College Management System - Complete API Documentation

Table of Contents

1. Authentication Journey
2. Admin Journey
3. Teacher User Journey
4. Student User Journey
5. API Requirements
6. Screen List + UI Planning
7. System Assumptions
8. Screen-Wise Functionalities
9. Overall Solutioning
10. Implementation Approach
11. Database Schema Design

Authentication Journey

1. Login Page

What user sees:

- Email field
- Password field
- Login button

API called when user clicks "Login":

> POST /auth/login

HTTP Request:

```
POST /auth/login
Content-Type: application/json

{
  "email": "admin@college.com",
  "password": "Admin@123"
}
```

Backend flow:

1. Reads USERS
2. Verifies password
3. Resolves role
4. Generates JWT

HTTP Response:

```
{
  "access_token": "jwt_token",
  "role": "ADMIN",
  "user": {
    "id": "admin-uuid",
    "first_name": "System",
    "last_name": "Admin"
  }
}
```

Browser stores token → redirects to Admin Dashboard

2. Forgot Password Page

What user sees:

- Email input
- Submit button

API on submit:

> POST /auth/forgot-password

Request:

```
{
  "email": "student@college.com"
}
```

Response:

```
{
  "message": "Reset link sent to email"
}
```

3. Reset / Change Password Page

API:

> POST /auth/reset-password

Request:

```
{
  "token": "reset-token",
  "new_password": "New@123"
}
```

4. Logout

What user sees:

- Logout button in navigation

API called when user clicks "Logout":

> POST /auth/logout

HTTP Request:

```
POST /auth/logout
Authorization: Bearer jwt_token
```

HTTP Response:

```
{
  "message": "Logged out successfully"
}
```

Browser clears token → redirects to Login Page

Admin Journey

1. Admin Dashboard

What admin sees:

- Cards: Students count, Teachers count, Sections count, Pending leaves, Active exams
- Recent activity

- Navigation menu: Student, Teacher and Academic Structure Management Page, Subject Management, Teacher Assignment, Timetable, Exams, Notifications, Results Review Page
- Chatbot widget

APIs called on page load:

1. Dashboard stats

> GET /admin/dashboard

HTTP Request:

```
GET /admin/dashboard
Authorization: Bearer jwt
```

Response:

```
{
  "students": 1200,
  "teachers": 75,
  "sections": 30,
  "pending_leaves": 18,
  "active_exams": 4
}
```

2. Activity feed

> GET /admin/dashboard/activity

Response:

```
{
  "activities": [
    "Teacher Rahul submitted marks",
```

```
      "Student Riya applied for leave"
    ]
  }
```

Chatbot

> POST /chat/query

```
{
  "message": "Show branch-wise performance"
}
```

2. Student Management Page

What admin sees:

- List of students
- Add button
- Import CSV
- Edit / deactivate buttons

API on page load:

> GET /admin/students

Response:

```
{
  "students": [
    {
      "id": "stu-1",
      "name": "Amit Sharma",
      "section": "CSE-A",
      "is_active": true
    }
  ]
```

```
}
```

Add student:

> POST /admin/students

```
{
  "email": "new@college.com",
  "first_name": "Riya",
  "last_name": "Patel",
  "branch_id": "cse-id",
  "section_id": "sec-a-id"
}
```

Edit student:

> PUT /admin/students/{studentId}

HTTP Request:

```
PUT /admin/students/stu-1
Authorization: Bearer jwt
Content-Type: application/json

{
  "first_name": "Amit",
  "last_name": "Sharma",
  "email": "amit.updated@college.com",
  "branch_id": "cse-id",
  "section_id": "sec-a-id",
  "is_active": true
}
```

HTTP Response:

```
{
  "message": "Student updated successfully",
```

```
  "student": {
    "id": "stu-1",
    "first_name": "Amit",
    "last_name": "Sharma",
    "email": "amit.updated@college.com"
  }
}
```

Import CSV:

> POST /admin/students/import

Response:

```
{
  "total_rows": 100,
  "inserted": 95,
  "failed": 5
}
```

3. Teacher Management Page

API on page load:

> GET /admin/teachers

Response:

```
{
  "teachers": [
    {
      "id": "t-1",
      "name": "Rahul Mehta",
      "department": "CSE"
    }
  ]
}
```

Add teacher:

> POST /admin/teachers

```
{
  "email": "teacher@college.com",
  "first_name": "Rahul",
  "last_name": "Mehta",
  "department": "CSE"
}
```

Edit teacher:

> PUT /admin/teachers/{teacherId}

HTTP Request:

```
PUT /admin/teachers/t-1
Authorization: Bearer jwt
Content-Type: application/json

{
  "first_name": "Rahul",
  "last_name": "Mehta",
  "email": "rahul.updated@college.com",
  "department": "CSE",
  "is_active": true
}
```

HTTP Response:

```
{
  "message": "Teacher updated successfully",
  "teacher": {
    "id": "t-1",
    "first_name": "Rahul",
    "last_name": "Mehta",
    "email": "rahul.updated@college.com"
  }
```

```
}
```

4. User Profile (Read-only)

API:

> GET /admin/students/{studentId}

Response:

```
{
  "id": "stu-1",
  "name": "Amit Sharma",
  "dob": "2003-01-10",
  "section": "CSE-A"
}
```

5. Academic Structure Page

Page load APIs:

```
GET /admin/branches
GET /admin/semesters
GET /admin/sections
```

Update branch:

> PATCH /admin/branches/{branchId}

HTTP Request:

```
PATCH /admin/branches/cse-id
Authorization: Bearer jwt
```

```
Content-Type: application/json

{
  "branch_name": "Computer Science & Engineering",
  "is_active": true
}
```

HTTP Response:

```
{
  "message": "Branch updated successfully",
  "branch": {
    "id": "cse-id",
    "branch_name": "Computer Science & Engineering",
    "is_active": true
  }
}
```

Update semester:

> PATCH /admin/semesters/{semesterId}

HTTP Request:

```
PATCH /admin/semesters/sem-6
Authorization: Bearer jwt
Content-Type: application/json

{
  "semester_name": "Semester 6",
  "is_active": true
}
```

