Upskillab Website

1. Home Page

- **Type**: Dynamic + Static
- **UI Components**:
    - **Header**: Logo ("Upskillab"), Navigation (Home, Courses, About Us, Career, Contact Us), Search Bar, Login Button.
    - **Hero**: "Never Stop Learning" + "Life Never Stop Teaching" + CTA ("Start Your Learning Journey") + Abstract shapes.
    - **Trending Categories**: Medical, Management, Professional Development, Technology, Psychology (Dynamic).
    - **About Sneak Peek**: "Get More About Us" + "8K+ Enrolled Students" (Static + Dynamic stat).
    - **Top Courses**: Course cards (e.g., "Certified AI/ML Practitioner") with prices, ratings, seats (Dynamic).
    - **Instructors**: "Our Top Class & Expert Instructors" + stats (5K+ Students, 15+ Courses) (Static + Dynamic).
    - **FAQs**: Static questions about Upskillab (Static).
    - **CTA Section**: "Learn with Experts", "Get Placement with MNC", etc. (Static).
    - **Testimonials**: Student reviews (Dynamic).
    - **Footer**: Logo, links, contact info, newsletter signup (Static + Dynamic form).
- **Frontend**: Responsive design, carousels for courses/testimonials, animations.
- **Backend**:
    - GET /api/courses/featured (top courses).
    - GET /api/categories (trending categories).
    - GET /api/testimonials (student reviews).
    - GET /api/stats (enrollment/instructor stats).

2. About Us

- **Type**: Static
- **UI Components**:
    - Full "About Us" description: "Upskillab is the premier online platform for upskilling..." + mission/vision.
    - Company details (Trivision Partners Pvt Ltd, Noida).
- **Frontend**: Clean layout, static text, optional team section.
- **Backend**: None (static).

3. Career

- **Type**: Dynamic
- **UI Components**:
    - Job listings (title, location, "Apply Now" button).

    - Filters (e.g., role type).
- **Frontend**: Grid layout, filterable list, job detail modal.
- **Backend**:
    - GET /api/jobs (fetch job listings).
    - POST /api/jobs/apply (optional application submission).

4. Contact Us

- **Type**: Static
- **UI Components**:
    - Contact form (name, email, message).
    - Address: H-187, Sector 63, Noida, UP 201301.
    - Mob: +91-9958958123, Email: info@upskillab.com.
- **Frontend**: Form with validation, embedded map (optional).
- **Backend**:
    - POST /api/contact (handle form submission).

5. Policy

- **Type**: Static
- **UI Components**:
    - Privacy policy text, readable layout.
- **Frontend**: Simple static page.
- **Backend**: None.

6. Terms & Conditions (T&C)

- **Type**: Static
- **UI Components**:
    - T&C text, last updated date.
- **Frontend**: Collapsible sections.
- **Backend**: None (static, unless CMS-driven).

I'll provide a detailed breakdown of the **Course Page** and **Course Detail** pages for the Upskillab website, based on your earlier inputs and requirements. These pages are dynamic, showcasing available courses and their specifics to engage students. Below, I'll outline the UI, functionality, frontend, backend, and step-by-step flow for both pages, tailored to your needs.

---

1. Course Page

Purpose: Display a list of all available courses, allowing users to browse, filter, and select a course for more details.

UI Components

- **Header**: Logo ("Upskillab"), Navigation (Home, Courses, About Us, Career, Contact Us), Search Bar, Login Button.
- **Title**: "Explore Our World's Best Courses".
- **Subtext**: "Boost your skills in data science, AI, cybersecurity, and development."
- **Filters**:
    - Categories (e.g., Technology, Psychology, Medical, Management, Professional Development).
    - Price range (e.g., ₹0-₹10,000, ₹10,000-₹20,000).
    - Sort by (e.g., Popularity, Newest, Price: Low to High).
- **Course Cards** (Dynamic):
    - Image/thumbnail.
    - Title (e.g., "Certified AI/ML Practitioner").
    - Rating (e.g., "4.8 Reviews").
    - Price (e.g., ₹19,999, discounted from ₹29,999).
    - Availability (e.g., "Only 5 Seats Available").
    - "Enquiry" button (links to Course Detail).
- **Pagination/Infinite Scroll**: For browsing multiple courses.
- **CTA**: "See All Courses" (if limited on homepage, redundant here).
- **Footer**: Same as landing page (links, contact info, newsletter).

Steps

1. **User Visits Course Page**:
    - Navigates to /courses (e.g., via nav link).
2. **Loads Course List**:
    - Frontend fetches and displays all courses by default.
3. **Applies Filters**:
    - User selects filters (e.g., "Technology" category, "₹10,000-₹20,000").
    - Frontend updates list dynamically.
4. **Searches for Course**:
    - User types in search bar (e.g., "Data Science").
    - Frontend filters results in real-time or queries backend.
5. **Selects a Course**:
    - Clicks "Enquiry" on a course card → Redirects to /courses/{id} (Course Detail).
6. **Browses More**:
    - Uses pagination or scrolls for additional courses.

