PRODUCT REQUIREMENT DOCUMENT (PRD)

Smart Classroom & Timetable Scheduler

---

1. PROJECT OVERVIEW

The Smart Classroom & Timetable Scheduler is a production-ready web-based platform designed to automate the complex process of timetable generation for large-scale higher education institutions. Built on the MERN stack (MongoDB, Express.js, React, Node.js), the system addresses challenges in scheduling by optimizing resource allocation, preventing clashes, and supporting multidisciplinary learning as per NEP 2020 guidelines.

Target Scale:

- 20+ Departments
- 100+ Branches
- 500+ Sections
- Multiple faculty members teaching across departments
- Labs requiring extended durations

---

2. PROBLEM STATEMENT

Higher education institutions face significant challenges in timetable preparation:

- **Manual Scheduling Issues:** Time-consuming process prone to human errors
- **Resource Constraints:** Limited classrooms, labs, and faculty availability
- **Clash Management:** Faculty, room, and section conflicts
- **Uneven Distribution:** Imbalanced faculty workload and classroom utilization
- **NEP 2020 Complexity:** Multidisciplinary courses and electives increase scheduling complexity
- **Scalability:** Managing 500+ sections across 100+ branches manually is impractical
- **Transparency:** Lack of visibility into workload distribution and resource utilization

---

3. OBJECTIVES

1. **Automate Timetable Generation:** Reduce manual effort by 90%+ through intelligent scheduling algorithms
2. **Eliminate Clashes:** Ensure zero faculty, room, and section conflicts
3. **Optimize Resource Utilization:** Maximize classroom and lab usage efficiency
4. **Balance Workload:** Distribute teaching hours fairly among faculty
5. **Support Large Scale:** Handle 20+ departments, 100+ branches, 500+ sections seamlessly
6. **Ensure Flexibility:** Accommodate labs, electives, and cross-departmental teaching
7. **Provide Transparency:** Real-time visibility into schedules and workload
8. **Enable Easy Modifications:** Quick adjustments and regeneration capabilities
9. **Production Ready:** Scalable, secure, and maintainable system

---

4. SCOPE OF THE SYSTEM

In Scope:

- Faculty, subject, department, branch, section management
- Room and lab inventory management
- Automated timetable generation with constraint satisfaction
- Support for labs with extended duration (100+ minutes)
- Weekly hours allocation per subject
- Multi-department faculty assignment
- Clash detection (faculty, room, section)
- Multiple timetable generation alternatives
- Faculty workload analytics and visualization
- Export functionality (PDF, Excel)
- Role-based access control (Admin, Faculty, HOD, Student)
- Responsive web interface

- Real-time notifications for clashes and updates

Out of Scope (Future Enhancements):

- Mobile native applications
- AI-powered predictive analytics
- Student attendance integration
- Exam scheduling
- Teacher evaluation integration
- Automated room booking beyond timetable

---

5. USER ROLES

5.1 Admin

- **Access Level:** Full system access
- **Responsibilities:**
    - Add/edit/delete faculty, subjects, rooms, departments
    - Configure system parameters (working days, time slots)
    - Generate and publish timetables
    - Manage user accounts
    - View system-wide analytics

5.2 HOD (Head of Department)

- **Access Level:** Department-specific access
- **Responsibilities:**
    - View department timetables
    - Monitor faculty workload within department
    - Request timetable modifications
    - Export department schedules

5.3 Faculty

- **Access Level:** Personal view access
- **Responsibilities:**
    - View personal timetable
    - Check teaching workload
    - View assigned rooms and sections
    - Export personal schedule

5.4 Student

- **Access Level:** Section-specific view
- **Responsibilities:**
    - View section timetable
    - Check room assignments
    - Export class schedule

---

6. FUNCTIONAL REQUIREMENTS

6.1 Faculty Management

FR-1.1: System shall allow Admin to add faculty with following details:

- Faculty ID (unique)
- Name
- Department(s) affiliated
- Email and contact
- Specialization/subjects they can teach
- Maximum weekly teaching hours
- Preferred time slots (optional)

FR-1.2: System shall support faculty teaching multiple departments

FR-1.3: System shall validate faculty availability before assignment

FR-1.4: System shall track faculty workload in real-time

6.2 Subject Management

FR-2.1: System shall allow Admin to create subjects with:

- Subject code (unique)
- Subject name
- Department and branch
- Subject type (Theory/Lab/Practical)
- Weekly hours required
- Semester/Year
- Eligible faculty list

FR-2.2: System shall support elective subjects with multiple faculty options

FR-2.3: System shall allow different durations for theory (50-60 min) and lab (100+ min) sessions

---

6.3 Department & Section Management

FR-3.1: System shall manage departments with:

- Department code and name
- HOD assignment
- Associated branches

FR-3.2: System shall manage branches with:

- Branch code and name
- Parent department
- Semester/year divisions
- Number of sections

FR-3.3: System shall manage sections with:

- Section identifier (e.g., CSE-3A, CSE-3B)
- Student strength

- Branch and semester
- Subject assignments

FR-3.4: System shall scale to support 20+ departments, 100+ branches, 500+ sections

---

6.4 Room & Lab Management

FR-4.1: System shall maintain room inventory with:

- Room number/name
- Room type (Classroom/Lab/Auditorium)
- Capacity
- Department allocation (if any)
- Available equipment/facilities

FR-4.2: System shall track room availability in real-time

FR-4.3: System shall prioritize lab allocation for lab subjects

FR-4.4: System shall prevent double-booking of rooms

6.5 Timetable Generation

FR-5.1: System shall generate weekly timetables automatically based on constraints

FR-5.2: System shall allow configuration of:

- Working days (5 or 6 days)
- Time slots per day (typically 7-8 slots)
- Slot duration (50-60 minutes for theory, 100+ for labs)
- Break times
- Lunch hours

FR-5.3: System shall generate timetables considering:

- Subject weekly hours requirement
- Faculty availability and maximum workload

- Room availability and capacity
- Section student strength
- Lab duration requirements

FR-5.4: System shall provide multiple alternative timetables if requested

FR-5.5: System shall allow manual adjustments to generated timetables

FR-5.6: System shall save timetable versions with timestamps

FR-5.7: System shall publish approved timetables to respective users

---

6.6 Lab Assignment Logic

FR-6.1: System shall allocate consecutive slots for labs (e.g., 2 consecutive 50-min slots for 100-min lab)

FR-6.2: System shall ensure lab rooms are assigned to lab subjects

FR-6.3: System shall verify lab capacity matches section strength

FR-6.4: System shall schedule labs preferably in later slots to allow room flexibility