HTTP Response:

```
{
  "message": "Semester updated successfully",
  "semester": {
```

```
    "id": "sem-6",
    "semester_name": "Semester 6",
    "is_active": true
  }
}
```

Create branch:

> POST /admin/branches

HTTP Request:

```
POST /admin/branches
Authorization: Bearer jwt
Content-Type: application/json

{
  "branch_name": "Computer Science & Engineering",
  "branch_code": "CSE",
  "is_active": true
}
```

HTTP Response:

```
{
  "message": "Branch created successfully",
  "branch": {
    "id": "cse-id",
    "branch_name": "Computer Science & Engineering",
    "branch_code": "CSE",
    "is_active": true
  }
}
```

Create semester:

> POST /admin/semesters

HTTP Request:

```
POST /admin/semesters
Authorization: Bearer jwt
Content-Type: application/json

{
  "semester_name": "Semester 6",
  "semester_number": 6,
  "is_active": true
}
```

HTTP Response:

```
{
  "message": "Semester created successfully",
  "semester": {
    "id": "sem-6",
    "semester_name": "Semester 6",
    "semester_number": 6,
    "is_active": true
  }
}
```

Create section:

> POST /admin/sections

HTTP Request:

```
POST /admin/sections
Authorization: Bearer jwt
Content-Type: application/json

{
  "branch_id": "cse-id",
  "semester id": "sem-6",
  "section_name": "A",
  "max_students": 60
}
```

HTTP Response:

```
{
  "message": "Section created successfully",
  "section": {
    "id": "sec-a-id",
    "branch_id": "cse-id",
    "semester_id": "sem-6",
    "section_name": "A",
    "max_students": 60
  }
}
```

6. Subjects Management Page

Page load:

> GET /admin/subjects

Create subject:

> POST /admin/subjects

HTTP Request:

```
POST /admin/subjects
Authorization: Bearer jwt
Content-Type: application/json

{
  "subject_code": "CS601",
  "subject_name": "Machine Learning",
  "subject_type": "ELECTIVE",
  "branch_id": "cse-id",
  "semester_id": "sem-6"
}
```

HTTP Response:

```
{
  "message": "Subject created successfully",
  "subject": {
    "id": "cs601-id",
    "subject_code": "CS601",
    "subject_name": "Machine Learning",
    "subject_type": "ELECTIVE"
  }
}
```

Update subject:

> PATCH /admin/subjects/{subjectId}

HTTP Request:

```
PATCH /admin/subjects/cs601-id
Authorization: Bearer jwt
Content-Type: application/json

{
  "subject_code": "CS601",
  "subject_name": "Machine Learning",
  "subject_type": "ELECTIVE",
  "is_active": true
}
```

HTTP Response:

```
{
  "message": "Subject updated successfully",
  "subject": {
    "id": "cs601-id",
    "subject_code": "CS601",
    "subject_name": "Machine Learning",
    "subject_type": "ELECTIVE",
    "is_active": true
  }
}
```

7. Teacher Assignment Page

Page load:

> GET /admin/teacher-assignments

HTTP Request:

```
GET /admin/teacher-assignments
Authorization: Bearer jwt
```

HTTP Response:

```
{
  "assignments": [
    {
      "id": "assign-1",
      "teacher_id": "t-1",
      "teacher_name": "Rahul Mehta",
      "section_id": "sec-a",
      "section_name": "CSE-A",
      "subject_id": "cs601",
      "subject_name": "Machine Learning",
      "is_active": true
    }
  ]
}
```

Create teacher assignment:

> POST /admin/teacher-assignments

HTTP Request:

```
POST /admin/teacher-assignments
```

```
Authorization: Bearer jwt
Content-Type: application/json

{
  "teacher_id": "t-1",
  "section_id": "sec-a",
  "subject_id": "cs601"
}
```

HTTP Response:

```
{
  "message": "Teacher assignment created successfully",
  "assignment": {
    "id": "assign-1",
    "teacher_id": "t-1",
    "section_id": "sec-a",
    "subject_id": "cs601"
  }
}
```

Update teacher assignment:

> PATCH /admin/teacher-assignments/{assignmentId}

HTTP Request:

```
PATCH /admin/teacher-assignments/assign-1
Authorization: Bearer jwt
Content-Type: application/json

{
  "teacher_id": "t-2",
  "section_id": "sec-a",
  "subject_id": "cs601",
  "is_active": true
}
```

HTTP Response:

```
{
  "message": "Teacher assignment updated successfully",
  "assignment": {
    "id": "assign-1",
    "teacher_id": "t-2",
    "section_id": "sec-a",
    "subject_id": "cs601",
    "is_active": true
  }
}
```

8. Timetable Management Page

Page load:

> GET /admin/timetable/{sectionId}

Create timetable:

> POST /admin/timetable

Update timetable:

> PUT /admin/timetable/{timetableId}

HTTP Request:

```
PUT /admin/timetable/tt-1
Authorization: Bearer jwt
Content-Type: application/json

{
  "section_id": "sec-a-uuid",
  "day": "MONDAY",
  "period": 3,
  "subject_id": "ml-subject-uuid",
  "teacher_id": "t-1",
  "start_time": "10:00",
  "end_time": "11:00",
  "room": "B-204"
```

```
}
```

HTTP Response:

```
{
  "message": "Timetable updated successfully",
  "timetable": {
    "id": "tt-1",
    "day": "MONDAY",
    "period": 3,
    "subject": "Machine Learning",
    "room": "B-204"
  }
}
```

9. Exams Management Page

Page load:

> GET /admin/exams

Create exam:

> POST /admin/exams

Update exam:

> PUT /admin/exams/{examId}

HTTP Request:

```
PUT /admin/exams/exam-1
Authorization: Bearer jwt
Content-Type: application/json

{
  "exam_name": "Mid-Semester Exam",
```

```
  "subject_id": "ml-subject-uuid",
  "section_id": "sec-a-uuid",
  "exam_date": "2026-03-15",
  "total_marks": 100,
  "passing_marks": 40
}
```

HTTP Response:

```
{
  "message": "Exam updated successfully",
  "exam": {
    "id": "exam-1",
    "exam_name": "Mid-Semester Exam",
    "subject": "Machine Learning",
    "exam_date": "2026-03-15"
  }
}
```