Frontend

- **Framework**: React/Vue for dynamic rendering.
- **Components**:
    - CourseCard: Reusable card with image, title, price, etc.
    - FilterBar: Dropdowns/checklist for categories, price, sorting.
    - SearchBar: Real-time input with debounce.
- **Features**:

- Responsive grid layout (e.g., 3 columns desktop, 1 column mobile).
- Hover effects on cards.
- Infinite scroll or numbered pagination.

Backend

- **Endpoint**: GET /api/courses
  - Query Params:
    - category: Filter by category (e.g., "Technology").
    - priceMin, priceMax: Filter by price range.
    - sort: Sort by field (e.g., "price", "rating").
    - search: Search by title (e.g., "Data Science").
    - page, limit: Pagination (e.g., page 1, 10 items per page).
  - Response: { courses: [{ id, title, category, price, originalPrice, rating, seats, imageUrl }], total: number }.
- **Database**: Courses table (id, title, category, price, original_price, rating, seats_available, image_url, description).

---

2. Course Detail

Purpose: Provide in-depth information about a specific course, encouraging enrollment.

UI Components

- **Header**: Same as Course Page (navigation, search, login).
- **Course Hero**:
  - Image/banner (e.g., course-specific graphic).
  - Title (e.g., "Certified AI/ML Practitioner (CAMP)").
  - Rating (e.g., "4.8 Reviews").
  - Price (e.g., ₹19,999, strikethrough ₹29,999).
  - Availability (e.g., "Only 5 Seats Available").
  - "Enroll Now" button (leads to Checkout).
- **Tabs/Sections**:
  - **Overview**: Brief description (e.g., "Master AI/ML with hands-on projects").
  - **Curriculum**: List of modules (e.g., "1. Intro to AI, 2. Machine Learning Basics").
  - **Instructor**: Name, bio, photo (e.g., "Dr. John Doe, 10+ years in AI").
  - **Reviews**: Student testimonials (e.g., "Great course! Learned a lot.").
- **Details**:
  - Duration (e.g., "12 weeks").
  - Level (e.g., "Intermediate").
  - Certificate (e.g., "Online Certificate upon completion").
- **Related Courses**: Suggestions (e.g., "Full Stack Development", "Data Science").
- **CTA**: "Enroll Now" (repeated) + "Contact Us" (for inquiries).

- **Footer**: Same as landing page.

Steps

1. **User Visits Course Detail**:
    - Navigates to /courses/{id} (e.g., from Course Page "Enquiry" button).
2. **Loads Course Data**:
    - Frontend fetches and displays course details.
3. **Explores Tabs**:
    - Clicks through Overview, Curriculum, Instructor, Reviews.
4. **Decides to Enroll**:
    - Clicks "Enroll Now" → Redirects to /checkout?courseId={id}.
5. **Seeks Alternatives**:
    - Browses related courses → Redirects to other /courses/{id} pages.
6. **Contacts Support**:
    - Clicks "Contact Us" → Redirects to /contact-us.

Frontend

- **Framework**: React/Vue.
- **Components**:
    - CourseHero: Image, title, price, button.
    - Tabs: Switchable sections (Overview, Curriculum, etc.).
    - RelatedCourses: Mini course cards.
- **Features**:
    - Sticky "Enroll Now" button on scroll.
    - Responsive layout (tabs stack on mobile).
    - Star rating display (e.g., 4.8/5 with stars).

Backend

- **Endpoint**: GET /api/courses/{id}
    - Response: { id, title, category, price, originalPrice, rating, seats, imageUrl, description, duration, level, certificate, curriculum: [string], instructor: { name, bio, photo }, reviews: [{ studentName, text, rating }] }.
- **Related Courses**: GET /api/courses/related/{id}
    - Returns: { courses: [{ id, title, price, rating, imageUrl }] }.
- **Database**:
    - Courses table (as above).
    - Instructors table (id, name, bio, photo_url).
    - Reviews table (id, course_id, student_name, text, rating).

Complete Flow Summary

Course Page

1. Visit /courses → Browse courses → Apply filters/search → Click "Enquiry" → Go to Course Detail.

Course Detail

1. Visit /courses/{id} → View details (overview, curriculum, etc.) → Click "Enroll Now" → Go to Checkout OR explore related courses.

Team Implementation Guide

Frontend Team

- **Tasks**:
  - Build /courses page:
    - Filter bar with dynamic updates.
    - Course card grid with pagination/infinite scroll.
  - Build /courses/{id} page:
    - Hero section with enroll button.
    - Tabbed layout for course details.
    - Related courses section.
- **Priority**:
  - Course Page → Course Detail.

