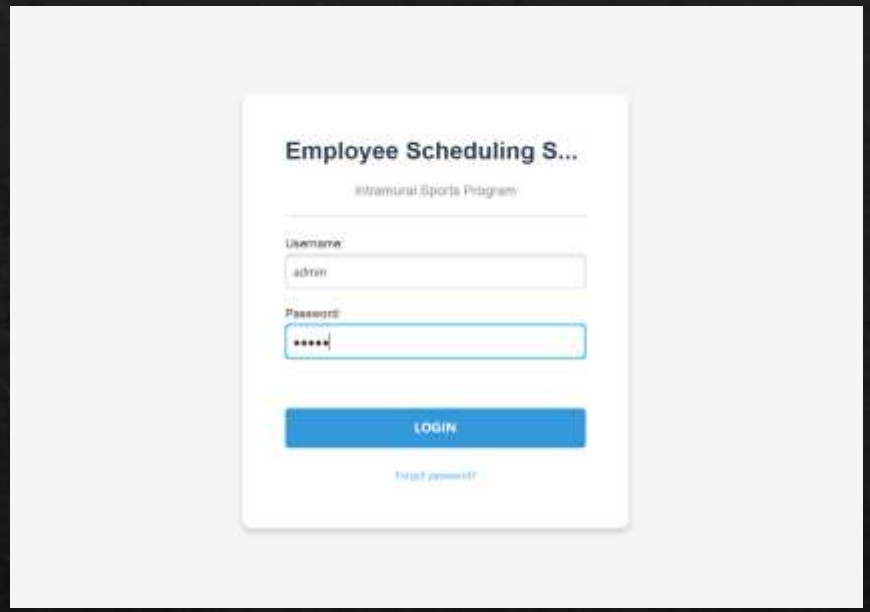
Implementation - Technologies & APIs

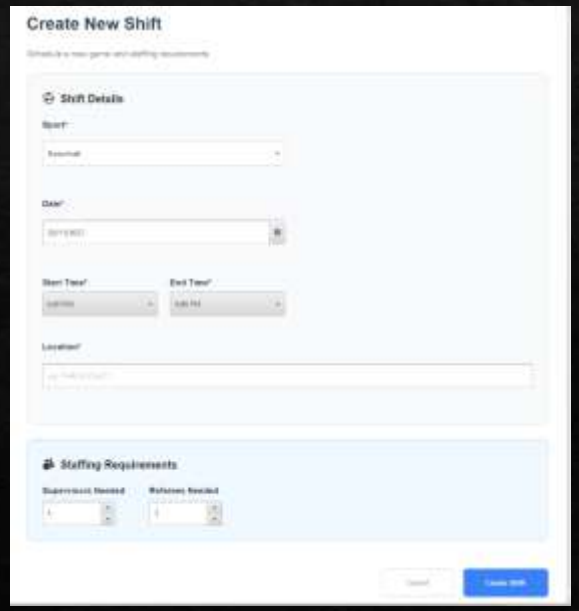
- **Core Technologies**
- **IDE:** We developed using Eclipse/IntelliJ IDEA providing code completion, debugging capabilities, and integrated build tools.
- **Build Tool:** We used Maven for dependency management, though our project primarily relied on Java standard library and JavaFX bundled with JDK.
- **Version Control:** We managed code using Git for version control and collaboration among team members.
- **Database Tools:** We used SQL Server Management Studio (SSMS) for database design, schema creation, and query testing during development.
- **MVC Pattern:** We separated concerns with Model classes representing data, View classes handling UI, and Controller classes coordinating between them, ensuring maintainability and testability.
- **Database Implementation**
- We used **JDBC API** for all database connectivity, executing queries through prepared statements to prevent SQL injection. We implemented the **DAO Pattern** to isolate database operations, with each DAO handling object-to-table mapping and transaction management. We configured connections through a utility class loading credentials from properties files. We managed transactions explicitly for multi-step operations, committing on success and rolling back on errors. We created indexes on frequently-queried columns to accelerate query execution time.