10. Results Review Page

```
GET /admin/exams/{examId}/marks
PATCH /admin/exams/{examId}/marks/approve
```

11. Admin Notifications Page

```
GET /admin/notifications
POST /admin/notifications
```

Teacher User Journey

1. Teacher Login → Teacher Dashboard

(Login flow already explained earlier, reused here)

After login, role = TEACHER → redirected to Teacher Dashboard

2. Screen: Teacher Dashboard

What Teacher sees:

- Assigned classes & subjects
- Pending attendance count
- Pending leave requests
- Upcoming exams
- Navigation menu: My Timetable, Attendance, Exams & Marks, Leave Approvals, Notifications
- Chatbot widget

API 1: Teacher Dashboard Summary

> GET /teacher/dashboard

HTTP Request:

```
GET /teacher/dashboard
Authorization: Bearer jwt_token
```

HTTP Response:

```
{
  "assigned_sections": [
    {
      "section_id": "sec-a-uuid",
      "section_name": "CSE-A",
      "subject": "Machine Learning"
    }
  ],
  "pending_attendance": 1,
  "pending_leaves": 3,
  "upcoming_exams": 2
}
```

Backend checks TEACHER_ASSIGNMENTS table to know which section and which subject. Teacher is allowed to see only these.

Teacher uses chatbot

> POST /chat/query

```
Authorization: Bearer jwt_token

{
  "message": "Show attendance defaulters in my class"
}
```

Response:

```
{
  "answer": "2 students are below 75% attendance",
  "students": ["Amit Sharma", "Riya Patel"]
}
```

3. Screen: My Timetable (Teacher)

What Teacher sees:

- Weekly timetable
- Periods
- Subject
- Section
- Room number

API called on page load:

> GET /teacher/timetable

HTTP Request:

```
GET /teacher/timetable
Authorization: Bearer jwt_token
```

HTTP Response:

```
{
  "timetable": [
    {
      "day": "MONDAY",
      "period": 3,
      "subject": "Machine Learning",
      "section": "CSE-A",
      "start_time": "10:00",
      "end_time": "11:00",
      "room": "B-204"
    }
  ]
}
```

Backend joins: TIMETABLE, TEACHER_ASSIGNMENTS

Create timetable (Role-based access):

> POST /teacher/timetable

HTTP Request:

```
POST /teacher/timetable
Authorization: Bearer jwt_token
Content-Type: application/json

{
  "section_id": "sec-a-uuid",
  "day": "TUESDAY",
  "period": 2,
  "subject_id": "ml-subject-uuid",
```

```
  "start_time": "09:00",
  "end_time": "10:00",
  "room": "B-205"
}
```

HTTP Response:

```
{
  "message": "Timetable created successfully",
  "timetable": {
    "id": "tt-2",
    "day": "TUESDAY",
    "period": 2,
    "subject": "Machine Learning",
    "room": "B-205"
  }
}
```

Backend validation:

- Teacher must be assigned to this section
- Teacher must be assigned to this subject
- No conflict with existing timetable

4. Screen: Attendance (Teacher)

What Teacher sees:

- Section selector
- Subject selector
- Student list
- Present / Absent buttons
- Submit attendance

API called on page load:

> GET /teacher/attendance/{sectionId}

HTTP Request:

```
GET /teacher/attendance/sec-a-uuid
Authorization: Bearer jwt_token
```

HTTP Response:

```
{
  "students": [
    {
      "student_id": "stu-1",
      "name": "Amit Sharma"
    },
    {
      "student_id": "stu-2",
      "name": "Riya Patel"
    }
  ]
}
```

Backend validates:

- Teacher is assigned to this section
- Subject belongs to teacher

Teacher clicks Submit Attendance:

> POST /teacher/attendance

HTTP Request:

```
POST /teacher/attendance
Authorization: Bearer jwt_token

{
```

```
  "student_id": "stu-1",
  "section_id": "sec-a-uuid",
  "subject_id": "ml-subject-uuid",
  "attendance_date": "2026-02-15",
  "status": "PRESENT"
}
```

HTTP Response:

```
{
  "message": "Attendance marked successfully"
}
```

Backend inserts record into ATTENDANCE table

Update attendance:

> PUT /teacher/attendance/{attendanceId}

HTTP Request:

```
PUT /teacher/attendance/att-1
Authorization: Bearer jwt_token
Content-Type: application/json

{
  "status": "ABSENT",
  "remarks": "Updated after verification"
}
```

HTTP Response:

```
{
  "message": "Attendance updated successfully",
  "attendance": {
    "id": "att-1",
    "student": "Amit Sharma",
```

```
      "status": "ABSENT",
      "date": "2026-02-15"
    }
  }
```

5. Screen: Exams & Marks (Teacher)

What Teacher sees:

- Assigned exams list
- Enter marks button

API called on page load:

> GET /teacher/exams

HTTP Request:

```
GET /teacher/exams
Authorization: Bearer jwt_token
```

Response:

```
{
  "exams": [
    {
      "exam_id": "exam-1",
      "subject": "Machine Learning",
      "date": "2026-03-01"
    }
  ]
}
```

Teacher selects exam → student list:

> GET /teacher/exams/{examId}/students

Response:

```
{
  "students": [
    { "student_id": "stu-1", "name": "Amit Sharma" },
    { "student_id": "stu-2", "name": "Riya Patel" }
  ]
}
```

Teacher submits marks:

> POST /teacher/exams/{examId}/marks

HTTP Request:

```
POST /teacher/exams/exam-1/marks
Authorization: Bearer jwt_token

{
  "student_id": "stu-1",
  "marks_obtained": 78,
  "is_absent": false
}
```

Response:

```
{
  "status": "SUBMITTED_FOR_APPROVAL"
}
```

Stored in EXAM_MARKS table (pending admin approval)

6. Screen: Leave Approvals (Teacher)

What Teacher sees:

- Student leave requests
- Approve / Reject buttons

API called on page load:

> GET /teacher/leaves

Response:

```
{
  "leaves": [
    {
      "leave_id": "leave-1",
      "student": "Amit Sharma",
      "reason": "Medical",
      "status": "PENDING"
    }
  ]
}
```

Teacher clicks Approve:

> PATCH /teacher/leaves/{leaveId}/approve

HTTP Request:

```
PATCH /teacher/leaves/leave-1/approve
Authorization: Bearer jwt_token
```

Response:

```
{
  "status": "APPROVED"
}
```

7. Screen: Teacher Notifications

What Teacher sees:

- List of notifications
- Read / unread state

API on page load:

> GET /teacher/notifications

Response:

```
{
  "notifications": [
    {
      "id": "noti-1",
      "message": "Marks approved by admin",
      "is_read": false
    }
  ]
}
```

Mark notification as read:

> PATCH /teacher/notifications/{id}/read

Send notification:

> POST /teacher/notifications

HTTP Request:

```
POST /teacher/notifications
Authorization: Bearer jwt_token
Content-Type: application/json

{
  "recipient_ids": ["stu-1", "stu-2"],
  "message": "Assignment submission deadline extended to Feb 20",
  "notification_type": "ANNOUNCEMENT"
}
```

HTTP Response:

```
{
  "message": "Notification sent successfully",
  "recipients_count": 2
}
```

Student User Journey

1. Student Dashboard

What Student sees:

- Attendance %
- Marks summary
- Upcoming exams
- Timetable snapshot
- Chatbot widget
- Navigation: Timetable, Attendance, Exams, Leave, Elective Selection

API called on page load:

> GET /student/dashboard

Response:

```
{
  "attendance_percentage": 82,
  "upcoming_exams": 2,
  "average_marks": 75
}
```

2. Screen: Student Timetable

API:

> GET /student/timetable

Response:

```
{
  "timetable": [
    {
      "day": "MONDAY",
      "period": 3,
      "subject": "Machine Learning",
      "room": "B-204"
    }
  ]
}
```

3. Screen: Student Attendance

API:

> GET /student/attendance

Response:

```
{
  "attendance": [
    {
      "subject": "Machine Learning",
      "attendance_percentage": 85
    }
  ]
}
```

Backend filters electives using STUDENT_ELECTIVES

4. Screen: Student Exams & Results

Page load API:

> GET /student/exams

Response:

```
{
  "exams": [
    {
      "exam_id": "exam-1",
      "subject": "Machine Learning",
      "date": "2026-03-01"
    }
  ]
}
```

View marks:

> GET /student/exams/{examId}/marks

Response:

```
{
  "marks": 78,
```

```
  "status": "PASSED"
}
```

5. Screen: Elective Selection

What Student sees:

- List of elective subjects
- Select button

API on page load (via subjects):

> GET /admin/subjects?type=ELECTIVE

Student selects elective:

> POST /student/electives

HTTP Request:

```
POST /student/electives
Authorization: Bearer jwt_token

{
  "subject_id": "ml-subject-uuid"
}
```

Response:

```
{
  "status": "ELECTED"
}
```

6. Screen: Leave & Notifications (Student)

Apply leave:

> POST /student/leaves

Request:

```
{
  "start_date": "2026-02-20",
  "end_date": "2026-02-21",
  "reason": "Family Function"
}
```

View leaves:

> GET /student/leaves

Notifications:

```
GET /student/notifications
PATCH /student/notifications/{id}/read
```

API Requirements

Authentication APIs

Method	Endpoint	Description	Request Body	Response
POST	/auth/login	User login	`{ "email": "user@college.com",`	`{ "access_token": "jwt", "role": "ADMIN/TEACHER/ST`

Method	Endpoint	Description	Request Body	Response
			`"password": "Pass@123" }`	`UDENT", "user": {...} }`
POST	/auth/logout	User logout	None	`{ "message": "Logged out successfully" }`
POST	/auth/forgot-password	Request password reset	`{ "email": "user@college.com" }`	`{ "message": "Reset link sent" }`
POST	/auth/reset-password	Reset password	`{ "token": "reset-token", "new_password": "New@123" }`	`{ "message": "Password reset successful" }`

Admin APIs

Dashboard

Method	Endpoint	Description	Response
GET	/admin/dashboard	Get dashboard stats	`{ "students": 1200, "teachers": 75, "sections": 30, "pending_leaves": 18, "active_exams": 4 }`
GET	/admin/dashboard/activity	Get recent activity feed	`{ "activities": [...] }`

Student Management

Method	Endpoint	Description	Request Body	Response
GET	/admin/students	Get all students	-	`{ "students": [...] }`
GET	/admin/students/{studentId}	Get student by ID	-	`{ "id": "stu-1", "name": "Amit", ... }`
POST	/admin/students	Create new student	`{ "email": "...", "first_name": "...", "last_name": "...", "branch_id": "...",`	`{ "message": "Student created",`

Method	Endpoint	Description	Request Body	Response
			`"section_id": "..." }`	`"student": {...} }`
PUT	/admin/students/{studentId}	Update student details	`{ "first_name": "...", "last_name": "...", "email": "...", "branch_id": "...", "section_id": "...", "is_active": true }`	`{ "message": "Student updated", "student": {...} }`
POST	/admin/students/import	Import students via CSV	CSV file	`{ "total_rows": 100, "inserted": 95, "failed": 5 }`

Teacher Management

Method	Endpoint	Description	Request Body	Response
GET	/admin/teachers	Get all teachers	-	`{ "teachers": [...] }`
GET	/admin/teachers/{teacherId}	Get teacher by ID	-	`{ "id": "t-1", "name": "Rahul", ... }`
POST	/admin/teachers	Create new teacher	`{ "email": "...", "first_name": "...", "last_name": "...", "department": "..." }`	`{ "message": "Teacher created", "teacher": {...} }`
PUT	/admin/teachers/{teacherId}	Update teacher details	`{ "first_name": "...", "last_name": "...", "email": "...", "department": "...", "is_active": true }`	`{ "message": "Teacher updated", "teacher": {...} }`
POST	/admin/teachers/import	Import teachers via CSV	CSV file	`{ "total_rows": 50,`

Method	Endpoint	Description	Request Body	Response
				`"inserted": 48, "failed": 2 }`