Backend Team

- **Tasks**:
  - Set up database:
    - Courses table with all fields (price, seats, etc.).
    - Link instructors and reviews to courses.
  - Implement APIs:
    - GET /api/courses (list with filters).
    - GET /api/courses/{id} (single course details).
    - GET /api/courses/related/{id} (related courses).
- **Priority**:
  - Course list API → Course detail API.
  -

9. Checkout

- **Type**: Dynamic
- **UI Components**:
  - Order summary (course, price).

    - Payment method selection, billing form.
- **Frontend**: Secure form, payment gateway UI (e.g., Stripe).
- **Backend**:
    - POST /api/checkout (process payment).

10. Registration (Students Only)

Purpose: Allow new students to create an account.

UI Components

- Form: Name, Email, Password, Confirm Password.
- CAPTCHA (e.g., reCAPTCHA) for bot protection.
- "Register" button.
- Link: "Already have an account? Login here."

Steps

1. **User Visits Registration Page**:
    - Navigates to /register (e.g., via "Become a Student" CTA on landing page).
2. **Fills Form**:
    - Enters Name (e.g., "Deepak Kumar"), Email (e.g., "deepak@example.com"), Password, Confirm Password.
    - Completes CAPTCHA.
3. **Submits Form**:
    - Clicks "Register".
4. **Frontend Validation**:
    - Checks: Email format, password strength (e.g., 8+ chars, mix of letters/numbers), passwords match.
    - If invalid, shows error (e.g., "Passwords don’t match").
5. **Backend Request**:
    - Sends POST /api/register with { name, email, password }.
6. **Backend Processing**:
    - Validates: Email not already in use.
    - Hashes password (e.g., bcrypt).
    - Creates student record in database (role: "student").
    - Generates OTP (e.g., 6-digit code like "123456").
    - Sends OTP to user’s email via email service (e.g., SendGrid).
7. **OTP Verification Screen**:
    - Frontend redirects to /verify-otp.
    - UI: "Enter the 6-digit OTP sent to your email" + input field + "Verify" button.
8. **User Enters OTP**:
    - Inputs OTP (e.g., "123456") and clicks "Verify".
9. **Backend OTP Check**:
    - Sends POST /api/verify-otp with { email, otp }.

    - Backend verifies OTP matches and hasn't expired (e.g., 5-min validity).
    - If valid, marks account as verified; if invalid, returns error ("Invalid or expired OTP").
10. **Success**:
    - Redirects to /login with message: "Registration successful! Please log in."

Frontend

- Framework: React/Vue.
- Form validation library (e.g., Formik).
- State management for OTP screen transition.

Backend

- Endpoints:
    - POST /api/register: { name, email, password } → Returns { message: "OTP sent" }.
    - POST /api/verify-otp: { email, otp } → Returns { message: "Verified" }.
- Database: Users table (id, name, email, password_hash, role, is_verified).
- Email service integration.

---

2. Login (Students + Teachers)

Purpose: Authenticate students (registered) and teachers (pre-assigned accounts).

UI Components

- Form: Email, Password.
- "Login" button.
- Links: "Forgot Password?" + "New user? Register here" (for students only).

Steps

1. **User Visits Login Page**:
    - Navigates to /login (e.g., via header button).
2. **Fills Form**:
    - Enters Email (e.g., "deepak@example.com") and Password.
3. **Submits Form**:
    - Clicks "Login".
4. **Frontend Validation**:
    - Checks: Email format, password not empty.
5. **Backend Request**:
    - Sends POST /api/login with { email, password }.
6. **Backend Processing**:

    - Checks: Email exists in database.
    - Verifies: Password matches hashed password.
    - Checks: Account is verified (for students).
    - Generates JWT token with user role (e.g., "student" or "teacher").
    - Returns { token, role }.
7. **Success**:
    - Frontend stores token (e.g., localStorage).
    - Redirects based on role:
        - Students → /student/dashboard.
        - Teachers → /teacher/dashboard.
8. **Failure**:
    - Returns error (e.g., "Invalid credentials") and displays on UI.

Frontend

- Simple form with error handling.
- Redirect logic based on role.

Backend

- Endpoint: POST /api/login → Returns { token, role }.
- JWT for session management.

---

3. Forgot Password

Purpose: Allow users (students or teachers) to reset their password.

UI Components

- Form: Email.
- "Send Reset Link" button.
- Subsequent screens: OTP input, New Password form.

