Technical Requirements:

Frontend Development:

- **Framework:** Next.js (for SEO, performance, and SEO Friendly search engine optimization best tool next Js).

- **Styling:** Tailwind CSS (for responsiveness and ease of customization).

Components:

1. Header with navigation.
2. Hero section for promotions/highlights.
3. Course catalog with filtering and sorting options.
4. Individual course detail pages.
5. User dashboard for enrolled courses.

Features:

6. Responsive design for all devices.
7. Accessibility compliance (ARIA, semantic HTML).
8. Smooth navigation with dynamic routing.

Backend Development:

1.Node js

2. Next.js API routes

Database:

9. Sanity (as your CMS/database to manage course content).

- API Design: RESTful or GraphQL APIs for:

1. User authentication.
2. Course listing and details.
3. Payment processing integration.
4. Content delivery.

Authentication & Authorization

5. Methods:
    1. OAuth (Google, GitHub, etc.) for quick sign-up/sign-in.
    2. Custom email/password authentication.
6. Role Management:

    1. Admin: Manage courses, users, and payments.
    2. Users: Access free/paid courses.

Payment Gateway Integration

7. Options:
    1. Jazzcash or easypias for paid course purchases.
8. Features:
    1. Handle one-time payments and invoices.
    2. Support multiple currencies.

5. Content Management

- Use Sanity for:

9. Adding/updating course details (titles, descriptions, videos, and PDFs).
10. Managing categories and tags for courses.

- Integration with the frontend for real-time updates.

User Dashboard

11. Features:
    1. View enrolled courses.
    2. Track course progress.
    3. Access certificates of completion (if applicable)

plan API requirements:

. Fetch All Courses

- **Endpoint**: `/api/courses`
- **Description**: Retrieve a list of all courses (both free and paid).
- **Meth**`Courses`

Query Parameters:

- `category` (optional): Filter courses by category (e.g., `coding`, `graphic design`).
- `price` (optional): Filter by price range or free courses (`free` or `paid`).
- `sort` (optional): Sort by `popular`, `new`, or `price`.
- `page` (optional): Pagination for large datasets.

Response:

```json
{
  "success": true,
  "data": [
    {
      "id": "course123",
      "title": "Introduction to Graphic Design",
      "description": "Learn the basics of graphic design...",
      "category": "Graphic Design",
      "price": 0,
      "rating": 4.8,
      "instructor": "Hassan "
    }
  ],
  "pagination": {
    "currentPage": 1,
    "totalPages": 10
  }
}
```

Fetch a Single Course

- **Endpoint**: `/api/courses/:id`
- **Description**: Retrieve details of a specific course by its ID.
- **Method**: `GET`
- **Path Parameters**:
    - `id` (required): The unique ID of the course.

```json
{
  "success": true,
  "data": {
    "id": "course123",
    "title": "Introduction to Graphic Design",
    "description": "Learn the basics of graphic design...",
    "category": "Graphic Design",
    "price": 0,
    "content": [
      { "id": "lesson1", "title": "Lesson 1: Basics", "duration": "15m" },
      { "id": "lesson2", "title": "Lesson 2: Tools", "duration": "20m" }
    ],
    "instructor": {
      "name": "hassan",
      "bio": "Expert designer with 10+ years of experience.",
      "rating": 4.8
    }
  }
}
```

Technical Documentation

System Architecture Overview

- **Frontend:** Built using **Next.js** for fast performance and server-side rendering.
- **Backend:** Using **Sanity CMS** for content management (courses, categories, user data).
- **Database:** Sanity manages schemas for structured data storage.
- **API:** RESTful APIs or GraphQL to fetch data dynamically for course details, user accounts, and payments.
- **Authentication:** Secure login using JWT tokens for access to paid courses.
- **Deployment:** Hosted on Vercel for seamless integration with Next.js.

Key Workflows

1. **User Browsing Courses:**
   - Workflow:
     - The user visits the homepage → selects a course → views course details.
   - Interaction:
     - Data fetched from `/courses` endpoint (GET request).
2. **User Purchasing Courses:**
   - Workflow:
     - The user selects a paid course → proceeds to checkout → completes payment.
   - Interaction:

        - Payment details sent to `/payment` endpoint (POST request).
        - Course access updated in the database.

3. **Admin Adding Courses:**
    - Workflow:
        - Admin logs into Sanity CMS → uses a schema form to add course details → updates course list.
    - Interaction:
        - Course schema updates Sanity database automatically.

---

2. API Endpoints

Endpoint	Method	Purpose	Response Example
`/courses`	GET	Fetches all course details	`[ { "id": 1, "name": "Graphic Design", "price": 100 } ]`
`/course/:id`	GET	Fetches specific course details	`{ "id": 1, "name": "Graphic Design", "price": 100 }`
`/auth/login`	POST	Handles user login	`{ "token": "abc123" }`
`/auth/signup`	POST	Handles user signup	`{ "message": "Signup Successful" }`
`/payment`	POST	Processes payment for paid courses	`{ "status": "success", "courseId": 1 }`

---

3. Sanity Schema Example

```
export default {
  name: 'course',
  type: 'document',
  fields: [
    { name: 'title', type: 'string', title: 'Course Title' },
    { name: 'description', type: 'text', title: 'Course Description' },
    { name: 'price', type: 'number', title: 'Course Price' },
    { name: 'isPaid', type: 'boolean', title: 'Is this a Paid Course?' },
    { name: 'category', type: 'string', title: 'Category' },
    { name: 'duration', type: 'string', title: 'Duration (in hours)' }
  ]
};
```

---

4. Collaboration and Refinement

Group Discussions

- Organize brainstorming sessions using **Google Meet** to discuss API designs.
- Identify:
    - How to structure user authentication.
    - Payment gateway options.

Peer Review

- Share API designs and schema drafts with team members.
- Review course schema for missing fields (e.g., course ratings or user feedback).

---

5. Version Control

- Use GitHub to:
    - Create a separate branch for each module (e.g., `auth`, `courses`, `payment`).
    - Merge branches to maintain a clean `main` branch after peer review.
- Ensure commits follow clear messages like:
    - `feat: Added API endpoint for fetching courses`
    - `fix: Resolved bug in user login flow`

---

6. Key Outcomes

- A detailed technical foundation document.
- Clear API design aligned with marketplace goals.
- Structured Sanity schemas for easy course management.
- Improved teamwork and transparency using GitHub and peer review.