Academic Structure

Method	Endpoint	Description	Request Body	Response
GET	/admin/branches	Get all branches	-	`{ "branches": [...] }`
POST	/admin/branches	Create new branch	`{ "branch_name": "...", "branch_code": "...", "is_active": true }`	`{ "message": "Branch created", "branch": {...} }`
PATCH	/admin/branches/{branchId}	Update branch	`{ "branch_name": "...", "is_active": true }`	`{ "message": "Branch updated", "branch": {...} }`
GET	/admin/semesters	Get all semesters	-	`{ "semesters": [...] }`
POST	/admin/semesters	Create new semester	`{ "semester_name": "...", "semester_number": 6, "is_active": true }`	`{ "message": "Semester created", "semester": {...} }`
PATCH	/admin/semesters/{semesterId}	Update semester	`{ "semester_name": "...", "is_active": true }`	`{ "message": "Semester updated", "semester": {...} }`
GET	/admin/sections	Get all sections	-	`{ "sections": [...] }`
POST	/admin/sections	Create new section	`{ "branch_id": "...", "semester_id": "...", "section_name": "A", "max_students": 60 }`	`{ "message": "Section created", "section": {...} }`

Subjects

Method	Endpoint	Description	Request Body	Response
GET	/admin/subjects	Get all subjects	-	`{ "subjects": [...] }`
POST	/admin/subjects	Create new subject	`{ "subject_code": "CS601", "subject_name": "ML", "subject_type": "ELECTIVE", "branch_id": "...", "semester_id": "..." }`	`{ "message": "Subject created", "subject": {...} }`
PATCH	/admin/subjects/{subjectId}	Update subject	`{ "subject_code": "...", "subject_name": "...", "subject_type": "...", "is_active": true }`	`{ "message": "Subject updated", "subject": {...} }`

Teacher Assignment

Method	Endpoint	Description	Request Body	Response
GET	/admin/teacher-assignments	Get all assignments	-	`{ "assignments": [...] }`
POST	/admin/teacher-assignments	Assign teacher to section/subject	`{ "teacher_id": "...", "section_id": "...", "subject_id": "..." }`	`{ "message": "Assignment created", "assignment": {...} }`
PATCH	/admin/teacher-assignments/{assignmentId}	Update teacher assignment	`{ "teacher_id": "...", "section_id": "...", "subject_id": "...", "is_active": true }`	`{ "message": "Assignment updated", "assignment": {...} }`

Timetable

Method	Endpoint	Description	Request Body	Response
GET	/admin/timetable/{sectionId}	Get timetable for section	-	`{ "timetable": [...] }`
POST	/admin/timetable	Create timetable entry	`{ "section_id": "...", "day": "MONDAY", "period": 3, "subject_id": "...", "teacher_id": "...", "start_time": "10:00", "end_time": "11:00", "room": "B-204" }`	`{ "message": "Timetable created" }`
PUT	/admin/timetable/{timetableId}	Update timetable entry	`{ "section_id": "...", "day": "...", "period": 3, "subject_id": "...", "teacher_id": "...", "start_time": "...", "end_time": "...", "room": "..." }`	`{ "message": "Timetable updated", "timetable": {...} }`

Exams

Method	Endpoint	Description	Request Body	Response
GET	/admin/exams	Get all exams	-	`{ "exams": [...] }`
POST	/admin/exams	Create new exam	`{ "exam_name": "...", "subject_id": "...", "section_id": "...", "exam_date": "...", "total_marks": 100, "passing_marks": 40 }`	`{ "message": "Exam created" }`
PUT	/admin/exams/{examId}	Update exam	`{ "exam_name": "...", "subject_id": "...", "section_id": "...", "exam_date": "...", "total_marks": 100, "passing_marks": 40 }`	`{ "message": "Exam updated", "exam": {...} }`

Method	Endpoint	Description	Request Body	Response
GET	/admin/exams/{examId}/marks	Get marks for exam	-	`{ "marks": [...] }`
PATCH	/admin/exams/{examId}/marks/approve	Approve submitted marks	-	`{ "message": "Marks approved" }`

Notifications

Method	Endpoint	Description	Request Body	Response
GET	/admin/notifications	Get all notifications	-	`{ "notifications": [...] }`
POST	/admin/notifications	Create/send notification	`{ "recipient_ids": [...], "message": "...", "notification_type": "..." }`	`{ "message": "Notification sent" }`

Teacher APIs

Dashboard

Method	Endpoint	Description	Response
GET	/teacher/dashboard	Get teacher dashboard summary	`{ "assigned_sections": [...], "pending_attendance": 1, "pending_leaves": 3, "upcoming_exams": 2 }`

Timetable

Method	Endpoint	Description	Request Body	Response
GET	/teacher/timetable	Get teacher's timetable	-	`{ "timetable": [...] }`
POST	/teacher/timetable	Create timetable (role-based)	`{ "section_id": "...", "day": "...", "period": 2, "subject_id": "...", "start_time": "...",`	`{ "message": "Timetable created", "timetable": {...} }`

Method	Endpoint	Description	Request Body	Response
			`"end_time": "...", "room": "..." }`	

Attendance

Method	Endpoint	Description	Request Body	Response
GET	/teacher/attendance/{sectionId}	Get students for attendance	-	`{ "students": [...] }`
POST	/teacher/attendance	Mark attendance	`{ "student_id": "...", "section_id": "...", "subject_id": "...", "attendance_date": "...", "status": "PRESENT/ABSENT" }`	`{ "message": "Attendance marked" }`
PUT	/teacher/attendance/{attendanceId}	Update attendance	`{ "status": "PRESENT/ABSENT", "remarks": "..." }`	`{ "message": "Attendance updated", "attendance": {...} }`

Exams & Marks

Method	Endpoint	Description	Request Body	Response
GET	/teacher/exams	Get assigned exams	-	`{ "exams": [...] }`
GET	/teacher/exams/{examId}/students	Get students for exam	-	`{ "students": [...] }`
POST	/teacher/exams/{examId}/marks	Submit marks	`{ "student_id": "...", "marks_obtained": 78, "is_absent": false }`	`{ "status": "SUBMITTED_FOR_APPROVAL" }`

Leave Approvals

Method	Endpoint	Description	Request Body	Response
GET	/teacher/leaves	Get pending leave requests	-	`{ "leaves": [...] }`
PATCH	/teacher/leaves/{leaveId}/approve	Approve leave request	-	`{ "status": "APPROVED" }`