Steps

1. **User Initiates Reset**:
    - Clicks "Forgot Password?" on /login → Redirects to /forgot-password.
2. **Enters Email**:
    - Inputs Email (e.g., "deepak@example.com") and clicks "Send Reset Link".
3. **Frontend Validation**:
    - Checks: Valid email format.
4. **Backend Request**:
    - Sends POST /api/forgot-password with { email }.
5. **Backend Processing**:

    - Checks: Email exists in database.
    - Generates OTP (e.g., "789012").
    - Sends OTP to email with message: "Your reset OTP is 789012".
    - Stores OTP temporarily (e.g., Redis, 5-min expiry).
6. **OTP Verification Screen**:
    - Redirects to /reset-password/otp.
    - UI: "Enter OTP sent to your email" + input field + "Verify" button.
7. **User Enters OTP**:
    - Inputs "789012" and clicks "Verify".
8. **Backend OTP Check**:
    - Sends POST /api/verify-reset-otp with { email, otp }.
    - Verifies OTP; if valid, returns { resetToken } (short-lived token for password reset).
9. **New Password Screen**:
    - Redirects to /reset-password.
    - UI: "New Password" + "Confirm New Password" + "Submit" button.
10. **User Submists New Password**:
    - Enters new password and submits.
11. **Backend Updates Password**:
    - Sends POST /api/reset-password with { resetToken, newPassword }.
    - Validates token, hashes new password, updates user record.
12. **Success**:
    - Redirects to /login with message: "Password reset successful! Please log in."

Frontend

- Multi-step form (email → OTP → new password).
- Error handling for invalid OTP/token.

Backend

- Endpoints:
    - POST /api/forgot-password: { email } → Returns { message: "OTP sent" }.
    - POST /api/verify-reset-otp: { email, otp } → Returns { resetToken }.
    - POST /api/reset-password: { resetToken, newPassword } → Returns { message: "Password updated" }.
- Temporary OTP storage (e.g., Redis).

---

4. OTP Verification (General Notes)

- **Purpose**: Used in Registration and Forgot Password for security.
- **Implementation**:
    - 6-digit numeric code (e.g., "123456").

    - Sent via email (or SMS if added later).
    - Expires in 5 minutes.
- **Backend Logic**:
    - Generate OTP: Random number or library (e.g., otp-generator in Node.js).
    - Store OTP: In-memory (Redis) or database with expiry timestamp.
    - Validate OTP: Match user input with stored value, check expiry.

Complete Flow Summary

Registration

1. Fill form → Submit → Receive OTP → Verify OTP → Account created → Login.

Login

1. Fill form → Submit → Authenticated → Redirect to dashboard (student/teacher).

Forgot Password

1. Enter email → Receive OTP → Verify OTP → Set new password → Login with new credentials.

Team Implementation Guide

Frontend Team

- **Tasks**:
    - Build /register page with form + OTP screen.
    - Build /login page with form + "Forgot Password?" link.
    - Build /forgot-password flow (email → OTP → reset password).
    - Use consistent styling (e.g., Material-UI, Tailwind).
- **Priority**:
    - Registration → Login → Forgot Password.

Backend Team

- **Tasks**:
    - Set up user database (students + pre-assigned teachers).
    - Implement APIs:
        - Registration: /api/register, /api/verify-otp.
        - Login: /api/login.

        - Forgot Password: /api/forgot-password, /api/verify-reset-otp, /api/reset-password.
    - Integrate email service (e.g., SendGrid) for OTP delivery.
    - Use JWT for authentication, Redis for OTP storage.
- **Priority**:
    - Registration APIs → Login API → Forgot Password APIs.
    -

12. Student Blog

- **Type**: Dynamic
- **UI Components**:
    - Blog post list (titles, thumbnails).
    - Search, category filters, comment section.
- **Frontend**: Blog card layout, comment UI.
- **Backend**:
    - GET /api/blogs/student (fetch student blogs).
    - POST /api/blogs/{id}/comments (post comments).

13. Industry Expert Blog (New Addition)

- **Type**: Dynamic
- **UI Components**:
    - Blog post list focused on industry insights (e.g., AI trends, career tips).
    - Author bio per post (e.g., expert name, credentials).
    - Search, category filters, comment section.
- **Frontend**:
    - Similar to Student Blog but with emphasis on expert authorship.
    - Professional layout, author profile snippets.
- **Backend**:
    - GET /api/blogs/expert (fetch expert blogs).
    - POST /api/blogs/{id}/comments (post comments).

Student Backend (Student Portal)

Overview

- **Purpose**: A personalized portal for students to manage their learning experience—tracking progress, accessing materials, attending classes, exploring career options, and interacting with instructors.
- **Access**: After successful login (/login), students are redirected to /student/dashboard.
- **Dynamic Nature**: All sections pull real-time data via APIs, tailored to the logged-in student.

