# LearnNova Website

## Project Overview

> **LearnNova** is an e-learning platform designed to empower students with access to high-quality educational content across various fields. The platform provides a user-friendly interface and an interactive learning experience. Students can explore available courses, enroll, and track their progress, while administrators have access to a dedicated dashboard to efficiently manage course content.

---

# Database Schema Details

### Users Table

- **Fields**:
  - `id` (Primary Key, Auto Increment)
  - `name`: User's full name
  - `email`: Unique email address
  - `password`: Hashed password
  - `phone`: User's phone number
  - `role`: Enum ('user', 'admin')
  - `is_deleted`: Boolean (default: false)

### Categories Table

- **id** (Primary Key, Auto Increment)
- **name**: Name of the category (e.g., "Programming", "Design", "Data Science")
- **description**: A brief description of the category (e.g., "Courses related to programming and development")
- **is_deleted**: Boolean (default: false) – Marks if the category is logically deleted but still exists in the database
- **created_at**: Timestamp – The date and time when the category was created
- **updated_at**: Timestamp – The date and time when the category was last updated

## Courses Table

- **Fields**:
  - `id` (Primary Key, Auto Increment)
  - `title`: Name of the course
  - `description`: Detailed course description
  - `category_id`: Foreign Key (references `categories.id`)
  - `price`: Decimal (default 0.00)
  - `image`: URL to course image
  - `teacher_id`: Foreign Key (references `teachers.id`)
  - `is_deleted`: Boolean (default: false)
  - `duration`: VARCHAR (e.g., "3 hours", "4 weeks", "2 months")
  - `created_at`: Timestamp
  - `updated_at`: Timestamp
  - `category_id`: Foreign Key (references `categories.id`)

## Teachers Table

- **Fields**:
  - `id` (Primary Key, Auto Increment)
  - `user_id`: Foreign Key (references `users.id`)
  - `bio`: Biography of the teacher
  - `profile_picture`: URL to teacher's profile picture

- **facebook_link**: URL to teacher's Facebook profile
- **linkedin_link**: URL to teacher's LinkedIn profile
- **twitter_link**: URL to teacher's Twitter profile

### Cart Table

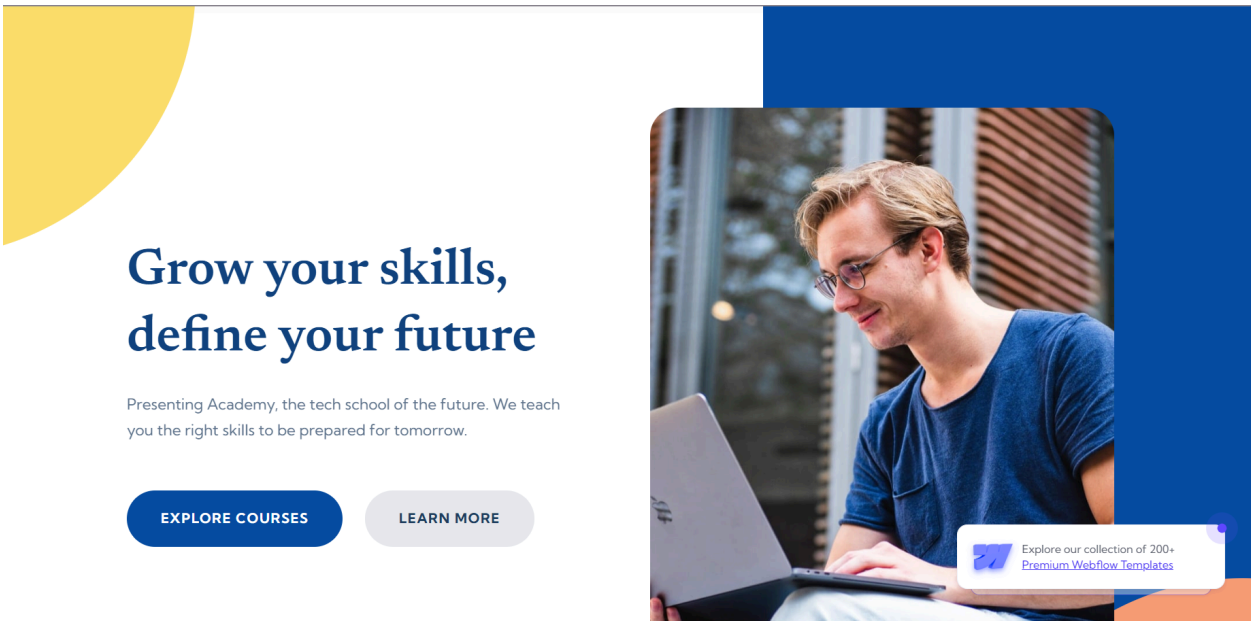
- **Fields**:
  - `id` (Primary Key, Auto Increment)
  - `user_id`: Foreign Key (references `users.id`)
  - `items`: Array of items containing:
    - `product_id`: Foreign Key (references `courses.id`)
    - `quantity`: Integer (required)
    - `price`: Decimal (required)
  - `total_price`: Decimal (calculated from items)
  - `is_deleted`: Boolean (default: false)
  - `created_at`: Timestamp

---

# Pages

## 1 - Home Page

- Header ⇒ (logo / Home / About / courses / Teachers / sign up and login / cart )
- go to courses page