---

6.7 Clash Detection Logic

FR-7.1: Faculty Clash Detection

- System shall prevent assigning same faculty to multiple sections at same time
- System shall alert if faculty exceeds maximum weekly hours

FR-7.2: Room Clash Detection

- System shall prevent double-booking of rooms
- System shall alert if room capacity is insufficient

FR-7.3: Section Clash Detection

- System shall prevent scheduling multiple subjects for same section simultaneously
- System shall ensure sections don't have gaps exceeding configured limits

FR-7.4: System shall display all detected clashes with:

- Clash type
- Affected entities (faculty/room/section)
- Conflicting time slots
- Suggested resolutions

FR-7.5: System shall prevent timetable publication if unresolved clashes exist

---

6.8 Faculty Workload View

FR-8.1: System shall display faculty workload dashboard showing:

- Total weekly teaching hours
- Subject-wise distribution
- Section-wise distribution
- Department-wise distribution (for multi-dept faculty)
- Daily workload distribution
- Comparison with maximum allowed hours

FR-8.2: System shall generate workload reports for HODs and Admin

FR-8.3: System shall highlight faculty with uneven workload distribution

---

6.9 Export Functionality

FR-9.1: System shall export timetables to PDF format with:

- Professional formatting
- Institution logo and header
- Section/faculty/room-wise views
- Color-coded subjects

FR-9.2: System shall export timetables to Excel format with:

- Editable cells
- Filters and sorting

- Multiple sheet support (one per section/faculty)

FR-9.3: System shall allow bulk export of all department timetables

---

6.10 Authentication & Authorization

FR-10.1: System shall implement secure login with email and password

FR-10.2: System shall support role-based access control (RBAC)

FR-10.3: System shall implement JWT-based session management

FR-10.4: System shall enforce password policies (minimum 8 characters, alphanumeric)

FR-10.5: System shall provide password reset functionality

FR-10.6: System shall log all critical actions (timetable generation, modifications, deletions)

---

7. NON-FUNCTIONAL REQUIREMENTS

7.1 Performance

NFR-1.1: Timetable generation shall complete within 2 minutes for 100 sections

NFR-1.2: System shall support 1000+ concurrent users

NFR-1.3: Page load time shall not exceed 3 seconds

NFR-1.4: API response time shall be under 500ms for 95% of requests

NFR-1.5: Database queries shall be optimized with proper indexing

---

7.2 Scalability

NFR-2.1: System architecture shall support horizontal scaling

NFR-2.2: Database shall handle 500+ sections without performance degradation

NFR-2.3: System shall support adding new departments/branches without code changes

---

7.3 Security

NFR-3.1: All API endpoints shall be authenticated

NFR-3.2: Passwords shall be hashed using bcrypt

NFR-3.3: System shall prevent SQL injection and XSS attacks

NFR-3.4: System shall implement CORS policies

NFR-3.5: Sensitive data shall be encrypted at rest

NFR-3.6: System shall maintain audit logs for 90 days

7.4 Usability

NFR-4.1: Interface shall be responsive (desktop, tablet, mobile)

NFR-4.2: System shall provide intuitive navigation

NFR-4.3: Error messages shall be user-friendly and actionable

NFR-4.4: System shall provide tooltips and help documentation

---

7.5 Reliability

NFR-5.1: System uptime shall be 99.5% or higher

NFR-5.2: System shall implement automated database backups (daily)

NFR-5.3: System shall handle failures gracefully with proper error handling

---

7.6 Maintainability

NFR-6.1: Code shall follow MERN best practices

NFR-6.2: System shall include comprehensive API documentation

NFR-6.3: Code shall be modular and follow separation of concerns

NFR-6.4: System shall include unit tests (minimum 70% coverage)

---

8. SYSTEM WORKFLOW

8.1 Initial Setup Workflow

1. Admin logs into the system

2. Admin configures system parameters (working days, slots, durations)
3. Admin adds departments
4. Admin adds branches under departments
5. Admin adds rooms and labs
6. Admin adds faculty with department affiliations
7. Admin creates subjects with weekly hours
8. Admin creates sections and assigns subjects

8.2 Timetable Generation Workflow

1. Admin selects department/branch/semester for timetable generation
2. System validates all required data is present (faculty, subjects, rooms, sections)
3. Admin clicks "Generate Timetable"
4. System runs scheduling algorithm:
    - Allocates subjects to time slots
    - Assigns faculty to subjects
    - Assigns rooms to sessions
    - Checks constraints and clashes
5. System displays generated timetable with clash report
6. If clashes exist:
    - Admin reviews clash details
    - Admin either:
        - Manually resolves clashes, OR
        - Regenerates with different parameters
7. Once clash-free, Admin reviews workload distribution
8. Admin approves and publishes timetable
9. System notifies all affected users (Faculty, HOD, Students)

8.3 User Access Workflow

Faculty:

1. Faculty logs in
2. Views personal timetable dashboard
3. Checks weekly schedule, rooms, sections
4. Exports personal timetable if needed

HOD:

1. HOD logs in
2. Selects department
3. Views department-wise timetables
4. Checks faculty workload reports
5. Exports department schedules

Student:

1. Student logs in
2. Selects semester and section
3. Views section timetable
4. Notes room assignments and timings
5. Exports timetable for offline reference