---

1. Dashboard

- **Description**: The central hub where students get an overview of their learning journey and navigate to other features.
- **UI Components**:
    - **Welcome Message**: "Welcome, Deepak Kumar!" (fetched from user profile).
    - **Quick Stats**:
        - "Courses Enrolled: 2" (count of active courses).
        - "Overall Progress: 60%" (average completion across courses).
        - "Upcoming Classes: 3" (count of classes in the next 7 days).
    - **Sidebar Navigation**: Links to all sections (Learning History, Study Material, etc.).
    - **Recent Activity**: "Last watched: AI Basics Video - 10 mins ago" (timestamped list of recent actions).
- **Steps**:
    - Student enters email (e.g., "deepak@example.com") and password at /login.
    - On success, browser redirects to /student/dashboard.
    - Frontend sends GET /api/student/dashboard request.
    - Backend returns data, and UI renders stats and activity dynamically.
- **Frontend**:
    - Framework: React/Vue.
    - Layout: Card-based (e.g., 3 cards for stats), sidebar on the left (collapsible on mobile).
    - Libraries: Use a state manager (e.g., Redux) to store user data post-login.
    - Styling: Tailwind CSS for responsiveness (e.g., flex flex-col md:flex-row).
- **Backend**:
    - **Endpoint**: GET /api/student/dashboard
    - **Logic**:
        - Query database for user details (SELECT name FROM students WHERE id = ?).
        - Count enrolled courses (SELECT COUNT(*) FROM enrollments WHERE student_id = ?).
        - Calculate progress (SELECT AVG(progress) FROM progress WHERE student_id = ?).
        - Fetch upcoming classes (SELECT COUNT(*) FROM schedule WHERE date > NOW() AND course_id IN (SELECT course_id FROM enrollments WHERE student_id = ?)).
        - Get recent activity (SELECT action, timestamp FROM activity WHERE student_id = ? ORDER BY timestamp DESC LIMIT 5).
    - **Response**: { "name": "Deepak Kumar", "enrolledCourses": 2, "progress": 60, "upcomingClasses": 3, "recentActivity": [{ "action": "Watched AI Basics Video", "timestamp": "2025-03-19T10:00:00Z" }] }.
- **Database**:
    - students table: id, name, email, password_hash.
    - enrollments table: student_id, course_id.

    - progress table: student_id, course_id, progress.
    - activity table: student_id, action, timestamp.

---

2. Student Learning with Information History

- **Description**: Tracks the student's progress across enrolled courses with detailed history.
- **UI Components**:
    - **Course List**:
        - "AI/ML Practitioner - 75% complete" (progress bar).
        - "Data Science - 20% complete" (progress bar).
    - **Filters**: Dropdowns for "All", "In Progress", "Completed".
    - **Details**: "Completed on: Mar 15, 2025", "Grade: A" (if applicable).
- **Steps**:
    - Student clicks "Learning History" in sidebar → Redirects to /student/learning-history.
    - Page loads all enrolled courses with progress.
    - Student selects "Completed" filter → UI updates to show only finished courses.
    - Clicks a course (e.g., "AI/ML Practitioner") → Expands to show detailed history (e.g., module completion dates).
- **Frontend**:
    - Component: CourseProgressCard (title, progress bar, expandable details).
    - Filter logic: Client-side filtering or API call with query params (e.g., ?status=completed).
    - Animation: Smooth expand/collapse (e.g., CSS transition: height 0.3s).
- **Backend**:
    - **Endpoint**: GET /api/student/learning-history
    - **Logic**:
        - Fetch enrolled courses (SELECT course_id FROM enrollments WHERE student_id = ?).
        - Join with progress (SELECT c.title, p.progress, p.completion_date FROM courses c JOIN progress p ON c.id = p.course_id WHERE p.student_id = ?).
        - Filter by status if provided (WHERE p.progress = 100 for completed).
    - **Response**: { "courses": [{ "id": 1, "title": "AI/ML Practitioner", "progress": 75, "completionDate": null }, { "id": 2, "title": "Data Science", "progress": 20, "completionDate": null }] }.
- **Database**:
    - courses table: id, title, description.
    - progress table: student_id, course_id, progress, completion_date.

---

3. Study Material

- **Description**: Provides access to course-related resources (PDFs, slides, etc.).
- **UI Components**:
    - **Material List**:
        - "AI Basics - Lecture 1.pdf" (with file size, e.g., "2 MB").
        - "Data Science - Slides.pptx" (with download icon).
    - **Search Bar**: "Search materials..." (e.g., type "AI" to filter).
    - **Filter**: Dropdown for courses (e.g., "AI/ML Practitioner").
- **Steps**:
    - Student clicks "Study Material" → Redirects to /student/study-material.
    - Loads all materials for enrolled courses.
    - Types "AI" in search → List filters to AI-related materials.
    - Clicks download → File downloads (e.g., via <a href> or API).
- **Frontend**:
    - Component: MaterialItem (title, file type icon, download button).
    - Search: Debounced input (e.g., 300ms delay) to filter locally or call API.
    - Lazy loading: Load materials as user scrolls (if many files).
- **Backend**:
    - **Endpoint**: GET /api/student/study-material?courseId={id}
    - **Logic**:
        - Fetch materials for enrolled courses (SELECT m.id, m.title, m.type, m.url FROM materials m JOIN enrollments e ON m.course_id = e.course_id WHERE e.student_id = ? AND (m.course_id = ? OR ? IS NULL)).
        - Optional search param (WHERE m.title LIKE '%searchTerm%').
    - **Response**: { "materials": [{ "id": 1, "title": "Lecture 1.pdf", "type": "pdf", "url": "https://storage.example.com/lecture1.pdf" }] }.
- **Database**:
    - materials table: id, course_id, title, type, url.