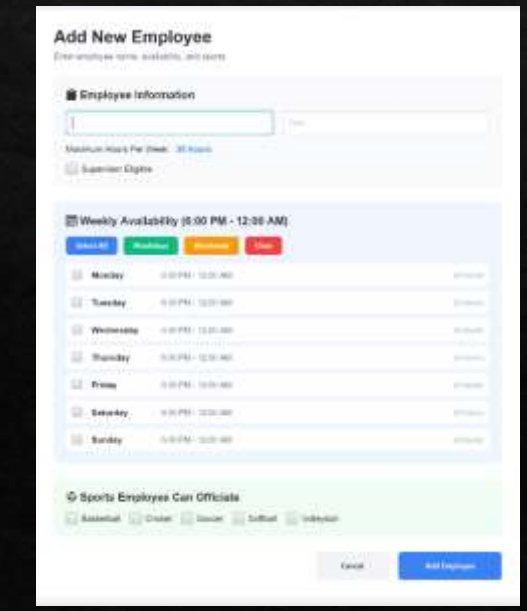
Implementation - Technologies & APIs

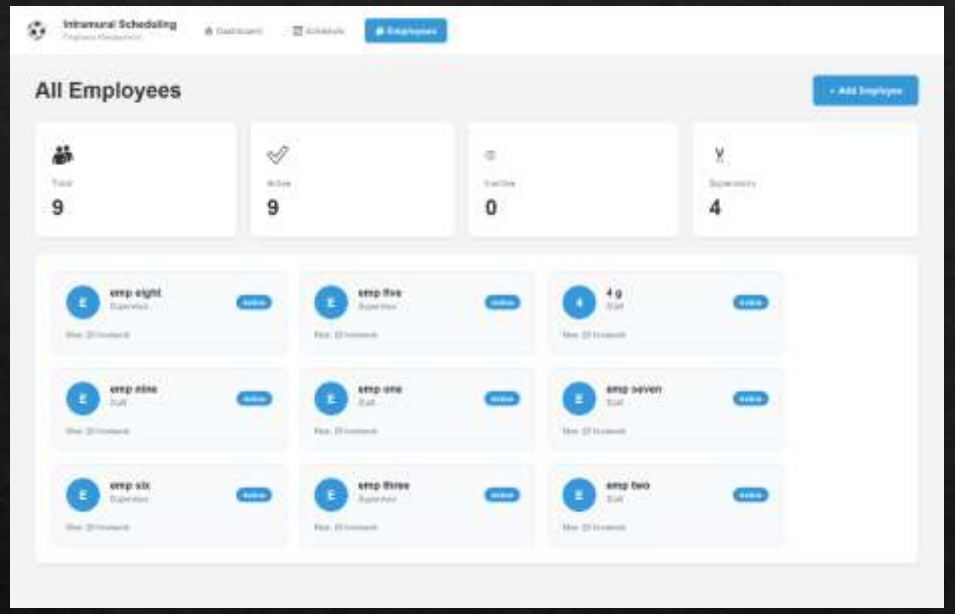
- **Business Logic Implementation**
- We leveraged **Java Time API** extensively for all date and time operations including calculating shift durations, determining week boundaries, and checking time overlaps. We used **Java Collections Framework** with lists for ordered data, maps for indexed lookups, and sets for tracking unique items like already-recommended employees. We applied **Java Streams API** for functional operations including filtering eligible candidates, transforming data structures, and sorting recommendations by score. We employed **Java Security API** for password hashing using SHA-256 encryption with Base64 encoding.
- **User Interface Implementation**
- We built responsive layouts using **JavaFX Scene Graph** containers organized hierarchically for main panels, scrollable regions, and grid arrangements. We utilized standard **JavaFX Controls** for all user interactions including text inputs, date pickers, dropdown menus, radio button groups for exclusive selections, checkboxes for multiple options, and action buttons. We created modal dialogs that block parent windows during data entry operations. We applied custom CSS styling for professional appearance including colors, fonts, spacing, and hover effects. We implemented reactive state management maintaining session data in memory for tentative assignments and UI expansion tracking. We used callbacks to enable communication between views without tight coupling, allowing child dialogs to trigger parent view refreshes upon successful operations.