---

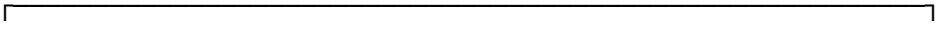
9. HIGH-LEVEL ARCHITECTURE

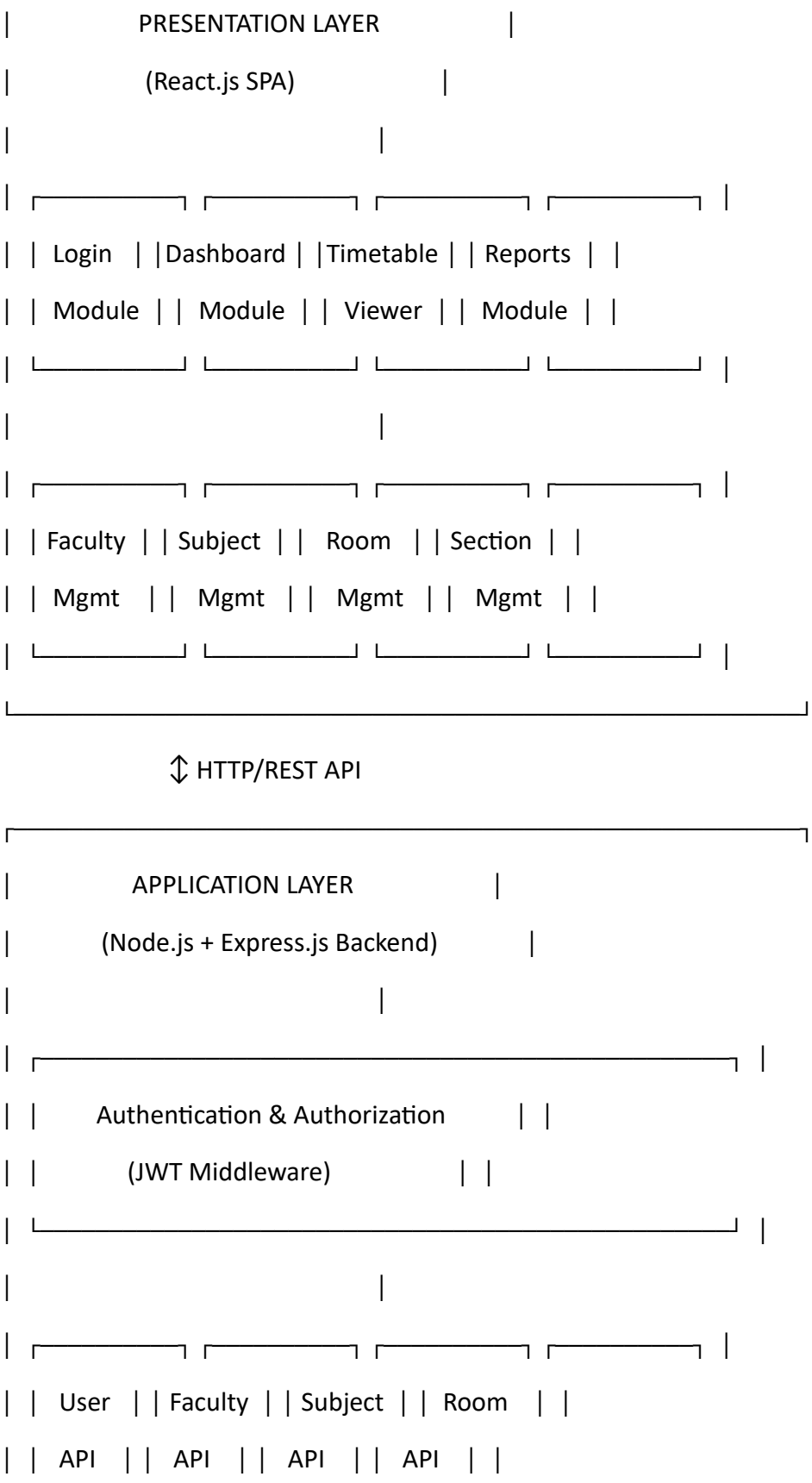
9.1 Architecture Pattern

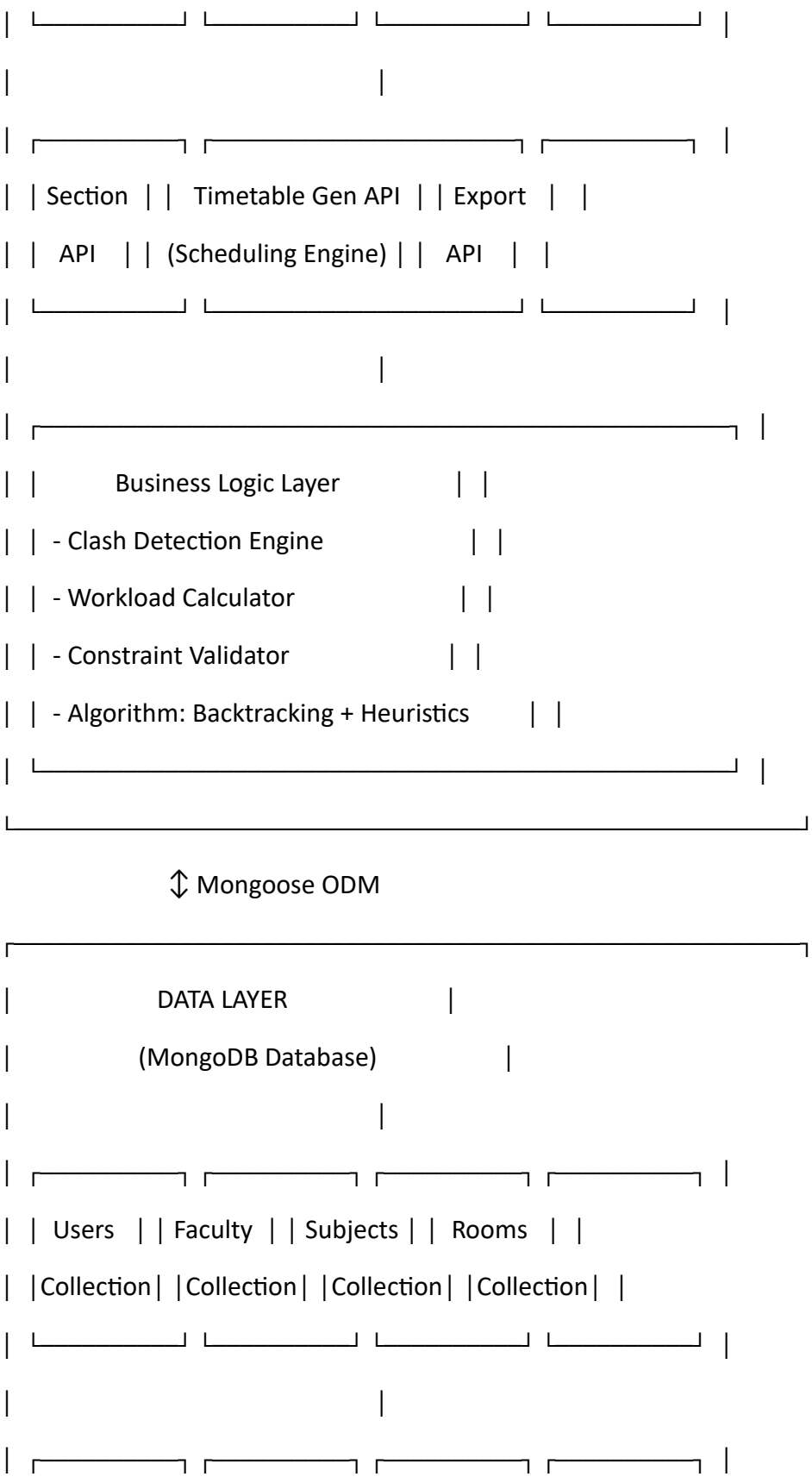
Three-Tier Architecture:

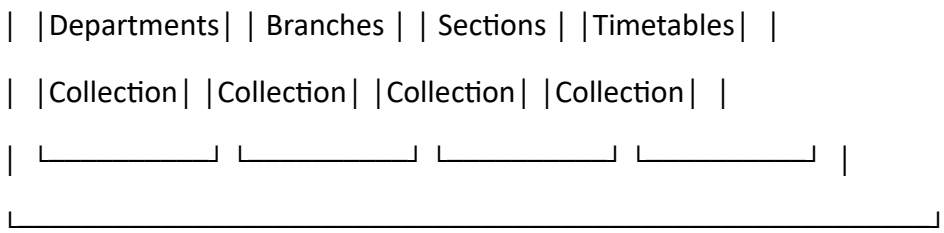
- **Presentation Layer:** React.js frontend
- **Application Layer:** Node.js + Express.js REST APIs
- **Data Layer:** MongoDB database

9.2 Component Diagram




PRESENTATION LAYER
(React.js SPA)
Login Module
Dashboard Module
Timetable Viewer
Reports Module
Faculty Mgmt
Subject Mgmt
Room Mgmt
Section Mgmt
HTTP/REST API
APPLICATION LAYER
(Node.js + Express.js Backend)
Authentication & Authorization
(JWT Middleware)
User API
Faculty API
Subject API
Room API


Section API
Timetable Gen API (Scheduling Engine)
Export API
Business Logic Layer
- Clash Detection Engine
- Workload Calculator
- Constraint Validator
- Algorithm: Backtracking + Heuristics
Mongoose ODM
DATA LAYER
(MongoDB Database)
Users Collection
Faculty Collection
Subjects Collection
Rooms Collection



9.3 Technology Stack Details

Frontend:

- React.js 18+ (UI framework)
- React Router (Navigation)
- Redux Toolkit (State management for large scale)
- Axios (HTTP client)
- Material-UI / Ant Design (UI components)
- jsPDF / ExcelJS (Export functionality)
- React-Bootstrap / Tailwind CSS (Styling)

Backend:

- Node.js 18+ (Runtime)
- Express.js 4+ (Web framework)
- Mongoose (MongoDB ODM)
- JWT (Authentication)
- bcrypt (Password hashing)
- express-validator (Input validation)
- winston (Logging)

Database:

- MongoDB 6+ (NoSQL database)
- MongoDB Atlas (Cloud hosting recommended for production)

DevOps & Deployment:

- Git (Version control)

- Docker (Containerization)
- AWS/Azure/Heroku (Hosting)
- Nginx (Reverse proxy)
- PM2 (Process manager)

---

10. DATABASE ENTITIES (Tables/Collections)

10.1 Users Collection

javascript

```
{
  _id: ObjectId,
  userId: String (unique),
  name: String,
  email: String (unique),
  password: String (hashed),
  role: String (enum: ['admin', 'hod', 'faculty', 'student']),
  department: ObjectId (ref: Department),
  facultyId: ObjectId (ref: Faculty), // if role is faculty
  section: ObjectId (ref: Section), // if role is student
  createdAt: Date,
  updatedAt: Date,
  isActive: Boolean
}
```

Indexes: userId, email, role

10.2 Departments Collection

javascript

{

_id: ObjectId,

deptCode: String (unique),

deptName: String,

hod: ObjectId (ref: Faculty),

description: String,

createdAt: Date,

updatedAt: Date

}

Indexes: deptCode

10.3 Branches Collection

javascript

{

_id: ObjectId,

branchCode: String (unique),

branchName: String,

department: ObjectId (ref: Department),

duration: Number, // in semesters

description: String,

createdAt: Date,

updatedAt: Date

}

Indexes: branchCode, department

10.4 Faculty Collection

javascript

```
{
  _id: ObjectId,
  facultyId: String (unique),
  name: String,
  email: String,
  phone: String,
  departments: [ObjectId] (ref: Department), // multi-dept support
  specialization: [String],
  eligibleSubjects: [ObjectId] (ref: Subject),
  maxWeeklyHours: Number,
  preferredSlots: [
    {
      day: String,
      slotNumber: Number
    }
  ],
  createdAt: Date,
  updatedAt: Date,
  isActive: Boolean
}
```

Indexes: facultyId, departments

10.5 Subjects Collection

javascript

```
{
```

```
  _id: ObjectId,
  subjectCode: String (unique),
  subjectName: String,
  department: ObjectId (ref: Department),
  branch: ObjectId (ref: Branch),
  semester: Number,
  subjectType: String (enum: ['theory', 'lab', 'practical']),
  weeklyHours: Number,
  sessionDuration: Number, // in minutes (50/60 for theory, 100+ for lab)
  eligibleFaculty: [ObjectId] (ref: Faculty),
  isElective: Boolean,
  createdAt: Date,
  updatedAt: Date
}
```

Indexes: subjectCode, department, branch, semester

10.6 Rooms Collection

javascript

```
{
  _id: ObjectId,
  roomNumber: String (unique),
  roomType: String (enum: ['classroom', 'lab', 'auditorium']),
  capacity: Number,
  building: String,
  floor: String,
  department: ObjectId (ref: Department), // null if shared
```

facilities: [String], // e.g., ['projector', 'ac', 'computers']

createdAt: Date,

updatedAt: Date,

isActive: Boolean

}

Indexes: roomNumber, roomType, department

10.7 Sections Collection

javascript

{

_id: ObjectId,

sectionCode: String (unique), // e.g., 'CSE-3A'

branch: ObjectId (ref: Branch),

semester: Number,

studentStrength: Number,

subjects: [ObjectId] (ref: Subject),

academicYear: String, // e.g., '2024-2025'

createdAt: Date,

updatedAt: Date

}

Indexes: sectionCode, branch, semester

10.8 Timetables Collection

javascript

{

_id: ObjectId,

timetableName: String,

department: ObjectId (ref: Department),

branch: ObjectId (ref: Branch),

semester: Number,

academicYear: String,

generatedBy: ObjectId (ref: User),

generatedAt: Date,

status: String (enum: ['draft', 'published', 'archived']),

schedule: [

{

section: ObjectId (ref: Section),

day: String, // 'Monday', 'Tuesday', etc.

slotNumber: Number, // 1, 2, 3, ...

subject: ObjectId (ref: Subject),

faculty: ObjectId (ref: Faculty),

room: ObjectId (ref: Room),

startTime: String, // '09:00 AM'

endTime: String, // '09:50 AM'

duration: Number, // in minutes

isLabSession: Boolean

}

],

clashes: [

{

```
    clashType: String (enum: ['faculty', 'room', 'section']),
    description: String,
    affectedEntities: [ObjectId],
    day: String,
    slotNumber: Number,
    isResolved: Boolean
  }
 ],

 version: Number,
 publishedAt: Date,
 createdAt: Date,
 updatedAt: Date
}
```