Notifications

Method	Endpoint	Description	Request Body	Response
GET	/teacher/notifications	Get teacher notifications	-	`{ "notifications": [...] }`
POST	/teacher/notifications	Send notification to students	`{ "recipient_ids": [...], "message": "...", "notification_type": "..." }`	`{ "message": "Notification sent", "recipients_count": 2 }`
PATCH	/teacher/notifications/{id}/read	Mark notification as read	-	`{ "message": "Marked as read" }`

Student APIs

Dashboard

Method	Endpoint	Description	Response
GET	/student/dashboard	Get student dashboard	`{ "attendance_percentage": 82, "upcoming_exams": 2, "average_marks": 75 }`

Timetable

Method	Endpoint	Description	Response
GET	/student/timetable	Get student's timetable	`{ "timetable": [...] }`

Attendance

Method	Endpoint	Description	Response
GET	/student/attendance	Get attendance records	`{ "attendance": [...] }`

Exams & Results

Method	Endpoint	Description	Response
GET	/student/exams	Get exam schedule	`{ "exams": [...] }`
GET	/student/exams/{examId}/marks	Get marks for exam	`{ "marks": 78, "status": "PASSED" }`

Electives

Method	Endpoint	Description	Request Body	Response
POST	/student/electives	Select elective subject	`{ "subject_id": "..." }`	`{ "status": "ELECTED" }`

Leave & Notifications

Method	Endpoint	Description	Request Body	Response
POST	/student/leaves	Apply for leave	`{ "start_date": "...", "end_date": "...", "reason": "..." }`	`{ "message": "Leave applied" }`
GET	/student/leaves	Get leave history	-	`{ "leaves": [...] }`
GET	/student/notifications	Get notifications	-	`{ "notifications": [...] }`
PATCH	/student/notifications/{id}/read	Mark notification as read	-	`{ "message": "Marked as read" }`

Chatbot API

Method	Endpoint	Description	Request Body	Response
POST	/chat/query	Query chatbot (role-based)	`{ "message": "Show attendance defaulters" }`	`{ "answer": "...", "data": [...] }`

Screen List + UI Planning

Authentication Screens

- Login
- Forgot Password
- Reset / Change Password

Admin Screens

- Admin Dashboard
- Student Management
- Teacher Management
- User Profile (Read-Only)
- Academic Structure Management
- Subjects Management (Core & Elective)
- Teacher Assignment
- Timetable Management
- Exams Management
- Results Review & Publish
- Admin Notifications

Teacher Screens

- Teacher Dashboard
- Teacher Timetable
- Attendance
- Exams & Marks
- Leave Approvals
- Teacher Notifications

Student Screens

- Student Dashboard
- Student Timetable
- Student Attendance
- Student Exams & Results
- Elective Selection

- Student Leave & Notifications

System Assumptions

1. Organizational Assumptions

1. University / Institution Exists

- The system is built for one institution
- Multi-university support is out of scope
- All data belongs to a single academic organization
- Your schema does not have a university_id or tenant_id

Reason: Schema design indicates single-tenant architecture.

2. Academic Structure Is Well-Defined

- Branches (CSE, ECE, etc.) exist
- Semesters are predefined
- Sections belong to a branch + semester

Reason: Students, subjects, exams, timetable, and attendance all depend on this hierarchy.

2. User & Role Assumptions

3. Roles Are Fixed and Finite

- Only three roles exist: ADMIN, TEACHER, STUDENT
- No dynamic or custom roles

Reason: RBAC logic and APIs are hard-coded around these roles.

Critical assumption

4. At Least One Admin Exists Before System Use

- First admin is created via DB seed script OR secure bootstrap process
- No public signup for admin

Reason: Without an admin, no configuration can happen.

5. Admins Do Not Manage Other Admins (by design)

- Admins manage only: Students, Teachers
- Admin visibility into other admins is restricted

Reason: Security and privilege isolation.

3. Data Integrity Assumptions

6. Email Is Unique Across All Users

- No two users share the same email
- Email is the login identifier

Reason: Authentication depends on unique email lookup.

7. Users Are Soft-Deleted, Not Hard-Deleted

- Users are deactivated via is_active
- Historical data (attendance, marks) remains intact

Reason: Attendance, exams, and results depend on historical users.

8. A Student Belongs to Only One Section at a Time

- No multi-section enrollment
- Section changes happen administratively

Reason: Attendance and timetable logic assume one section per student.

4. Academic Flow Assumptions

9. Teacher Authority Is Explicitly Assigned

- Teachers cannot mark attendance, enter marks, or approve leaves unless assigned via Teacher Assignment

Reason: TEACHER_ASSIGNMENTS is the authority source.

10. Subjects Belong to One Branch + Semester

- A subject cannot span multiple branches or semesters

Reason: Timetable, exams, and electives depend on this mapping.

11. Electives Are Chosen Explicitly by Students

- Elective subjects are created by admin
- Elective subjects are selected by students
- Core subjects apply to all students in a section

Reason: Attendance and exams are filtered using STUDENT_ELECTIVES.

5. Attendance & Exam Assumptions

12. Attendance Is Marked Once Per Subject Per Day

- One attendance record per: Student, Subject, Date

Reason: Prevents duplicate attendance entries.

13. Exams Are Section + Subject Specific

- An exam belongs to one subject and one section
- No cross-section exams

Reason: Marks and approval workflow rely on this.

14. Exam Marks Require Admin Approval

- Teacher submits marks
- Admin approves
- Students see results only after approval

Reason: Prevents unauthorized or premature result exposure.

6. Leave & Notification Assumptions

15. Leave Approval Is Teacher-Driven

- Student applies for leave
- Assigned teacher reviews it
- Admin sees summary only

Reason: Teacher is closest authority to students.

16. Notifications Are Per-User, Not Broadcast Rows

- A notification row exists per recipient
- Broadcast = multiple inserts

Reason: Your NOTIFICATIONS table has user_id.

7. Security & Access Assumptions

17. Backend Enforces RBAC (Not UI)

- Frontend hides buttons
- Backend always validates role & ownership

Reason: UI alone cannot be trusted.

18. JWT Is Required for Every Protected API

- Token contains: user_id, role
- Token is validated on every request

Reason: Stateless, secure access control.