- **Browse our popular courses ⇒ from backend**
-

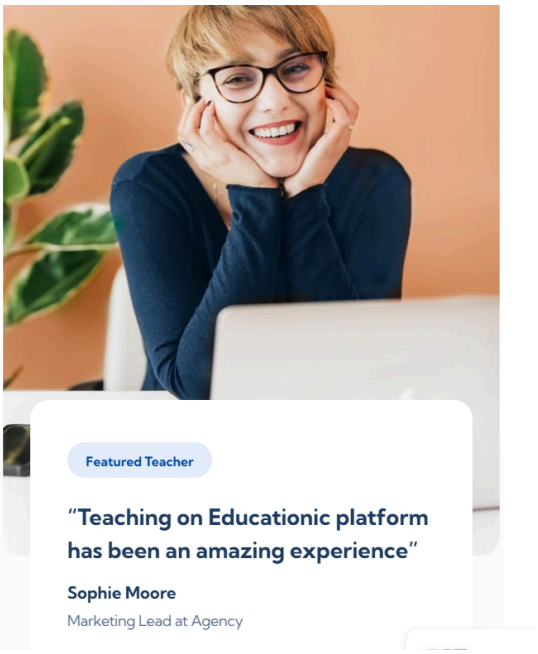

- go to teachers page

## Courses taught by industry leaders around the world

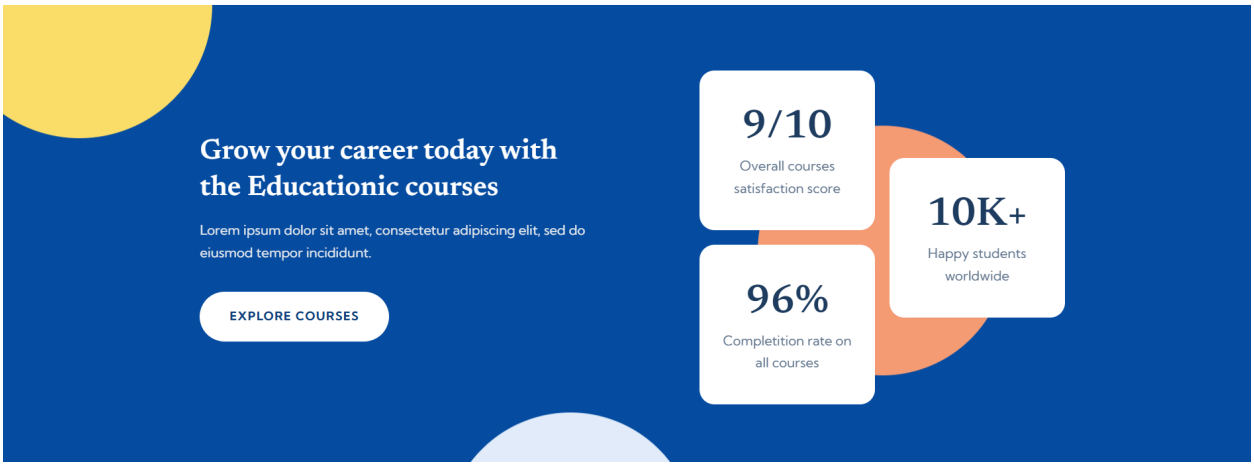
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

BROWSE TEACHERS

BECOME A TEACHER




- go to courses page




- go to about page

## About Education

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.

**Industry expert teachers**

Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit. Felis donec massa aliquam id dolor .

**Up-to-date course content**

Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit. Felis donec massa aliquam id dolor .

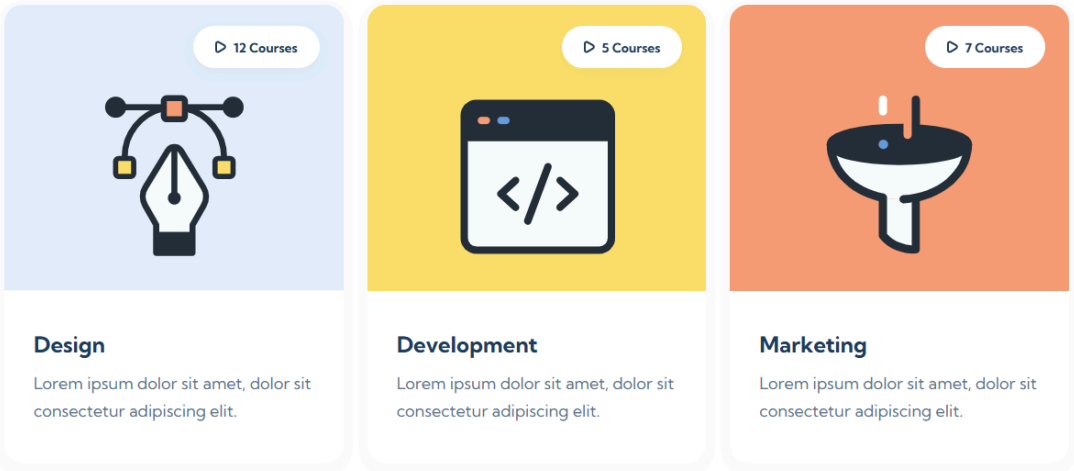
**Students community**

Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit. Felis donec massa aliquam id dolor .