Indexes: department, branch, semester, academicYear, status

10.9 SystemConfig Collection

```
javascript
{
 _id: ObjectId,
 configKey: String (unique),
 configValue: Mixed,
 description: String,

 // Example configs:
 // {configKey: 'workingDays', configValue: ['Monday', 'Tuesday', ...]}
```

```
  // {configKey: 'slotsPerDay', configValue: 8}
  // {configKey: 'slotDuration', configValue: 50}
  // {configKey: 'breakSlots', configValue: [4, 7]}
  // {configKey: 'lunchSlot', configValue: 5}

  createdAt: Date,
  updatedAt: Date
}
```

Indexes: configKey

10.10 AuditLogs Collection

```
javascript
{
  _id: ObjectId,
  userId: ObjectId (ref: User),
  action: String, // 'create', 'update', 'delete', 'generate_timetable'
  entityType: String, // 'faculty', 'subject', 'timetable', etc.
  entityId: ObjectId,
  changes: Object, // before and after values
  ipAddress: String,
  timestamp: Date
}
```

Indexes: userId, entityType, timestamp

10.11 Database Relationships

```
Department (1) ——— (N) Branches
Department (1) ——— (N) Faculty
Department (1) ——— (N) Subjects
Branch (1) ——— (N) Sections
Branch (1) ——— (N) Subjects
Section (N) ——— (N) Subjects
Subject (N) ——— (N) Faculty
Faculty (N) ——— (N) Departments
Timetable (N) ——— (1) Section
Timetable (N) ——— (1) Subject
Timetable (N) ——— (1) Faculty
Timetable (N) ——— (1) Room
```

---

11. TIMETABLE GENERATION LOGIC

11.1 Algorithm Overview

The system uses a **Constraint Satisfaction Problem (CSP) approach** with **Backtracking** and **Heuristic Optimizations**. This is practical and implementable within 3 weeks.

Algorithm Type: Backtracking with Forward Checking and Heuristics

Time Complexity: O(b^d) where b is branching factor, d is depth

- With heuristics and pruning, practical for 100+ sections

11.2 Step-by-Step Timetable Generation Logic

Step 1: Data Collection and Validation

```
Input:
- Selected department/branch/semester
- Sections to schedule
- Subjects per section with weekly hours
- Available faculty
- Available rooms
- System configuration (days, slots, durations)

Validation:
1. Check all sections have assigned subjects
2. Check all subjects have eligible faculty
3. Check sufficient rooms are available
4. Check faculty total load doesn't exceed system capacity
5. Verify weekly hours are achievable within available slots
```

Step 2: Initialize Data Structures

javascript

```
// Weekly timetable grid
timetableGrid = {
  section_id: {
    'Monday': [slot1, slot2, slot3, ...],
    'Tuesday': [...],
    ...
  }
}

// Faculty schedule tracker
facultySchedule = {
  faculty_id: {
    'Monday': [slot1, slot2, ...],
    totalWeeklyHours: 0
  }
}

// Room availability tracker
roomAvailability = {
  room_id: {
    'Monday': [available_slots],
    ...
  }
}
```

// Subject hours remaining

subjectHoursRemaining = {

subject_id: weeklyHoursRequired

}

Step 3: Prioritize Subjects (Heuristic)

Priority Order:

1. Lab subjects (need consecutive slots)
2. Subjects with fewer eligible faculty (constrained)
3. Subjects with higher weekly hours
4. Theory subjects

javascript

subjects.sort((a, b) => {

if (a.isLab && !b.isLab) return -1;

if (a.eligibleFaculty.length < b.eligibleFaculty.length) return -1;

if (a.weeklyHours > b.weeklyHours) return -1;

return 0;

});

```

#### **Step 4: Slot Allocation Loop**

```

For each section:

For each subject in priority order:

hoursToSchedule = subject.weeklyHours

```
While hoursToSchedule > 0:

  // Step 4a: Find available slot
  For each day in workingDays:
    For each slot in availableSlots:

      // Skip break/lunch slots
      if (slot is break/lunch) continue;

      // Check if slot is available for section
      if (section has class at this slot) continue;

      // For labs, check if next slot is also available
      if (subject.isLab):
        if (next slot not available) continue;

      // Step 4b: Find available faculty
      availableFaculty = []
      For each faculty in subject.eligibleFaculty:
        if (faculty is free at this slot):
          if (faculty.weeklyHours < faculty.maxWeeklyHours):
            availableFaculty.push(faculty)

      if (availableFaculty is empty):
        continue; // No faculty available
```

```
// Select faculty with minimum current workload (load balancing)
selectedFaculty = facultyWithMinWorkload(availableFaculty)

// Step 4c: Find available room
availableRooms = []
For each room in rooms:
  if (room.capacity >= section.studentStrength):
    if (subject.isLab and room.type == 'lab') OR
       (subject.isTheory and room.type == 'classroom'):
      if (room is free at this slot):
        availableRooms.push(room)

if (availableRooms is empty):
  continue; // No room available

// Select room (prefer department-specific rooms)
selectedRoom = selectOptimalRoom(availableRooms, section.department)

// Step 4d: Assign to timetable
if (subject.isLab):
  // Assign 2 consecutive slots
  assignSlot(section, day, slot, subject, faculty, room)
  assignSlot(section, day, slot+1, subject, faculty, room)
  markFacultyBusy(faculty, day, slot, slot+1)
  markRoomBusy(room, day, slot, slot+1)
  hoursToSchedule -= subject.sessionDuration
```

```
        else:
          // Assign single slot
          assignSlot(section, day, slot, subject, faculty, room)
          markFacultyBusy(faculty, day, slot)
          markRoomBusy(room, day, slot)
          hoursToSchedule -= subject.sessionDuration

        break; // Move to next hour requirement

    // If no slot found after trying all days/slots
    if (hoursToSchedule not reduced):
      // Backtrack or report infeasibility
      return GENERATE_ALTERNATIVE or REPORT_CONSTRAINT_FAILURE
```