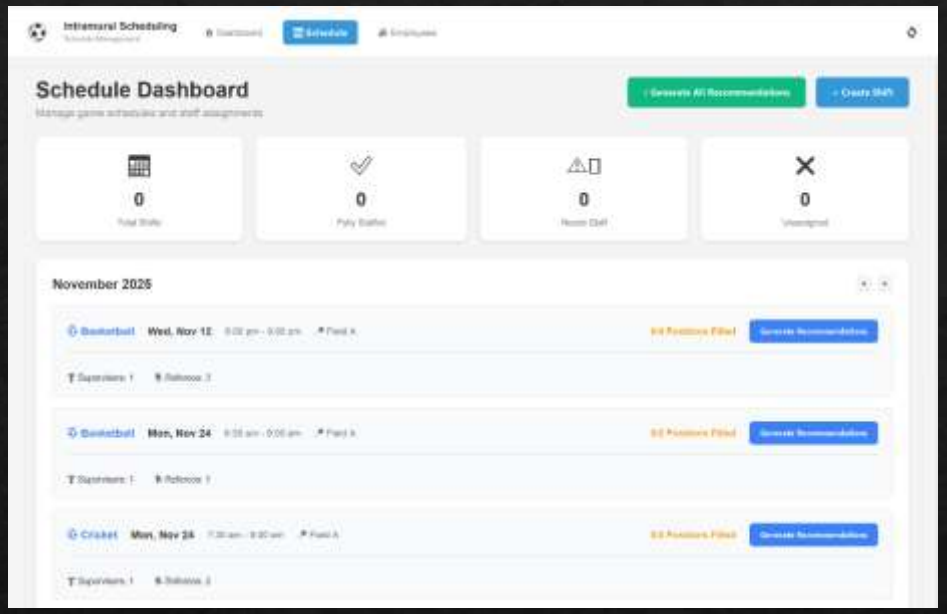

Employee Scheduling S...
Intramural Sports Program
Username
Password
LOGIN

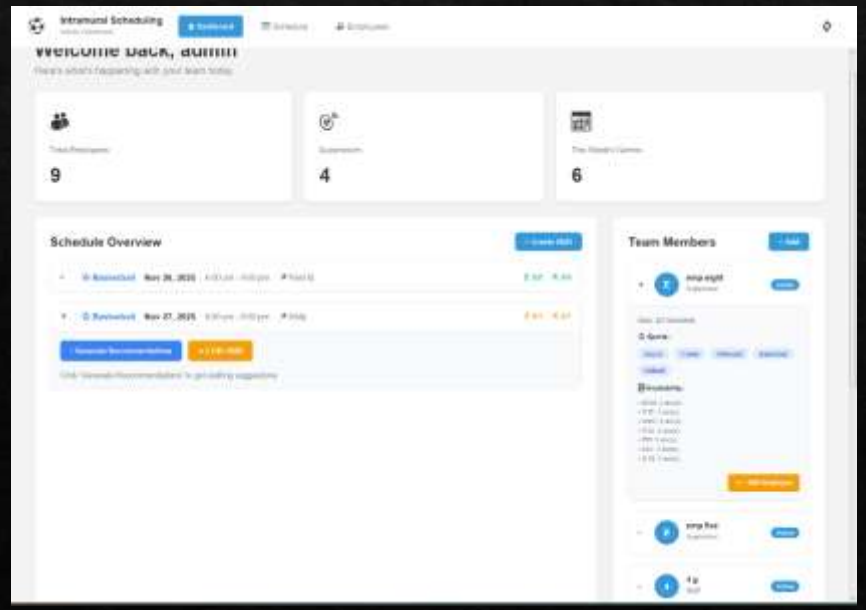

Employee Scheduling S...
Intramural Sports Program
Username
admin
Password
LOGIN


Create New Shift
Shift Details
Staffing Requirements