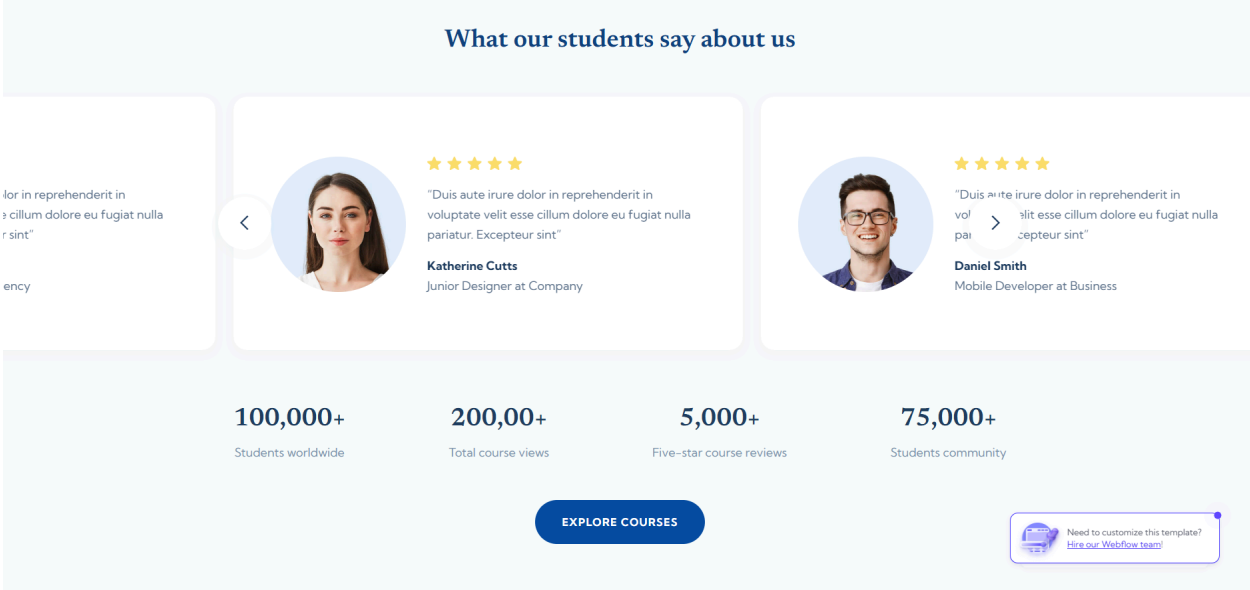
ABOUT EDUCATION

- go to multiple pages to specific category from cousers

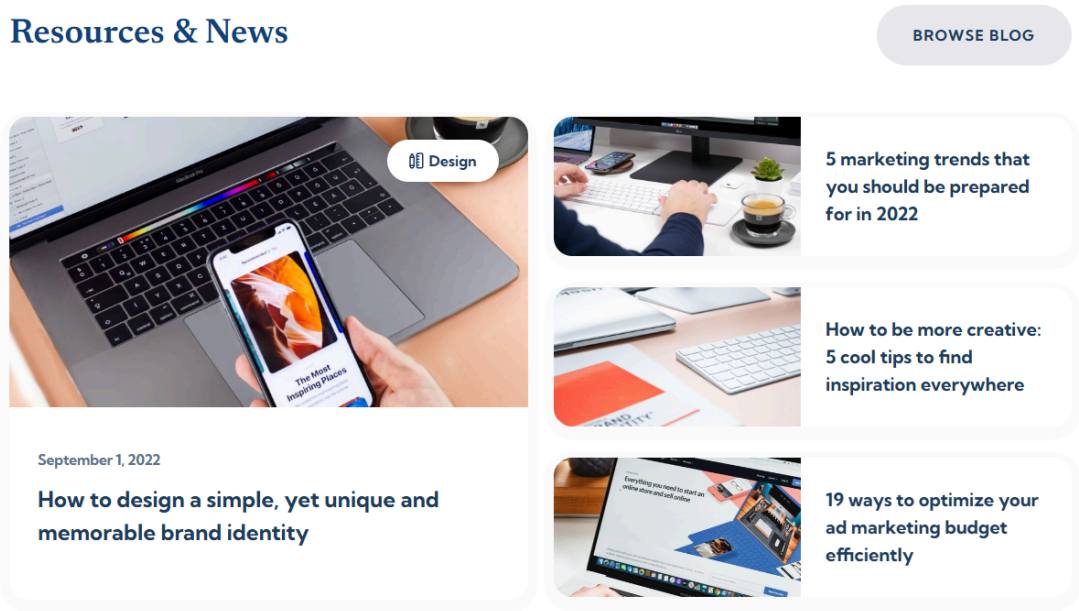
## Browse our courses by category



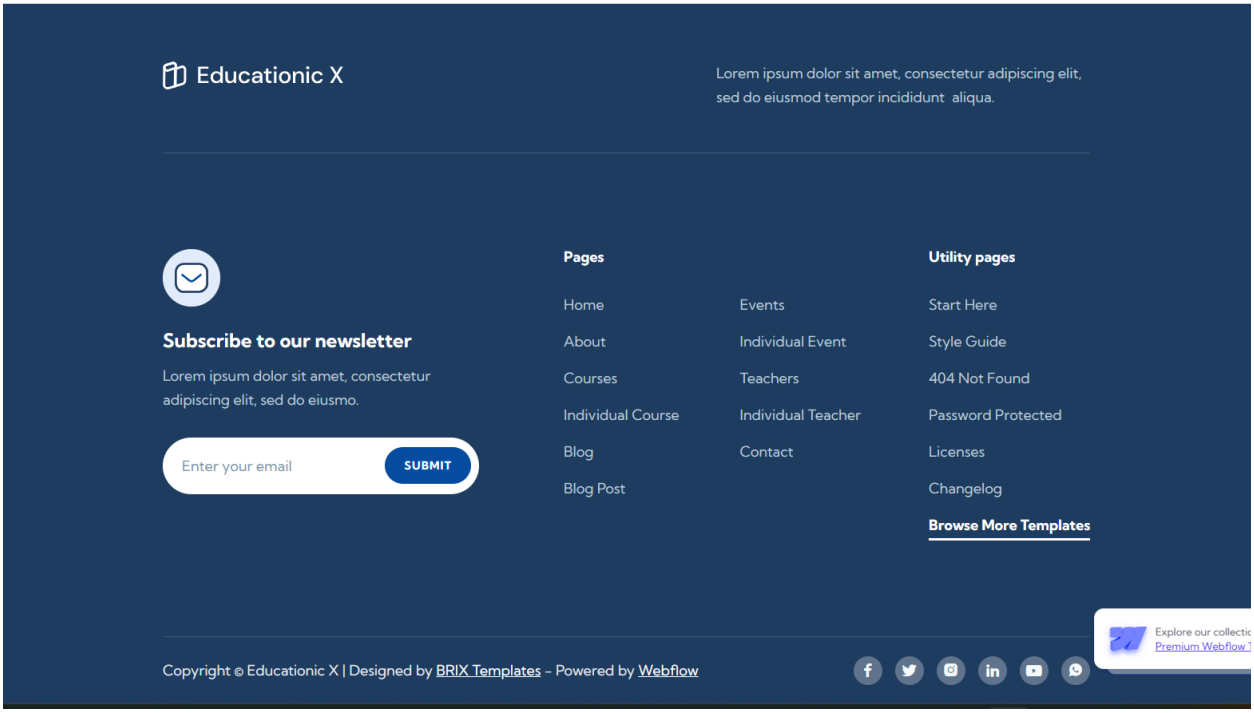
- just dummy data to show only not from backend