8. Operational Assumptions

19. CSV Imports Are Clean and Validated

- Admin uploads correct CSV format
- Backend validates each row
- Partial success is allowed

Reason: Batch operations must not corrupt data.

20. System Is Used in Normal Academic Scale

- Thousands of students
- Hundreds of teachers
- Not millions

Reason: Query patterns and pagination assume this scale.

9. Chatbot-Specific Assumptions

21. Chatbot Is Read-Only

- Chatbot does NOT: Create, Update, or Delete data
- Only queries existing data

Reason: Safety and data integrity.

22. Chatbot Respects RBAC Automatically

- Admin → system-wide data
- Teacher → assigned scope
- Student → own data

Reason: Chatbot uses same permission layer as APIs.

Screen-Wise Functionalities

Authentication Screens

1. Login

- Enter email and password
- Authenticate user
- Resolve user role (Admin / Teacher / Student)
- Create user session
- Redirect to role-specific dashboard

2. Forgot Password

- Enter registered email
- Request password reset link

3. Reset / Change Password

- Reset password using secure token
- Force password change on first login
- Validate password strength

Admin Screens

4. Admin Dashboard

- View total students count
- View total teachers count
- View total sections count
- View pending leave requests
- View active exams
- View recent system activity
- Navigate to all admin modules
- Use chatbot for system-level queries

5. Student Management

- View list of students
- Search, filter, and paginate students
- Add student manually
- Import students via CSV
- Edit student details
- Activate or deactivate student
- Open student profile (read-only)

6. Teacher Management

- View list of teachers

- Search, filter, and paginate teachers
- Add teacher manually
- Import teachers via CSV
- Edit teacher details
- Activate or deactivate teacher
- Open teacher profile (read-only)

7. User Profile (Read-Only)

- View personal details
- View academic or department details
- View account status (active/inactive)

8. Academic Structure Management

- Create and manage branches
- Create and manage semesters
- Create and manage sections
- Define section capacity
- Enable or disable academic entities

9. Subjects Management (Core & Elective)

- Create new subjects
- Update subject details
- Mark subject as CORE or ELECTIVE
- Map subjects to branch and semester
- Enable or disable subjects

10. Teacher Assignment

- Assign teacher to a section
- Assign teacher to a subject
- View existing teacher assignments
- Remove teacher assignments

- Define teacher authority boundaries

11. Timetable Management

- Create timetable for sections
- Assign subjects and teachers to time slots
- Schedule core subjects
- Schedule elective slots
- Validate timetable conflicts
- Publish timetable

12. Exams Management

- Create exams
- Define exam type and schedule
- Set total and passing marks
- Publish exams

13. Results Review & Publish

- View marks submitted by teachers
- Review subject-wise marks
- Approve or reject marks
- Publish results for students

14. Admin Notifications

- Send notifications to users
- Broadcast announcements
- View sent notifications

Teacher Screens

15. Teacher Dashboard

- View assigned sections and subjects
- View pending attendance tasks
- View pending leave requests
- View upcoming exams
- Navigate to teacher modules
- Use chatbot within assigned scope

16. Teacher Timetable

- View personal teaching schedule
- View class timings and rooms

17. Attendance

- Select section and subject
- View enrolled students
- Mark attendance for core subjects
- Mark attendance for elective subjects
- Add attendance remarks
- Submit attendance

18. Exams & Marks

- View assigned exams
- View students for an exam
- Enter marks for students
- Mark students as absent
- Submit marks for approval

19. Leave Approvals

- View student leave requests
- Approve leave requests
- Reject leave requests

20. Teacher Notifications

- View notifications
- Mark notifications as read

Student Screens

21. Student Dashboard

- View attendance percentage
- View marks summary
- View upcoming exams
- View timetable snapshot
- View performance analytics
- Use chatbot for self-data queries

22. Student Timetable

- View class timetable
- View elective slots

23. Student Attendance

- View attendance history
- View subject-wise attendance
- View attendance percentage

24. Student Exams & Results

- View exam schedule
- View published results
- View subject-wise marks

25. Elective Selection

- View available elective subjects
- Select elective subjects
- Confirm elective selection
- Prevent duplicate or invalid selections

26. Student Leave & Notifications

- Apply for leave
- View leave status
- View notifications
- Mark notifications as read

Overall Solutioning

The proposed solution is a role-based School Management System designed to digitize and streamline core academic and administrative workflows defined in the PRD. The system focuses on clarity, minimalism, and correctness by ensuring that every feature, screen, API, and database table directly supports an in-scope requirement.

The solution is built around three primary user roles: Admin, Teacher, and Student. Each role has clearly defined responsibilities and access boundaries, ensuring secure and controlled interaction with the system. Role-based access control is enforced consistently across the backend APIs and frontend user interface to prevent unauthorized actions and data exposure.

At a high level, the admin acts as the system configurator and controller. Admin users are responsible for setting up academic structures (classes, subjects), onboarding students and teachers, assigning teachers to classes and subjects, creating timetables and exams, reviewing teacher-submitted marks, publishing results, and broadcasting announcements. This central control ensures data consistency and institutional oversight.

The Teacher role is designed around daily academic operations. Teachers can view their assigned timetables, mark attendance for their classes, enter exam marks, submit marks for approval, approve or reject student leave requests, and receive system notifications. Teachers are strictly restricted to data and actions related only to their assigned classes and subjects, as mandated by the PRD.

The student role is primarily read-only with limited controlled actions. Students can view their timetables, attendance records, exam schedules, and published results. They can also apply for leave and track its approval status, as well as receive announcements and notifications. Students can access only their own academic data, ensuring privacy and security.

From a data perspective, the system uses a normalized PostgreSQL schema with carefully chosen tables that avoid redundancy. A single user's table supports all roles, while operational tables such as attendance, exams, marks, leaves, timetables, and notifications capture academic activities. Derived data like results and performance metrics are computed dynamically from stored records instead of being redundantly persisted, keeping the data model simple and consistent.

The backend, implemented using FastAPI, serves as the single source of truth. It enforces all validations, business rules, and access checks defined in the PRD. RESTful APIs are grouped by role and responsibility, ensuring a clear separation of concerns and easy maintainability. Critical operations are wrapped in database transactions to maintain data integrity.