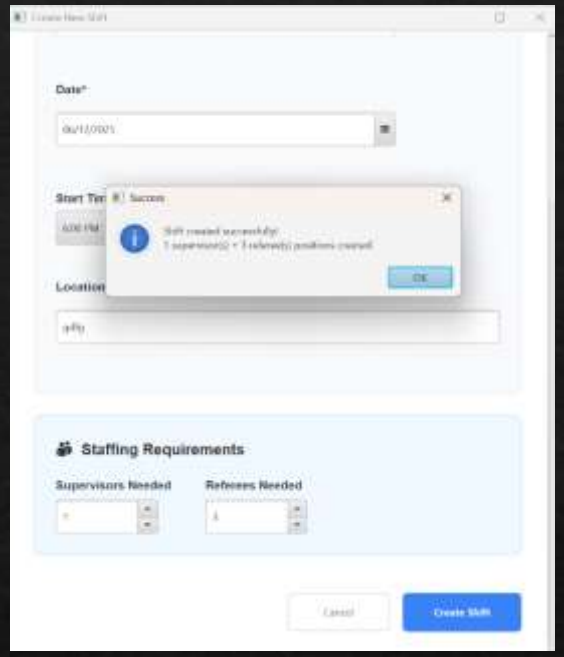

Add New Employee
Employee Information
Weekly Availability (6:00 PM - 12:00 AM)
Sports Employee Can Officiate

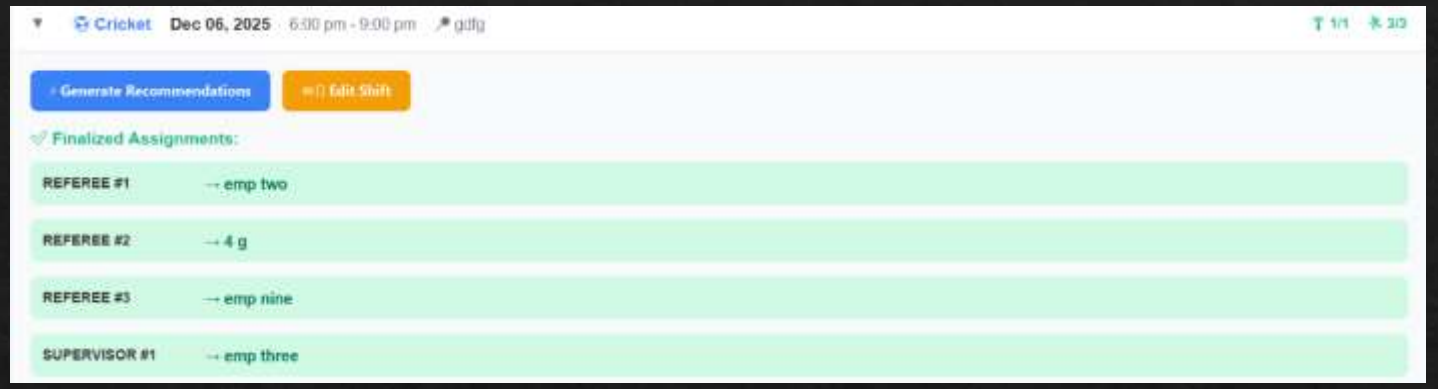

Intramural Scheduling
All Employees
9
9
0
4
emp eight
emp five
4 g
emp nine
emp one
emp seven
emp six
emp three
emp two