- all without browse blog

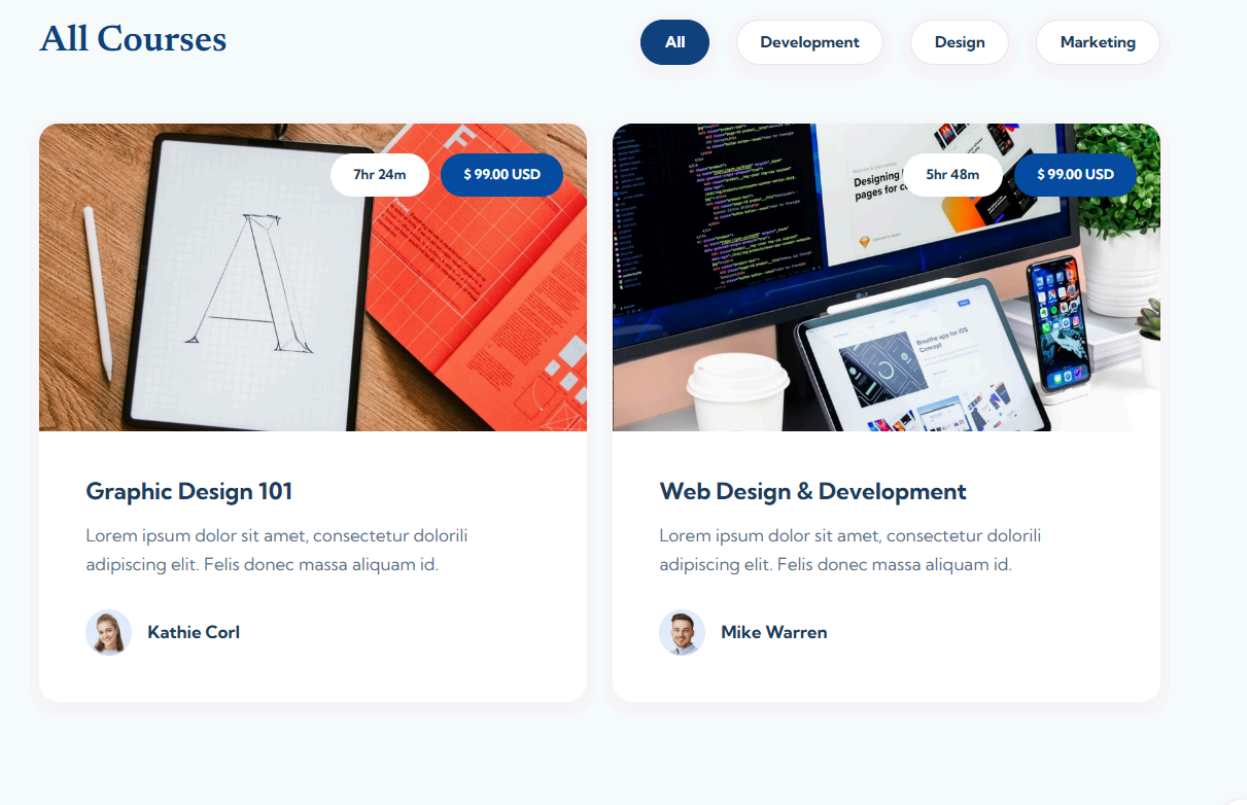



- Finally Footer




# 2 - Courses Page

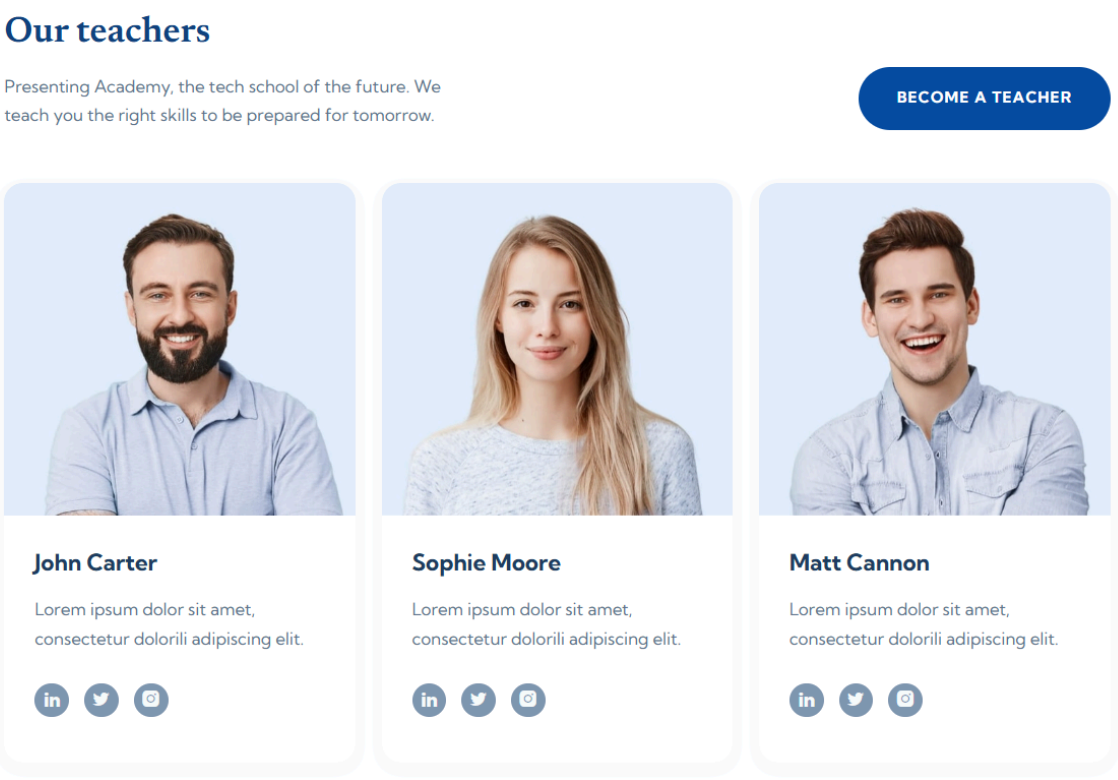
- **Browse our popular courses**
- All cousesrs by category and each category have page for it