Step 5: Clash Detection

After generation, run comprehensive clash detection:

javascript

```
clashes = []

// Faculty clash detection
For each faculty:
  For each day:
    For each slot:
      assignments = getAssignmentsForFacultyAtSlot(faculty, day, slot)
      if (assignments.length > 1):
        clashes.push({
          type: 'faculty',
```

```
        faculty: faculty,
        day: day,
        slot: slot,
        conflictingAssignments: assignments
      })

// Room clash detection
For each room:
  For each day:
    For each slot:
      assignments = getAssignmentsForRoomAtSlot(room, day, slot)
      if (assignments.length > 1):
        clashes.push({
          type: 'room',
          room: room,
          day: day,
          slot: slot,
          conflictingAssignments: assignments
        })

// Section clash detection
For each section:
  For each day:
    For each slot:
      assignments = getAssignmentsForSectionAtSlot(section, day, slot)
      if (assignments.length > 1):
```

```
        clashes.push({
          type: 'section',
          section: section,
          day: day,
          slot: slot,
          conflictingAssignments: assignments
        })

// Check faculty workload limits
For each faculty:
  totalHours = calculateTotalWeeklyHours(faculty)
  if (totalHours > faculty.maxWeeklyHours):
    clashes.push({
      type: 'workload',
      faculty: faculty,
      totalHours: totalHours,
      maxAllowed: faculty.maxWeeklyHours
    })

return clashes
```

Step 6: Generate Alternative Timetables

If user requests alternatives:

```
javascript
alternatives = []

For iteration = 1 to maxAlternatives (e.g., 3):
```

```
  // Change heuristics slightly
  - Randomize faculty selection among equally loaded faculty
  - Try different slot starting points
  - Shuffle subject priority within same category

  generatedTimetable = generateTimetable()

  if (generatedTimetable has fewer clashes than previous):
    alternatives.push(generatedTimetable)

return alternatives
```

11.3 Optimization Techniques

1. **Heuristic Ordering:** Schedule constrained subjects first
2. **Load Balancing:** Distribute faculty workload evenly
3. **Greedy Room Assignment:** Prefer department-specific rooms
4. **Early Clash Detection:** Check constraints before assignment
5. **Caching:** Store faculty and room availability in memory
6. **Indexing:** Use MongoDB indexes for fast lookups

11.4 Edge Cases Handling

1. **Insufficient Faculty:** Alert admin to assign more faculty
2. **Insufficient Rooms:** Suggest sharing rooms across departments
3. **Unsatisfiable Constraints:** Provide detailed report of conflicts
4. **Lab Scheduling at Day End:** Don't schedule labs in last slot if they need 2 slots
5. **Faculty Preference:** Honor preferred slots when possible (soft constraint)

---

12. FLOWCHART DESCRIPTION

12.1 Main Timetable Generation Flowchart

```
START
  ↓
[Admin Login]
  ↓
[Select Department/Branch/Semester]
  ↓
[Click "Generate Timetable"]
  ↓
[System: Fetch Sections, Subjects, Faculty, Rooms]
  ↓
[Validate Input Data]
  ↓
```

{Is Data Valid?} —No—→ [Display Error: Missing Data] → END

↓ Yes

[Initialize Data Structures]

↓

[Sort Subjects by Priority]

↓

[FOR each Section]

↓

[FOR each Subject]

↓

[Calculate Hours to Schedule]

↓

[WHILE Hours Remaining > 0]

↓

[Try to Find Available Slot]

↓

{Slot Found?} —No—→ [Try Next Slot/Day]

↓ Yes

[Find Available Faculty]

↓

{Faculty Found?} —No—→ [Try Next Slot]

↓ Yes

[Find Available Room]

↓

{Room Found?} —No—→ [Try Next Slot]

↓ Yes

[Assign Slot to Timetable]

↓

[Update Faculty/Room Availability]

↓

[Reduce Hours Remaining]

↓

[Next Subject]

↓

[Next Section]

↓

[Run Clash Detection]

↓

{Clashes Found?} —Yes—→ [Display Clash Report]

↓ ↓

No [Admin: Resolve or Regenerate?]

↓ ↓

[Calculate Faculty Workload] [Regenerate] → [Back to Initialize]

↓

[Display Generated Timetable]

↓

[Admin Reviews Timetable]

↓

{Approve?} —No—→ [Make Manual Adjustments] → [Save as Draft]

↓ Yes

[Publish Timetable]

↓

[Notify Users (Faculty, HOD, Students)]

↓

END

```

---

### **12.2 Clash Detection Flowchart**

```

START: Clash Detection

↓

[Initialize Clash Array = []]

↓

[FOR each Faculty]

↓

[FOR each Day]

↓

[FOR each Slot]

↓

[Count Assignments at this Slot]

↓

{Count > 1?} —Yes—→ [Add Faculty Clash to Array]

↓ No

[Next Slot]

[Next Day]

[Next Faculty]

↓

[FOR each Room]

↓

[Similar nested loop for Room clashes]

↓

[FOR each Section]

↓

[Similar nested loop for Section clashes]

↓

[Check Faculty Workload Limits]

↓

{Any faculty exceeds max hours?} —Yes—→ [Add Workload Clash]

↓ No

[Return Clash Array]

↓

{Clash Array Empty?} —Yes—→ [Timetable is Valid]

↓ No

[Display Clashes to Admin]

↓

END

```

---

### **12.3 User Login and Access Flowchart**

```

START

↓

[User Opens Application]

↓

[Enter Email and Password]

↓

[Click Login]

↓

[Backend: Verify Credentials]

↓

{Valid Credentials?} —No—→ [Display Error: Invalid Login] → [Retry]

↓ Yes

[Generate JWT Token]

↓

[Send Token to Frontend]

↓

[Store Token in LocalStorage]

↓

[Redirect Based on Role]

↓

{Role?}

├– Admin —→ [Admin Dashboard]

├– HOD —→ [HOD Dashboard]

├– Faculty —→ [Faculty Dashboard]

└– Student —→ [Student Dashboard]

↓

[Display Respective Timetable/Data]

↓

END

```

---

## **13. UI MODULES**

### **13.1 Authentication Pages**

**13.1.1 Login Page**

- Email and password fields
- Remember me checkbox
- Forgot password link
- Login button
- Responsive design

**13.1.2 Forgot Password Page**

- Email input
- Send reset link button
- Back to login link

---

### **13.2 Admin Dashboard**
```

13.2.1 Main Dashboard

- Welcome message with admin name
- Quick statistics cards:
  - Total Departments
  - Total Faculty
  - Total Sections
  - Published Timetables
- Recent activities log
- Quick action buttons (Add Faculty, Generate Timetable)

13.2.2 Sidebar Navigation

- Dashboard
- Faculty Management
- Subject Management
- Department Management
- Branch Management
- Section Management
- Room Management
- Timetable Generation
- Reports
- Settings
- Logout

13.3 Faculty Management Module

13.3.1 Faculty List Page

- Data table with columns:
  - Faculty ID
  - Name
  - Department(s)
  - Specialization
  - Max Weekly Hours
  - Actions (Edit, Delete, View)
- Search and filter functionality
- Add Faculty button
- Export to Excel button