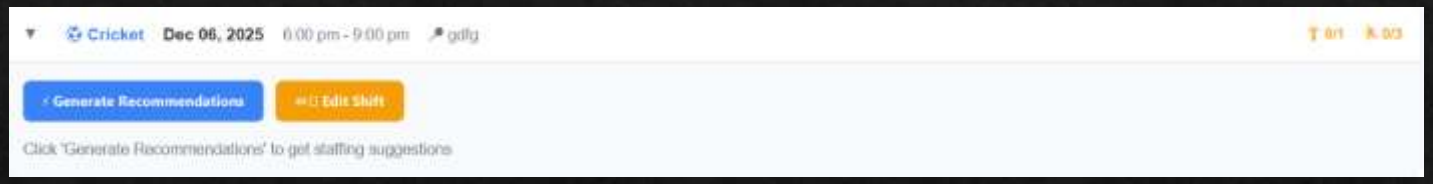

Intramural Scheduling
Schedule Dashboard
0
0
0
0
November 2025
Basketball Wed, Nov 12
Basketball Mon, Nov 24
Cricket Mon, Nov 24

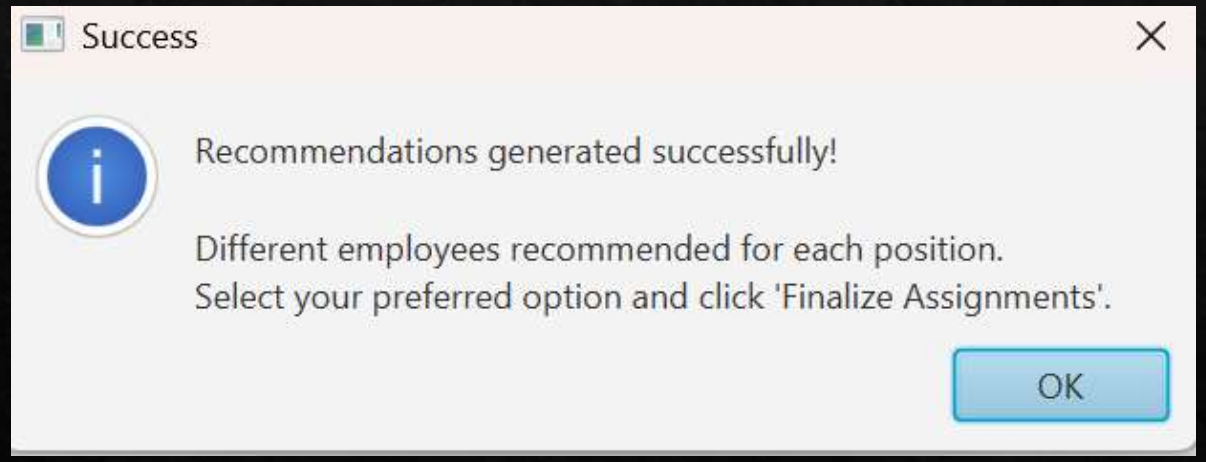

Intramural Scheduling
9
4
6
Schedule Overview
Team Members

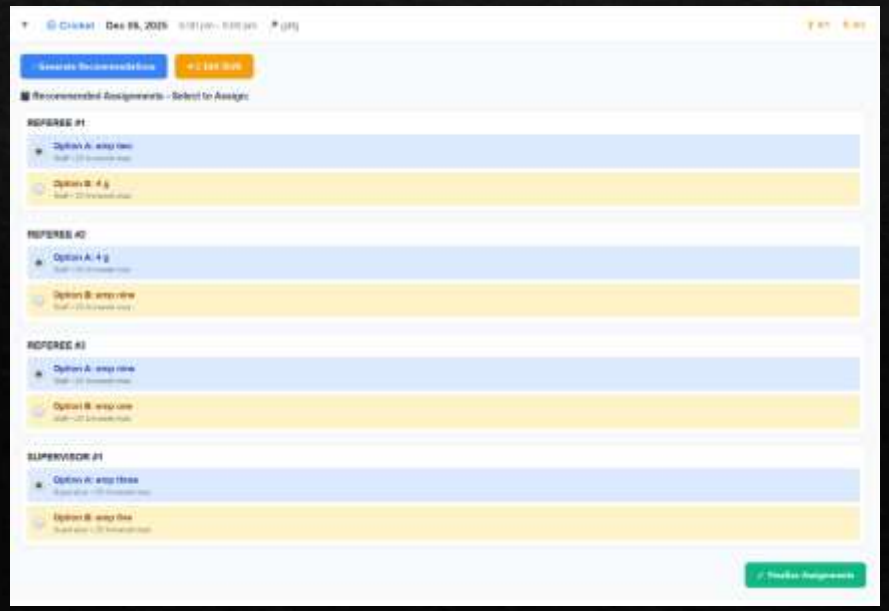

Basketball Dec 06, 2025 6:00 pm - 9:00 pm Field A
Generate Recommendations
Finalized Assignments:
REFEREE #1 → emp eight
REFEREE #2 → 4 g
REFEREE #3 → emp nine
SUPERVISOR #1 → emp three


Date*
Success
OK
Location
Staffing Requirements
Supervisors Needed
Referees Needed
Cancel
Create Shift


Cricket
Dec 06, 2025
6:00 pm - 9:00 pm
gdfg
1/1
3/3
Generate Recommendations
Edit Shift
Finalized Assignments:
REFEREE #1
→ emp two
REFEREE #2
→ 4 g
REFEREE #3
→ emp nine
SUPERVISOR #1
→ emp three


Cricket
Dec 06, 2025
6:00 pm - 9:00 pm
gdfg
0/1
0/3
Generate Recommendations
Edit Shift
Click 'Generate Recommendations' to get staffing suggestions


Success
Recommendations generated successfully!
Different employees recommended for each position.
Select your preferred option and click 'Finalize Assignments'.
OK



Discussion

- **Project Limitations and Challenges**
- **Incomplete Constraint Implementation:** Our system remains a work-in-progress regarding comprehensive constraint handling. We successfully implemented core constraints like double-booking prevention and hour limits, but deferred permanent conflicts, time-off requests, and detailed availability patterns to future development. The MVP treats missing availability data as "always available" rather than enforcing strict availability windows, acknowledging this simplification needs refinement for production deployment.
- **Recommendation vs Direct Assignment Trade-off:** We discovered that providing multiple recommendation options significantly increases complexity compared to direct automatic assignment. Offering Option A and Option B requires real-time conflict checking as administrators make selections, tracking tentative assignments across the entire session to prevent overlapping choices. Direct assignment would be simpler to implement but sacrifices administrator control, which stakeholders valued highly for this domain.
- **Static Weight Configuration:** Our scoring algorithm uses fixed weights (expertise 30%, hour balance 25%, etc.) that don't adapt over time. Ideally, weights should adjust based on weekly performance metrics and seasonal priorities, but implementing dynamic weight recalculation proved too complex for our timeline. This limitation means the system can't learn from successful assignments or adapt to changing organizational priorities without code modifications.
- **UI Visualization Limitations:** Our current expandable card interface, while functional, lacks the intuitive visual clarity of calendar-based scheduling. Displaying shifts on an interactive weekly or monthly calendar grid would provide better schedule overview and pattern recognition. However, implementing drag-and-drop calendar interactions with real-time conflict validation exceeded our four-week scope, leading us to choose the simpler list-based approach despite inferior user experience for schedule visualization.