- footer

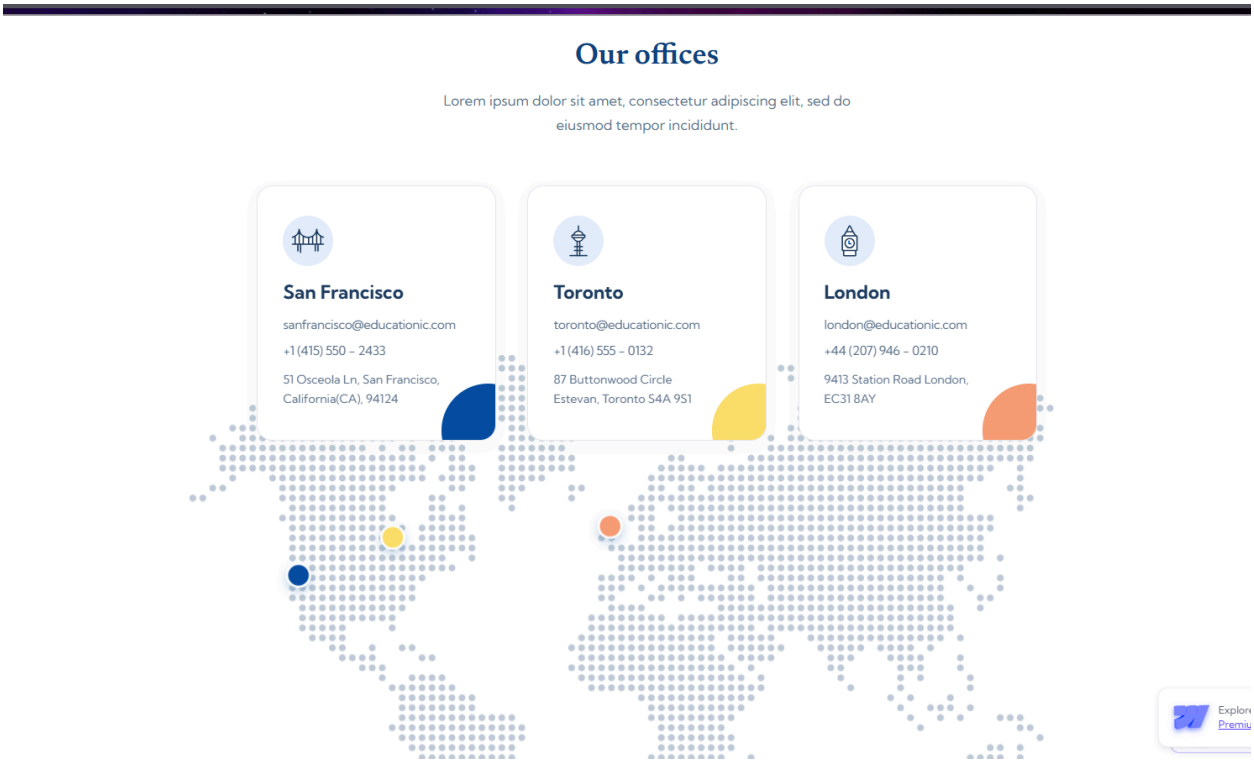
# 3 - Teachers Page

- bacome teacher ⇒ sign up as admin role ⇒ backend




- just dummy data not from backend

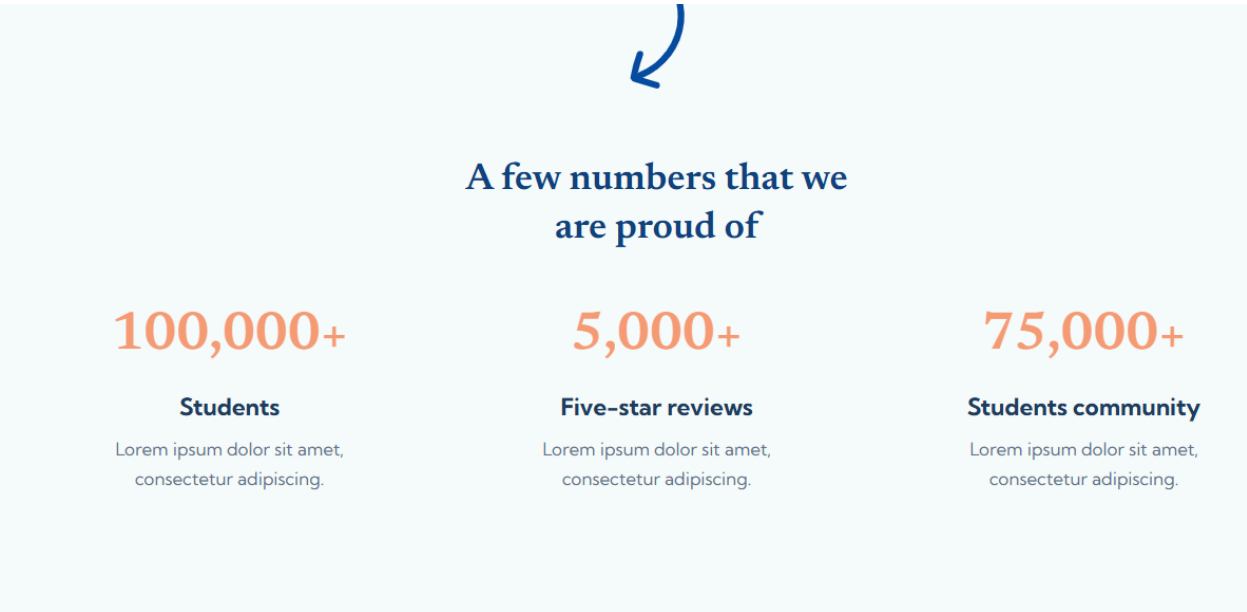

- we use here google maps Api to show map



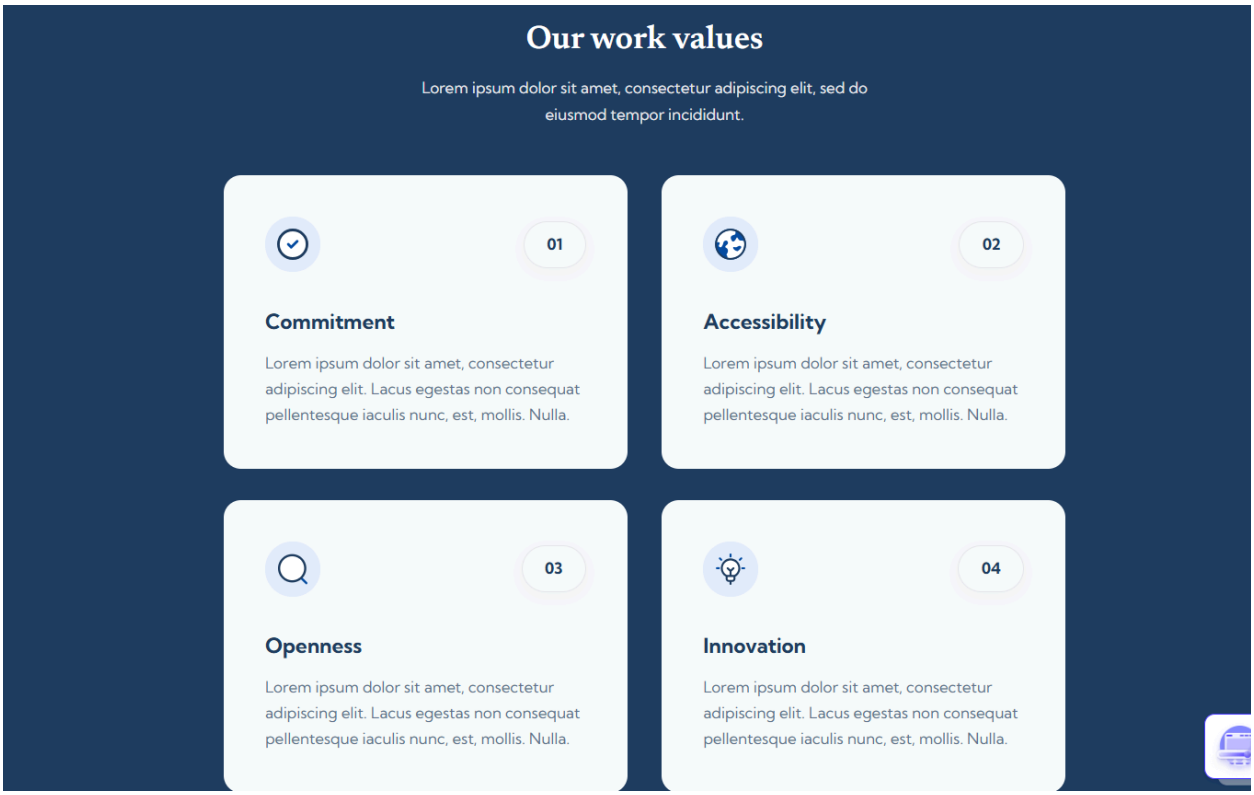
- Footer

# 4 - About Page

-




- **The mission behind Education platform**
- dummy data


Our work values
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt.
01
Commitment
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lacus egestas non consequat pellentesque iaculis nunc, est, mollis. Nulla.
02
Accessibility
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lacus egestas non consequat pellentesque iaculis nunc, est, mollis. Nulla.