---

4. Class Schedule

- **Description**: Displays upcoming and past class schedules with join options.
- **UI Components**:
    - **Calendar/List**:
        - "AI Basics - Mar 20, 2025, 10 AM" (with "Join" button if live).
        - "Data Science - Mar 15, 2025, 2 PM" (past, with recording link if available).
    - **Tabs**: "Upcoming", "Past".
- **Steps**:
    - Student clicks "Class Schedule" → Redirects to /student/schedule.
    - Default "Upcoming" tab shows future classes.
    - Switches to "Past" → Shows completed classes.
    - Clicks "Join" on a live class → Opens external link (e.g., Zoom).
- **Frontend**:

    - Component: ScheduleEvent (title, date, status, action button).
    - Library: FullCalendar for visual calendar or simple list with tabs.
    - Real-time: Poll API every minute to update "Live" status.
- **Backend**:
    - **Endpoint**: GET /api/student/schedule
    - **Logic**:
        - Fetch schedule for enrolled courses (SELECT s.id, s.title, s.date, s.join_url FROM schedule s JOIN enrollments e ON s.course_id = e.course_id WHERE e.student_id = ?).
        - Add status (isLive: NOW() BETWEEN date AND date + duration).
    - **Response**: { "events": [{ "id": 1, "title": "AI Basics", "date": "2025-03-20T10:00:00Z", "isLive": false, "joinUrl": "https://zoom.us/j/123" }] }.
- **Database**:
    - schedule table: id, course_id, title, date, duration, join_url.

---

5. Next Course Suggestion

- **Description**: Recommends courses based on current learning and interests.
- **UI Components**:
    - **Suggestions**:
        - "Data Science Basics - ₹19,999" (with description).
        - "Cyber Security Advanced" (with "Enroll Now" button).
- **Steps**:
    - Student clicks "Next Course" → Redirects to /student/next-course.
    - Sees tailored course suggestions.
    - Clicks "Enroll Now" → Redirects to /checkout?courseId={id}.
- **Frontend**:
    - Component: CourseCard (title, price, description, button).
    - Styling: Grid layout (e.g., 2-3 cards per row).
- **Backend**:
    - **Endpoint**: GET /api/student/next-course
    - **Logic**:
        - Analyze completed courses (SELECT course_id FROM progress WHERE student_id = ? AND progress = 100).
        - Suggest related courses (SELECT id, title, description, price FROM courses WHERE category IN (SELECT category FROM courses WHERE id IN (...)) AND id NOT IN (SELECT course_id FROM enrollments WHERE student_id = ?)).
    - **Response**: { "courses": [{ "id": 3, "title": "Data Science Basics", "description": "Learn data analysis", "price": 19999 }] }.
- **Database**:
    - courses table: id, title, category, description, price.

---

6. Job Opening Information

- **Description**: Lists job opportunities relevant to student's skills.
- **UI Components**:
    - **Job List**:
        - "Data Analyst - MNC, Noida" (with "Apply Now").
        - "AI Engineer - Bangalore".
    - **Filters**: Location, Role Type (e.g., "Full-time").
- **Steps**:
    - Student clicks "Job Openings" → Redirects to /student/jobs.
    - Browses jobs and applies filters (e.g., "Noida").
    - Clicks "Apply Now" → Opens external application link.
- **Frontend**:
    - Component: JobCard (title, company, location, button).
    - Filter bar: Dropdowns or checkboxes.
- **Backend**:
    - **Endpoint**: GET /api/student/jobs
    - **Logic**:
        - Fetch jobs (SELECT id, title, company, location, apply_url FROM jobs).
        - Filter by params (WHERE location = ? AND role_type = ?).
    - **Response**: { "jobs": [{ "id": 1, "title": "Data Analyst", "company": "MNC", "location": "Noida", "applyUrl": "https://mnc.com/apply" }] }.
- **Database**:
    - jobs table: id, title, company, location, role_type, apply_url.

---

7. Market Trends

- **Description**: Provides industry insights to guide career decisions.
- **UI Components**:
    - **Articles**: "AI Jobs Up 20% in 2025".
    - **Charts**: Bar graph showing job growth by sector.
- **Steps**:
    - Student clicks "Market Trends" → Redirects to /student/market-trends.
    - Reads articles and views charts.
- **Frontend**:
    - Component: TrendArticle (title, content, optional chart).
    - Library: Chart.js for visualizations.
- **Backend**:
    - **Endpoint**: GET /api/student/market-trends
    - **Logic**: Fetch static or admin-updated trends (SELECT id, title, content, chart_data FROM trends).