Conclusion

- **Project Completion Status**
- **Successfully Implemented Components:** We delivered a functional scheduling system within our four-week timeline with complete implementation of core features. The database layer includes all nine tables with proper relationships and indexes. The recommendation engine successfully filters candidates through hard constraint validation and scores them using weighted criteria. The hour tracking system automatically updates weekly totals when shifts are assigned. The UI provides expandable game and employee cards with radio button selection for assignment finalization. Double-booking prevention operates through session-state tracking and database validation. The system processes recommendations for 20+ employees
- **Incomplete or Partially Implemented Components:** The time-off request system exists only at the database and model layers—controller and view implementation remain incomplete, preventing employees from submitting requests or administrators from approving them. Permanent conflict management lacks UI for employee entry and validation during recommendation generation. Seasonal availability enforcement is simplified—the system treats missing data as "always available" rather than strictly validating availability windows. The notification system is entirely absent—no email alerts for assignments or schedule changes. Calendar-based visualization was not implemented—the current list-based interface lacks intuitive schedule overview. Performance tracking and dynamic weight adjustment are missing—the scoring algorithm uses fixed weights that cannot adapt based on historical data.

Future Work

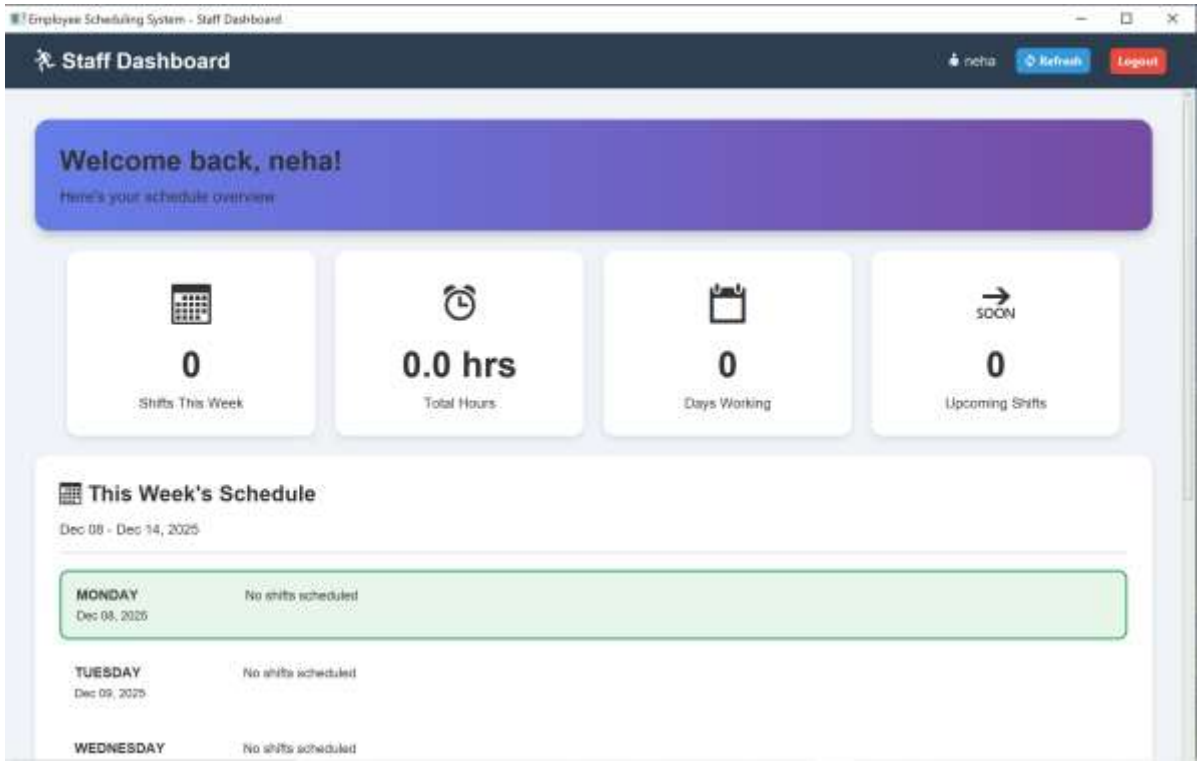
- CONSTRAINT SYSTEM IMPROVEMENTS:
- **Complete Time-Off Implementation:** Develop full workflow allowing employees to submit time-off requests with administrator approval interface and automatic exclusion from recommendations during approved periods.
- **Permanent Conflict Enforcement:** Create UI for employees to specify recurring unavailability (classes, other jobs) and integrate validation into recommendation generation preventing assignments during conflict times.
- **Strict Availability Validation:** Replace MVP "always available" assumption with strict enforcement requiring employees to have availability entries covering shift times before appearing in recommendations.
- **Concurrent Session Handling:** Add database-level locking mechanisms preventing race conditions when multiple administrators schedule simultaneously, with conflict resolution UI for overlapping modifications.
- **Advanced Constraint Rules:** Implement minimum rest periods between shifts, maximum consecutive work days, and sport certification expiration checking to ensure compliance with labor regulations and safety requirements.