03
Openness
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lacus egestas non consequat pellentesque iaculis nunc, est, mollis. Nulla.
04
Innovation
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lacus egestas non consequat pellentesque iaculis nunc, est, mollis. Nulla.

## Our teachers

Presenting Academy, the tech school of the future. We teach you the right skills to be prepared for tomorrow.

BECOME A TEACHER

### John Carter

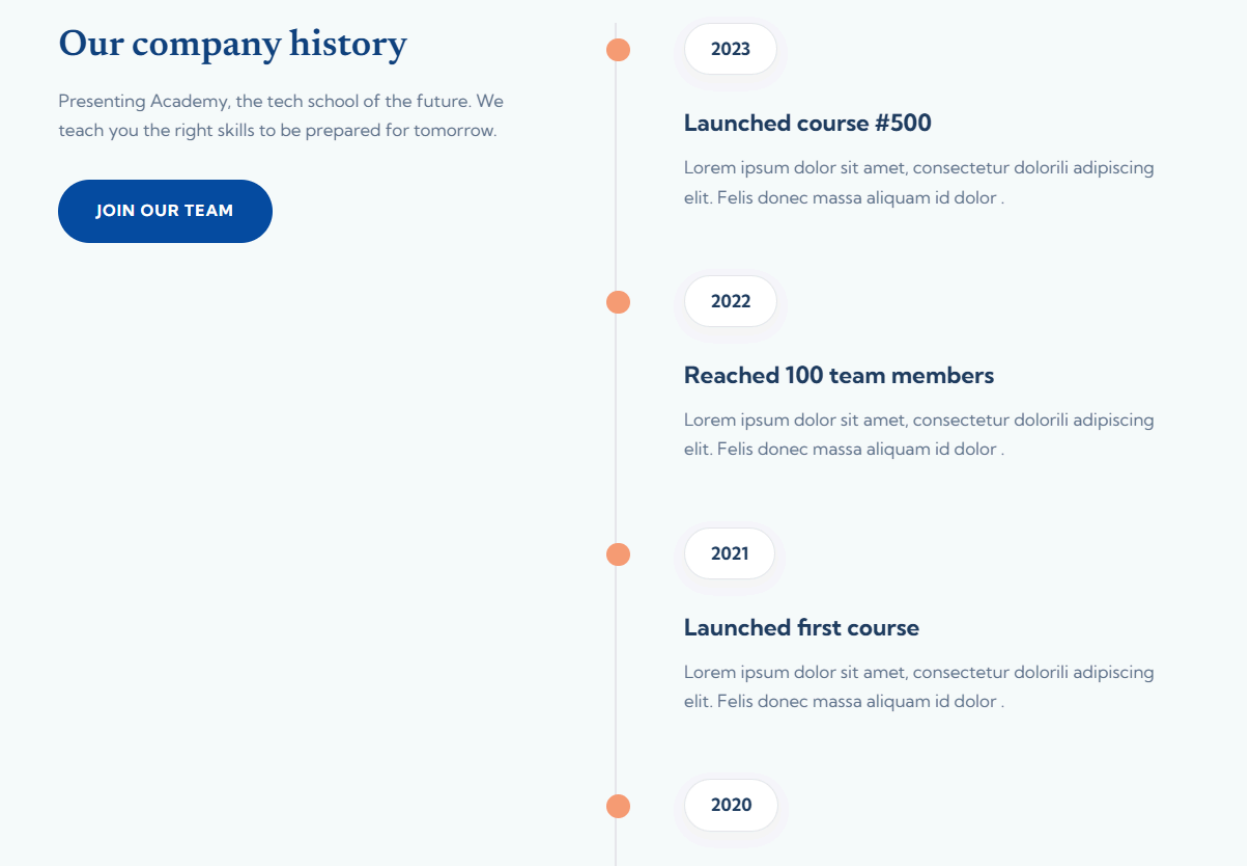
Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit.

### Sophie Moore

Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit.

### Matt Cannon

Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit.