13.3.2 Add/Edit Faculty Form

- Faculty ID (auto-generated or manual)
- Name
- Email
- Phone
- Department selection (multi-select for multi-dept faculty)
- Specialization tags
- Eligible subjects (multi-select)
- Max weekly hours
- Preferred time slots (optional grid)
- Submit and Cancel buttons

13.4 Subject Management Module

13.4.1 Subject List Page

- Data table with columns:
  - Subject Code
  - Subject Name
  - Department
  - Branch
  - Type (Theory/Lab)
  - Weekly Hours
  - Actions
- Filter by department, branch, semester
- Add Subject button

13.4.2 Add/Edit Subject Form

- Subject Code
- Subject Name
- Department dropdown
- Branch dropdown
- Semester number
- Subject Type (Theory/Lab/Practical radio buttons)
- Weekly Hours input
- Session Duration (minutes)
- Eligible Faculty (multi-select)

- Is Elective checkbox

- Submit and Cancel buttons

13.5 Department & Branch Management

13.5.1 Department List

- Card-based layout showing departments

- Each card shows:

  - Department name

  - HOD name

  - Number of branches

  - Edit and Delete icons

13.5.2 Branch List

- Nested under department

- Shows branch name, duration, number of sections

- Add Branch button per department

13.6 Section Management Module

13.6.1 Section List

- Data table with:

  - Section Code
  - Branch
  - Semester
  - Student Strength
  - Subjects Assigned
  - Actions
- Filter by branch and semester
- Add Section button

13.6.2 Add/Edit Section Form
- Section Code
- Branch dropdown
- Semester number
- Student Strength
- Academic Year
- Assign Subjects (multi-select with search)
- Submit and Cancel buttons

13.7 Room Management Module

13.7.1 Room List
- Data table with:
  - Room Number
  - Type (Classroom/Lab)

  - Capacity
  - Building & Floor
  - Department (if allocated)
  - Actions
- Filter by type and department
- Add Room button

13.7.2 Add/Edit Room Form
- Room Number
- Room Type dropdown
- Capacity
- Building name
- Floor
- Department allocation (optional)
- Facilities checklist (Projector, AC, Computers, etc.)
- Submit and Cancel buttons

13.8 Timetable Generation Module

13.8.1 Generate Timetable Page
- **Configuration Section:**
  - Select Department
  - Select Branch
  - Select Semester

  - Academic Year

  - Working Days selection

  - Slots per day

  - Slot duration

  - Break slots configuration

- **Options:**

  - Generate single timetable

  - Generate multiple alternatives (number input)

- **Generate Button** (prominent, primary color)

13.8.2 Timetable Preview Page

- Tabbed interface showing:

  - Section-wise timetables (tabs for each section)

  - Faculty-wise timetables

  - Room-wise timetables

- **Timetable Grid Display:**

```
  | Time/Day | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday |
  |----------|--------|---------|-----------|----------|--------|----------|
  | 9:00-9:50| Sub1   | Sub2    | Sub3      | Sub4     | Sub5   | Sub6     |
  |          | Fac1   | Fac2    | Fac3      | Fac4     | Fac5   | Fac6     |
  |          | Room1  | Room2   | Room3     | Room4    | Room5  | Room6    |
```

- **Color Coding:**
  - Theory classes: Light blue
  - Lab sessions: Light green
  - Break: Light yellow
  - Lunch: Light orange

- **Clash Indicator:**
  - Red border on cells with clashes
  - Clash count badge

13.8.3 Clash Report Panel

- Expandable panel showing:
  - Clash Type (Faculty/Room/Section/Workload)
  - Details (affected entities, time slot)
  - Suggested resolution
  - Resolve button (allows manual edit)

13.8.4 Faculty Workload Dashboard

- Faculty list with workload bars
- Visual representation (progress bars)
- Color coding:
  - Green: Under 70% capacity
  - Yellow: 70-90% capacity
  - Red: Over 90% capacity
- Export workload report button

13.8.5 Action Buttons

- Save as Draft
- Regenerate
- Publish Timetable
- Export (PDF/Excel dropdown)

13.9 HOD Dashboard

13.9.1 Department Overview

- Department statistics
- Faculty workload summary
- Timetable status (Draft/Published)

13.9.2 View Department Timetables

- Branch and semester selector
- Timetable grid display (similar to admin view)
- Faculty workload view
- Export functionality

13.10 Faculty Dashboard

13.10.1 Personal Timetable View

- Weekly schedule grid showing:
  - Day and time
  - Subject
  - Section
  - Room
- Current week highlighted
- Print and Export buttons

13.10.2 Workload Summary

- Total weekly hours
- Subject-wise distribution (pie chart)
- Daily distribution (bar chart)

13.11 Student Dashboard

13.11.1 Section Timetable View

- Section selector (if student is in multiple sections/electives)
- Weekly timetable grid
- Room locations
- Faculty names
- Export to PDF button

13.12 Reports Module

13.12.1 Report Types

- Faculty Workload Report (all faculty)
- Room Utilization Report
- Department-wise Timetables
- Clash History Report

13.12.2 Report Filters

- Date range
- Department
- Semester
- Export format (PDF/Excel)

13.13 Settings Module

13.13.1 System Configuration

- Working days configuration
- Slots per day
- Default slot duration
- Break and lunch slot configuration
- Academic year settings

13.13.2 User Management

- Add/Edit/Delete users
- Assign roles
- Reset passwords

---

14. FUTURE ENHANCEMENTS

1. **AI-Powered Optimization**
   - Machine learning to learn from past successful timetables
   - Predictive analytics for resource utilization
   - Smart suggestions based on historical data

2. **Mobile Applications**
   - Native iOS and Android apps
   - Push notifications for timetable updates
   - Offline access to timetables

3. **Advanced Analytics**
   - Room utilization heatmaps
   - Faculty workload trends
   - Student attendance correlation

4. **Integration Features**
   - LMS integration (Moodle, Canvas)

- Google Calendar / Outlook sync
- Exam scheduling module
- Attendance system integration

5. **Collaborative Features**
   - Faculty swap requests
   - Room booking beyond timetable
   - Student feedback on timings

6. **Smart Notifications**
   - Email/SMS alerts for timetable changes
   - Reminder notifications before classes
   - Faculty substitution alerts

7. **Advanced Constraints**
   - Faculty preference weightage
   - Student elective preferences
   - Interdisciplinary course scheduling

8. **Multi-Campus Support**
   - Manage multiple campuses from single interface
   - Cross-campus faculty scheduling

9. **Audit and Compliance**
   - Regulatory compliance reports
   - Accreditation documentation

- Historical timetable archives

10. **Performance Enhancements**
    - GraphQL API for efficient data fetching
    - Server-side caching (Redis)
    - WebSocket for real-time updates