Future Work

- DESIGN AND ALGORITHM ENHANCEMENTS:
- **Calendar-Based UI:** Replace list interface with interactive calendar grid showing all games in weekly/monthly views. Implement drag-and-drop assignment with real-time conflict validation and visual indicators (color-coding, icons) for staffing status and constraint violations.
- **Adaptive Scoring Algorithm:** Develop dynamic weight adjustment learning from historical assignment outcomes. Track attendance reliability, supervisor ratings, and employee satisfaction to automatically refine scoring weights through regression analysis of successful versus problematic assignments.
- **Global Optimization:** Explore constraint solver integration (Timefold Solver) for automated batch schedule generation with administrator override capability. Compare automated versus manual schedules using fairness metrics to evaluate whether full automation justifies increased complexity.
- **Multi-Objective Optimization:** Replace fixed weighted scoring with Pareto efficiency approaches optimizing multiple objectives simultaneously. Allow administrators to specify objective priorities per scheduling cycle with visualization showing trade-offs between competing goals.
- **Conflict Resolution Workflow:** Design interface for handling unavoidable constraint violations when insufficient staff exist. Provide relaxation suggestions and "what-if" analysis allowing administrators to simulate constraint modifications before committing changes.

Staff Panel

- Personal schedule view for employees
- View assigned shifts
- Check upcoming games
- Request time-off through system
- Update availability preferences



Reflection on technical Decisions

- **Architecture Choice Validation:** The layered MVC architecture proved highly effective, enabling parallel development and future maintainability through clear layer boundaries and unidirectional dependencies.
- **Manual Scheduling Justification:** Choosing manual recommendation over automated constraint solving was appropriate for our four-week timeline and team expertise, successfully delivering working functionality using familiar technologies rather than risking project failure learning specialized frameworks.
- **Session State Complexity:** Implementing Option A/B recommendations with session-state tracking introduced significant complexity but preserved administrator control that stakeholders valued, though the fragility of tentative assignment tracking represents technical debt requiring attention.
- **Duplicate Prevention Innovation:** Our penalty-based tracking mechanism preventing employee recommendation monopolization represents novel problem-solving demonstrating domain-specific fairness understanding beyond standard implementations, validated through testing showing effective distribution across the employee pool.

Job Assignment and Workload Division

- **Misha - Database & UI Integration (50%):** Designed database schema with nine tables and implemented all DAO classes handling JDBC operations and transaction management. Built complete JavaFX interface including Admin Dashboard, Schedule Builder, Employees Page, and modals with expandable cards, styling, and event handling, Staff Panel.
- **Anuj - Architecture & Algorithm Development (50%):** Designed three-tier MVC architecture and created all model classes. Developed five service classes including Scheduling Engine with recommendation algorithm, Conflict Checker, Hours Tracker, and Authentication Service. Implemented duplicate prevention mechanism, weighted scoring, and all controller/utility classes.