The frontend, built as a React Single Page Application, provides a clean and responsive user experience. Role-based dashboards and minimal screens ensure usability without overwhelming users. Reusable components and read-only overlays are used to reduce unnecessary navigation and complexity. Every screen maps directly to backend APIs, ensuring there is no unsupported or redundant UI.

Overall, the solution delivers a cohesive, secure, and scalable academic management platform that strictly adheres to the PRD, avoids over-engineering, and provides a clear, maintainable foundation for future enhancements.

Implementation Approach

Overall Architecture

The system follows a role-based, modular, client-server architecture aligned with the PRD requirements.

Frontend:

A Single Page Application (SPA) developed using JavaScript (React) to provide a responsive and role-based user interface.

Backend:

A set of RESTful APIs implemented using FastAPI, responsible for handling business logic, validations, and access control.

Database:

PostgreSQL is used as the primary relational database for storing all academic and operational data.

Authentication:

JWT-based authentication is used along with Role-Based Access Control (RBAC).

The frontend communicates with the backend exclusively through REST APIs.

The backend acts as the single source of truth and enforces all business rules defined in the PRD.

Role-Based Access Control (RBAC)

The system supports the following roles defined in the PRD:

- Admin
- Teacher
- Student

RBAC is enforced at two levels.

Backend Level

- Each API endpoint is protected using JWT authentication.
- Role information is embedded inside the JWT token.

Middleware validates:

- Authentication (valid token)
- Authorization (role-based access)

Examples:

- Only Admin users can access /admin/* APIs.
- Teachers cannot access admin configuration APIs.
- Students can only fetch their own data.

Frontend Level

- UI screens are conditionally rendered based on the logged-in user's role.
- Users only see screens relevant to their responsibilities.
- Unauthorized navigation is blocked at the routing level.

Authentication & Security Approach

Authentication Flow

- The user logs in using email and password.
- The backend validates credentials against the users table.

On successful validation:

- A JWT token is generated.
- User ID and role are embedded inside the token.
- The token is returned to the frontend and stored securely.
- All subsequent API requests include the JWT token.

Password Security

- Passwords are never stored in plain text.
- Passwords are hashed using industry-standard hashing algorithms.
- First-time login enforces a password change using the is_first_login flag.

Session Management

- Stateless authentication is implemented using JWT.
- Logout is handled via a POST API that invalidates the session or token.

Frontend Implementation Strategy

UI Design

- Role-based dashboards are provided for Admin, Teacher, and Student users.
- The number of screens is kept minimal as specified in the PRD.
- Reusable UI components are used for tables, forms, modals, and notifications.

Screen-to-API Mapping

- Each screen maps directly to one or more backend APIs.
- No screen exists without backend support.
- Read-only overlays are used instead of creating unnecessary screens, reducing UI complexity.

State Management

- Local state is used for form handling and UI interactions.
- A centralized API service layer is used for backend communication.
- Proper loading, error, and empty states are handled across all screens.

Backend Implementation Strategy

API Design

RESTful APIs are grouped by role:

- /auth/*
- /admin/*
- /teacher/*
- /student/*

A clear separation of concerns is maintained between:

- Controllers
- Validation
- Business Logic
- Database Access

Business Logic Handling

All validations are enforced at the backend, including:

- Role validation
- Teacher-class-subject assignment checks
- Exam publishing rules
- Attendance marking rules

Derived data such as results, percentages, and rankings are computed dynamically from stored records to avoid redundant storage.

Error Handling

- Standardized error responses are used.
- Meaningful HTTP status codes are returned.
- Validation errors are clearly communicated to the frontend.

Database Interaction Approach

Schema Usage

- PostgreSQL is used with a normalized schema.
- Relationships are enforced using foreign key constraints.
- No redundant or unused tables exist.

Key Design Decisions

- A single users table is used for all roles.
- No separate results table is maintained; results are computed from marks.
- Notifications are stored in a single unified table.
- Soft-delete logic is implemented using active or status flags where required.

Transactions

Critical operations such as student creation, exam publishing, and marks approval are handled using database transactions to ensure data consistency.

Notifications Handling

- Notifications are stored in a unified notifications table.

They are used for:

- System announcements
- Leave status updates
- Exam and result publishing
- Users fetch notifications via role-specific APIs.
- Read and unread states are managed using flags.
- No separate email logging table is maintained, keeping the design strictly aligned with the PRD.

Scalability & Maintainability

Scalability

- The stateless backend design supports horizontal scaling.
- There is a clear separation between frontend and backend layers.
- The database schema supports growth in users, classes, and academic records.

Maintainability

- A modular code structure is followed on both frontend and backend.
- API contracts are clear and well-defined.
- Role-based separation reduces coupling between modules.
- The system can be easily extended with additional features in the future without redesign.

Non-Functional Requirements Alignment

- Security: JWT authentication, role checks, hashed passwords
- Performance: Optimized database queries and pagination
- Maintainability: Modular architecture and clean separation of concerns
- Usability: Role-specific and minimal user interface
- Reliability: Use of database transactions and validations

This approach improves maintainability, scalability, and ease of testing.

Backend Folder Structure

```
backend/
├── app/
│   ├── main.py
│   ├── core/
│   ├── database/
│   ├── models/
│   ├── schemas/
│   ├── routers/
│   ├── services/
│   └── utils/
├── migrations/
├── requirements.txt
└── README.md
```

Frontend Folder Structure

```
frontend/
├── src/
│   ├── api/
│   ├── components/
│   ├── pages/
│   ├── routes/
│   ├── utils/
│   ├── App.js
│   └── index.js
├── package.json
└── README.md
```

Database Schema Design

The database is designed as a centralized relational system using PostgreSQL and acts as the single source of truth. It is optimized to remain simple while still covering all in-scope requirements from the PRD.

Key design principles:

- Strong relationships using foreign keys
- Role-based access enforced via mappings
- Soft deletes using is_active flags
- Audit fields for traceability

The core tables include:

- Users
- Branches
- Subjects
- Semester
- Section
- Teacher Assignments
- Attendance
- Notification

- Leave Application
- Timetable
- Exam Marks
- Exam
- Student_Elective

Entity Relationship Diagram:

[View ERD Diagram]