        - **Response**: { "trends": [{ "id": 1, "title": "AI Jobs Up 20%", "content": "AI demand is rising...", "chartData": { "labels": ["2024", "2025"], "values": [100, 120] } }] }.
- **Database**:
    - trends table: id, title, content, chart_data (JSON).

---

8. Recorded Videos

- **Description**: Access to past lecture recordings.
- **UI Components**:
    - **Video List**:
        - "AI Basics - Lecture 1" (50% watched).
        - "Data Science - Intro".
    - **Player**: Play, pause, progress bar.
- **Steps**:
    - Student clicks "Recorded Videos" → Redirects to /student/videos.
    - Selects a video → Plays with progress tracked.
    - Progress saves on pause/stop.
- **Frontend**:
    - Component: VideoItem (title, thumbnail, progress), VideoPlayer (controls).
    - Library: Video.js or HTML5 <video>.
- **Backend**:
    - **Endpoint**: GET /api/student/videos?courseId={id}
    - **Logic**:
        - Fetch videos (SELECT v.id, v.title, v.url, p.progress FROM videos v LEFT JOIN video_progress p ON v.id = p.video_id AND p.student_id = ? WHERE v.course_id = ?).
    - **Response**: { "videos": [{ "id": 1, "title": "Lecture 1", "url": "https://cdn.example.com/lecture1.mp4", "progress": 50 }] }.
- **Database**:
    - videos table: id, course_id, title, url.
    - video_progress table: student_id, video_id, progress.

---

9. Live Classes

- **Description**: Join scheduled live sessions.
- **UI Components**:
    - **Session List**:
        - "Cyber Security - Mar 21, 2025, 2 PM" (with "Join Now" if live).
- **Steps**:
    - Student clicks "Live Classes" → Redirects to /student/live-classes.
    - Sees upcoming live sessions.

    - Clicks "Join Now" when live → Opens link (e.g., Zoom).
- **Frontend**:
    - Component: LiveEvent (title, date, status, button).
    - Real-time: Poll API to update "Live" status.
- **Backend**:
    - **Endpoint**: GET /api/student/live-classes
    - **Logic**: Same as Schedule but filtered for live/upcoming (WHERE date >= NOW()).
    - **Response**: { "classes": [{ "id": 2, "title": "Cyber Security", "date": "2025-03-21T14:00:00Z", "joinUrl": "https://zoom.us/j/456", "status": "upcoming" }] }.
- **Database**:
    - schedule table (reused from Class Schedule).

---

10. Doubt Section

- **Description**: Ask and view doubts with teacher replies.
- **UI Components**:
    - **Form**: Textarea for "What is overfitting?" + submit button.
    - **Doubt List**:
        - "What is overfitting? - Deepak Kumar" → "It's when a model..." - Teacher Reply.
- **Steps**:
    - Student clicks "Doubt Section" → Redirects to /student/doubts.
    - Types and submits a doubt.
    - Views existing doubts and replies.
- **Frontend**:
    - Component: DoubtThread (question, replies, reply form).
    - Styling: Threaded layout (e.g., nested comments).
- **Backend**:
    - **Endpoints**:
        - GET /api/student/doubts → { "doubts": [{ "id": 1, "question": "What is overfitting?", "studentName": "Deepak", "replies": [{ "text": "It's...", "teacherName": "Dr. John" }] }] }.
        - POST /api/student/doubts → { "courseId": 1, "question": "What is overfitting?" }.
    - **Logic**:
        - Fetch doubts (SELECT d.id, d.question, s.name AS studentName, r.text, t.name AS teacherName FROM doubts d JOIN students s ON d.student_id = s.id LEFT JOIN replies r ON d.id = r.doubt_id LEFT JOIN teachers t ON r.teacher_id = t.id WHERE d.course_id IN (SELECT course_id FROM enrollments WHERE student_id = ?)).

        - Insert new doubt (INSERT INTO doubts (course_id, student_id, question) VALUES (?, ?, ?)).
- **Database**:
    - doubts table: id, course_id, student_id, question, created_at.
    - replies table: id, doubt_id, teacher_id, text.

Teacher Backend (Teacher Portal) Flow with Full Details

Overview

- **Purpose**: Teachers ke liye ek portal jahan woh apne courses manage kar sakein, students ke doubts solve kar sakein, aur apni teaching ko improve karne ke liye suggestions de sakein.
- **Access**: Login ke baad /teacher/dashboard pe redirect hota hai. Teachers pre-assigned hote hain, registration nahi karte.