---

15. ESTIMATED DEVELOPMENT TIMELINE (3 Weeks)

Week 1: Backend Foundation & Core Entities

Days 1-2: Project Setup & Database
- Initialize Node.js + Express project
- Setup MongoDB connection
- Create database schemas (Users, Departments, Branches, Faculty, Subjects, Rooms, Sections)
- Implement Mongoose models with validation
- Setup environment configuration

Days 3-4: Authentication & Basic APIs
- Implement JWT authentication
- Create login/logout APIs
- Password hashing with bcrypt
- Role-based middleware
- Create CRUD APIs for:

- Faculty
- Subjects
- Departments & Branches
- Rooms
- Sections

Days 5-7: Timetable Generation Core Logic
- Implement scheduling algorithm (CSP with backtracking)
- Build clash detection engine
- Create timetable generation API
- Implement faculty workload calculator
- Test algorithm with sample data
- Optimize for performance

Week 2: Frontend Development & Integration

Days 8-9: Frontend Setup & Authentication
- Initialize React project with Vite/CRA
- Setup React Router
- Setup Redux Toolkit for state management
- Implement login page
- Implement protected routes
- Connect to backend APIs

Days 10-11: Admin Module - Data Management

- Create Admin Dashboard
- Implement Faculty Management UI (List, Add, Edit, Delete)
- Implement Subject Management UI
- Implement Department & Branch Management UI
- Implement Room Management UI
- Implement Section Management UI
- Connect all forms to backend APIs
- Add validation and error handling

Days 12-13: Timetable Generation UI

- Create Timetable Generation configuration page
- Implement timetable grid display component
- Build clash report component
- Create faculty workload dashboard
- Integrate with backend generation API
- Add loading states and error handling

Day 14: Export Functionality

- Implement PDF export using jsPDF
- Implement Excel export using ExcelJS
- Add export buttons to timetable views
- Test export formats

Week 3: User Dashboards, Testing & Deployment

Days 15-16: User Role Dashboards

- Implement HOD Dashboard
- Implement Faculty Dashboard (personal timetable view)
- Implement Student Dashboard
- Add workload visualization charts
- Ensure role-based content rendering

Day 17: UI/UX Polish

- Responsive design implementation
- Add loading spinners and skeleton screens
- Improve error messages and user feedback
- Add tooltips and help text
- Color coding for timetable cells
- UI consistency check

Day 18: Testing

- Unit testing for critical backend functions (clash detection, scheduling)
- API testing with Postman/Thunder Client
- Frontend component testing
- End-to-end user flow testing
- Performance testing with large datasets (500 sections)
- Fix identified bugs

Day 19: Deployment Preparation

- Environment configuration for production
- Setup MongoDB Atlas for cloud database
- Create Docker containers (optional)
- Setup CI/CD pipeline (if using)
- Prepare deployment scripts

Day 20: Deployment & Documentation
- Deploy backend to cloud (Heroku/AWS/Azure)
- Deploy frontend (Netlify/Vercel)
- Setup reverse proxy (Nginx) if needed
- Configure CORS and security headers
- Create user documentation (Admin Guide, User Guide)
- Create API documentation
- Record demo video

Day 21: Buffer & Final Review
- Address any deployment issues
- Final testing in production environment
- Performance monitoring setup
- Backup strategy implementation
- Handover documentation
- Prepare for presentation/demo

Development Milestones Summary

| Milestone | Completion Date | Deliverables |

|----------|----------------|-------------|

| Backend Core | End of Week 1 | Database, APIs, Scheduling Algorithm |

| Frontend Core | End of Week 2 | Admin UI, Timetable Generation UI |

| Complete System | Day 20 | Deployed Production System |

| Final Review | Day 21 | Documentation, Demo Ready |

Risk Mitigation

1. **Algorithm Complexity:** Use proven backtracking approach; avoid over-optimization initially

2. **Time Constraints:** Focus on MVP features; defer advanced analytics to future

3. **Testing Time:** Continuous testing throughout; don't leave to last days

4. **Deployment Issues:** Test deployment process in Week 2

5. **Scope Creep:** Strictly adhere to defined requirements; note additional requests as future enhancements

---

APPENDIX

A. Technology Resources

Learning Resources:

- MongoDB University (free courses)

- Node.js Documentation

- React Official Docs

- Redux Toolkit Tutorial

Libraries to Use:

- Backend: express, mongoose, jsonwebtoken, bcrypt, express-validator, cors, dotenv

- Frontend: react, react-router-dom, @reduxjs/toolkit, axios, react-query, material-ui or antd

- Export: jspdf, exceljs

- Charts: recharts or chart.js

B. Sample Environment Variables

```
# Backend .env

NODE_ENV=production

PORT=5000

MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/timetable_db

JWT_SECRET=your_super_secret_jwt_key_change_this

JWT_EXPIRE=7d

BCRYPT_ROUNDS=10


# Frontend .env

REACT_APP_API_URL=https://your-backend-url.com/api
```

C. Sample API Endpoints

```

Authentication:

POST /api/auth/login

POST /api/auth/logout

POST /api/auth/forgot-password

POST /api/auth/reset-password

Faculty:

GET /api/faculty

GET /api/faculty/:id

POST /api/faculty

PUT /api/faculty/:id

DELETE /api/faculty/:id

Subjects:

GET /api/subjects

POST /api/subjects

PUT /api/subjects/:id

DELETE /api/subjects/:id

Timetables:

POST /api/timetables/generate

GET /api/timetables/:id

GET /api/timetables/department/:deptId

PUT /api/timetables/:id/publish

GET /api/timetables/:id/export/pdf
```

GET /api/timetables/:id/export/excel

Workload:

GET /api/faculty/:id/workload

GET /api/reports/faculty-workload

CONCLUSION

This PRD provides a comprehensive blueprint for building a production-ready Smart Classroom & Timetable Scheduler using the MERN stack within a 3-week timeline. The system is designed to scale to 20+ departments, 100+ branches, and 500+ sections while maintaining performance and user experience.

Key Success Factors:

1. Clear requirement definition and scope management
2. Practical algorithm implementation (CSP with backtracking)
3. Modular architecture for maintainability
4. Phased development approach
5. Continuous testing and iteration
6. Focus on core features before enhancements

The proposed solution addresses all stated requirements from the Smart India Hackathon problem statement and provides a solid foundation for future enhancements. With disciplined execution following this PRD, the team can deliver a functional, production-ready system within the 3-week timeline.

Next Steps:

1. Review and approve this PRD
2. Setup development environment
3. Begin Week 1 tasks immediately
4. Daily standup meetings to track progress
5. Weekly milestone reviews

Good luck with your project! 🚀