Our company history
Presenting Academy, the tech school of the future. We teach you the right skills to be prepared for tomorrow.
JOIN OUR TEAM
2023
Launched course #500
Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit. Felis donec massa aliquam id dolor .
2022
Reached 100 team members
Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit. Felis donec massa aliquam id dolor .
2021
Launched first course
Lorem ipsum dolor sit amet, consectetur dolorili adipiscing elit. Felis donec massa aliquam id dolor .
2020


- footer

---

# 5 - Sign UP page

**Purpose**: Allow new users to create an account on the platform.

**Fields**:

- **Full Name**: User's first and last name
- **Email**: User's email address (must be unique)
- **Password**: User's password (ensure it’s hashed and secure)
- **Confirm Password**: User re-enters their password for verification
- **Phone Number**: Optional field for phone number (can be useful for contact or recovery)
- **Role**: Dropdown or radio button to select the role (e.g., "User", "Teacher")

- **Profile Picture**: Optional field to upload a profile picture (may allow users to upload a photo for personalization)

**Actions**:

- **Sign Up Button**: Submits the registration form and creates the user account
- **Validation**:
  - Ensure the email is unique
  - Password and confirm password should match
  - Validations for email format and password strength

**Additional Features**:

- **Password Strength Indicator**: A visual cue to show password strength (optional)
- **Terms and Conditions Checkbox**: Users must agree to the terms of service before signing up

---

# 6 - Login Page

**Purpose**: Allow registered users to log into their accounts.

**Fields**:

- **Email**: User's email address
- **Password**: User's password (hashed and verified against the stored credentials)

**Actions**:

- **Login Button**: Submits the login form to authenticate the user

**Validation**:

- **Email Format Validation**: Ensure the email follows a valid email format
- **Error Handling**: Show clear error messages if the email or password is incorrect

**Additional Features**:

- **Remember Me Checkbox**: Option to remember the user’s login details for future sessions (sessions)

---

```
/learnnova-frontend
│
├── /assets              # Contains assets like images, icons, and font files.
│   ├── /images          # Project images (like course images, teacher profile pictures, etc.).
│   ├── /icons           # Icons used throughout the site.
│   └── /fonts           # Font files used in the project.
│
├── /public              # Public files that are directly served (like the HTML file).
│   ├── index.html       # Main HTML file for the project.
│   ├── favicon.ico      # The icon displayed in the browser tab.
│   └── /css             # Custom CSS files (if any additional custom styles are required).
│
├── /src                 # Main source files (JavaScript logic, components, etc.).
│   ├── /assets          # Internal assets like images.
│   ├── /components      # Reusable UI components.
│   │   ├── navbar.js    # Navigation bar component.
│   │   ├── courseCard.js  # Course card component (used to display individual courses).
│   │   ├── footer.js    # Footer component.
│   │   └── button.js    # Button component (for reusable buttons like "Sign Up" or "Add to
Cart").
│   │
│   ├── /pages           # Main pages of the website.
│   │   ├── home.js      # Homepage with featured courses or introductory content.
│   │   ├── courses.js   # Page displaying a list of courses.
│   │   ├── login.js     # Login page with the form for user credentials.
│   │   ├── signup.js    # Sign-up page for new user registration.
│   │   └── cart.js      # Shopping cart page showing the selected courses.
│   │
│   ├── /services        # Logic for interacting with the backend (API calls).
│   │   └── api.js       # Functions to handle API requests like login, registration, fetching
courses.
│   │
│   ├── /utils           # Helper functions (like form validation, utility functions).
│   │   └── validation.js  # Functions to validate user inputs (email format, password
strength, etc.).
│   │
│   ├── /config          # Configuration files (like Tailwind setup, other settings).
│   │   └── tailwind.config.js  # Tailwind CSS customization file (to add new colors, spacing,
```

etc.).
│   │
│   ├── index.js          # Main JavaScript file for the application (entry point).
│   └── app.js            # The core JavaScript logic for the app (if needed for routing or app-wide functions).
│
└── /dist                 # Distribution folder for build files (generated after running the build command).
└── /css              # Final CSS files after Tailwind is processed and any custom CSS is applied.
└── /js               # Final bundled JavaScript files.

---

/learnnova-backend
│
├── /public               # Publicly accessible files (like the entry point for the application).
│   ├── index.php           # The entry point to handle API requests and route them accordingly.
│   └── /assets             # Public assets (images, CSS files, etc.) that need to be served to the client.
│
├── /src                  # Core application logic.
│   ├── /controllers        # Controllers to handle incoming HTTP requests and provide responses.
│   │   ├── AuthController.php      # Handles user login, sign-up, and authentication.
│   │   ├── CourseController.php    # Manages course-related actions (like displaying courses, adding courses).
│   │   ├── UserController.php      # Manages user-related actions (like viewing profile, updating details).
│   │   ├── CartController.php      # Handles cart actions (adding courses to cart, removing, checkout).
│   │   └── TeacherController.php  # Manages teacher-related actions (getting teacher info, adding new teacher).
│   │
│   ├── /models             # Contains the data models that interact with the database.
│   │   ├── User.php          # Represents the User model and its interaction with the database.
│   │   ├── Course.php        # Represents the Course model (course details, assignments,

```
etc.).
│   │   ├── Cart.php          # Represents the Cart model (items, total price).
│   │   ├── Teacher.php       # Represents the Teacher model.
│   │   └── Category.php      # Represents the Course Category model.
│   │
│   ├── /middlewares         # Contains logic for handling middleware functions
(authentication, validation).
│   │   ├── AuthMiddleware.php    # Validates authentication for protected routes.
│   │   └── ValidationMiddleware.php  # Validates input data for each request (like form
submissions).
│   │
│   ├── /services            # Business logic and services for handling core functionality.
│   │   ├── AuthService.php    # Handles authentication services (login, register).
│   │   ├── CourseService.php  # Business logic related to courses.
│   │   ├── CartService.php    # Logic for handling cart actions.
│   │   ├── TeacherService.php # Logic for teacher management.
│   │   └── CategoryService.php  # Logic for managing course categories.
│   │
│   ├── /helpers             # Helper classes or utility functions used throughout the app.
│   │   ├── Database.php      # Contains the database connection logic.
│   │   └── ResponseHelper.php # Standardizes the API response format.
│   │
│   ├── /config              # Configuration files for setting up constants, API keys, DB
connection, etc.
│   │   └── config.php        # Main configuration file for the app (like database, environment
settings).
│   │
│   ├── /routes              # Routing definitions (map URLs to controllers).
│   │   └── api.php           # Define the routes for different parts of the API (user, courses,
etc.).
│   │
│   └── /database            # Database logic (migrations, seeds, etc.).
│       ├── migration.php      # Database migration file (e.g., creating tables).
│       └── seed.php          # Seed data for initial setup of the database.
│
├── /tests                  # Unit tests to ensure your code works correctly.
│   ├── /unit_tests          # Unit tests for individual classes and functions.
│   └── /integration_tests    # Tests for complete workflows and integrations.
│
```

```
├── /logs                # Logs for the application (errors, user activities, etc.).
│   └── app.log          # Log file for application-related logs.
│
├── .gitignore           # Git ignore file to exclude unnecessary files from version control.
├── composer.json        # PHP dependencies file (if using Composer).
└──
README.md                # Documentation for your backend API setup, structure, and usage.
```

# STAR Method For Project

### Situation:

We have a team of 3 members working on building an educational platform called "LearnNova." However, we need to define a clear development plan to distribute tasks and coordinate the work between the frontend and backend teams.