---

1. Dashboard

- **Type**: Dynamic
- **UI Components**:
    1. Welcome message: "Welcome, Dr. John Doe!"
    2. Quick stats: "Courses Taught: 3", "Upcoming Classes: 2".
    3. Sidebar navigation: Profile, Study Material, Schedule, Suggestions, Market Analysis, Live Classes, Doubt Handling.
    4. Recent activity: "Last class: AI Basics - Mar 18, 2025".
- **Steps**:
    1. Teacher /login pe credentials daalta hai.
    2. Success pe /teacher/dashboard pe redirect hota hai.
    3. Dashboard personalized data load karta hai (stats, recent activity).
- **Frontend**: Card layout with sidebar, responsive design.
- **Backend**: GET /api/teacher/dashboard → Returns { name, coursesTaught, upcomingClasses, recentActivity }.

---

2. Teacher Profile

- **Type**: Dynamic
- **UI Components**:
    - Details: Name ("Dr. John Doe"), Bio ("10+ years in AI"), Photo.

    - Edit button: Bio ya photo update karne ke liye form.
- **Steps**:
    - Sidebar se "Profile" click → /teacher/profile.
    - Profile dekhta hai ya edit karta hai.
    - Save pe changes backend pe update hote hain.
- **Frontend**: Profile card, editable form with file upload for photo.
- **Backend**:
    - GET /api/teacher/profile → { id, name, bio, photoUrl }.
    - PUT /api/teacher/profile → Updates { bio, photoUrl }.

---

3. Study Material Upload Option

- **Type**: Dynamic
- **UI Components**:
    - Upload form: File input (PDFs, slides) + course selection dropdown.
    - Material list: "AI Basics - Lecture 1.pdf" with delete/edit buttons.
- **Steps**:
    - "Study Material" click → /teacher/study-material.
    - Naya material upload karta hai ya purana manage karta hai.
    - Upload success pe list update hoti hai.
- **Frontend**: File upload UI, table ya card list.
- **Backend**:
    - POST /api/teacher/study-material → { courseId, file } upload.
    - GET /api/teacher/study-material?courseId={id} → { materials: [{ id, title, url }] }.
    - DELETE /api/teacher/study-material/{id} → Material delete.

---

4. Class Schedule

- **Type**: Dynamic
- **UI Components**:
    1. Calendar ya list: "AI Basics - Mar 20, 2025, 10 AM".
    2. "Start Class" button jab live ho.
    3. Past classes ka history.
- **Steps**:
    1. "Class Schedule" click → /teacher/schedule.
    2. Schedule dekhta hai ya live class start karta hai.
- **Frontend**: Calendar (e.g., FullCalendar), event buttons.
- **Backend**: GET /api/teacher/schedule → { events: [{ id, title, date, joinUrl, status }] }.

---

5. Teachers Suggestion

- **Type**: Dynamic
- **UI Components**:
    - Form: "Course Title: Data Science Advanced", description field.
    - Suggestion list: Past suggestions with status ("Pending", "Approved").
- **Steps**:
    - "Suggestions" click → /teacher/suggestions.
    - Naya course suggest karta hai ya status check karta hai.
- **Frontend**: Form, status table.
- **Backend**:
    - POST /api/teacher/suggestions → { title, description }.
    - GET /api/teacher/suggestions → { suggestions: [{ id, title, status }] }.

---

6. Market Analysis

- **Type**: Dynamic
- **UI Components**:
    1. Insights: "AI skills demand up 15%".
    2. Charts ya stats for course planning.
- **Steps**:
    1. "Market Analysis" click → /teacher/market-analysis.
    2. Trends padhta hai teaching improve karne ke liye.
- **Frontend**: Article list, chart library (e.g., Chart.js).
- **Backend**: GET /api/teacher/market-analysis → { trends: [{ id, title, content, chartData }] }.

---

7. Live Classes

- **Type**: Dynamic
- **UI Components**:
    1. Upcoming sessions: "Cyber Security - Mar 21, 2025, 2 PM".
    2. "Start Now" button jab live ho.
- **Steps**:
    1. "Live Classes" click → /teacher/live-classes.
    2. Live session start karta hai.
- **Frontend**: Event list, real-time status.
- **Backend**: GET /api/teacher/live-classes → { classes: [{ id, title, date, joinUrl, status }] }.

---

8. Doubt Handling

- **Type**: Dynamic

- **UI Components**:
    - Doubt list: "What is overfitting?" - Deepak Kumar.
    - Reply form har doubt ke liye.
- **Steps**:
    - "Doubt Handling" click → /teacher/doubts.
    - Students ke doubts dekhta hai aur reply karta hai.
- **Frontend**: Threaded UI, reply form.
- **Backend**:
    - GET /api/teacher/doubts → { doubts: [{ id, question, studentName, replies }] }.
    - POST /api/teacher/doubts/{id}/reply → { reply }.