### Task:

The team needs to create the core functionality of the website, enabling users to register, add courses, and manage content in an easy and secure manner.

### Action:

- **Frontend**: Build the user interface using **Tailwind CSS** and **JavaScript Native**. Design responsive user interfaces that work seamlessly across all devices.
- **Backend**: Set up the database using **PHP Native** and develop a **RESTful API** to manage users and courses.
- **Management**: Track the progress through weekly meetings and ensure continuous communication among the team.

### Result:

The result will be a fully integrated educational platform that offers a seamless user experience, allowing users to access courses, register, and track progress. Teachers will also be able to add and manage their own courses.

| Endpoint | Method | Description | Request Parameters | Response |
|---|---|---|---|---|
| **/api/users/register** | POST | User registration endpoint. Allows users to create a new account. | `name`, `email`, `password`, `phone` | `{ "message": "User registered successfully", "user": { ... } }` |
| **/api/users/login** | POST | User login endpoint. Allows users to log into their account. | `email`, `password` | `{ "message": "Login successful", "token": "JWT_token" }` |
| **/api/courses** | GET | Retrieves all available courses. | None | `[ { "id": 1, "title": "Course 1", "description": "...", ... } ]` |
| **/api/courses/{id}** | GET | Retrieves a single course by ID. | `id` (course ID) | `{ "id": 1, "title": "Course 1", "description": "...", ... }` |
| **/api/courses/create** | POST | Teacher creates a new course. | `title`, `description`, `category_id`, `price`, `image`, `teacher_id` | `{ "message": "Course created successfully", "course": { ... } }` |
| **/api/courses/{id}/update** | PUT | Update course details by ID. | `id` (course ID), `title`, `description`, `price`, `image` | `{ "message": "Course updated successfully", "course": { ... } }` |
| **/api/courses/{id}/delete** | DELETE | Deletes a course by ID. | `id` (course ID) | `{ "message": "Course deleted successfully" }` |
| **/api/courses/enroll** | POST | Enroll a user in a course. | `user_id`, `course_id` | `{ "message": "Enrollment successful", "enrollment": { ... } }` |
| **/api/users/{id}/profile** | GET | Retrieves a user's profile information. | `id` (user ID) | `{ "user": { "id": 1, "name": "John Doe", "email": "..." } }` |
| **/api/users/{id}/update** | PUT | Updates a user's profile | `id`, `name`, `email`, `phone` | `{ "message": "Profile updated` |

| | | | | |
|---|---|---|---|---|
| | | information. | | `successfully", "user": { ... } }` |
| **/api/users/{id}/delete** | DELETE | Deletes a user account. | `id` (user ID) | `{ "message": "User deleted successfully" }` |
| **/api/categories** | GET | Retrieves all course categories. | None | `[ { "id": 1, "name": "Programming", "description": "..." }, ... ]` |
| **/api/categories/create** | POST | Create a new course category. | `name`, `description` | `{ "message": "Category created successfully", "category": { ... } }` |
| **/api/categories/{id}/update** | PUT | Update an existing course category. | `id`, `name`, `description` | `{ "message": "Category updated successfully", "category": { ... } }` |
| **/api/categories/{id}/delete** | DELETE | Deletes a course category by ID. | `id` (category ID) | `{ "message": "Category deleted successfully" }` |
| **/api/cart** | GET | Retrieves the items in the user's cart. | `user_id` | `{ "items": [ { "product": { ... }, "quantity": 1, "price": 100 } ], "total_price": 200 }` |
| **/api/cart/add** | POST | Adds an item to the user's cart. | `user_id`, `product_id`, `quantity` | `{ "message": "Item added to cart", "cart": { ... } }` |
| **/api/cart/remove** | DELETE | Removes an item from the user's cart. | `user_id`, `product_id` | `{ "message": "Item removed from cart" }` |
| **/api/payment/checkout** | POST | Initiates payment for the user's cart. | `user_id`, `payment_method`, `total_price` | `{ "message": "Payment successful", "order_details": { ... } }` |

## Explanation:

- **POST** requests are used for creating or submitting data (e.g., user registration, creating a course).

- **GET** requests are used for retrieving data (e.g., retrieving user profile, course list).
- **PUT** requests are used to update existing data (e.g., updating course details, user profile).
- **DELETE** requests are used for removing data (e.g., deleting a course or user).
- Each endpoint has a specific function (creating, updating, retrieving, or deleting resources like users, courses, and categories) and requires specific parameters that the backend will process to respond appropriately.
- This table outlines the general structure of the API for your platform.

---

## General API Notes with Security Considerations

1. **User Registration (`/api/users/register`)**:
   - **Functionality**: Allows users to register by providing their details (name, email, password, phone).
   - **Security**:
     - **Password Hashing**: The user password should never be stored as plaintext in the database. Use a secure hashing algorithm like **bcrypt** or **argon2** to hash the password before saving it.
     - **Input Validation**: Sanitize and validate inputs to prevent SQL injection and XSS attacks.
2. **User Login (`/api/users/login`)**:
   - **Functionality**: Allows users to log in by providing their email and password.
   - **Security**:
     - **Password Hashing**: When comparing the login password to the stored hashed password in the database, use `password_verify()` in PHP for bcrypt or similar functions.
     - **JWT Authentication**: Upon successful login, generate a **JWT** token. This token should be returned in the response and used for authenticating future requests. Ensure the token is stored securely on the client-side (e.g., in HTTP-only cookies or local storage).

- **Token Expiry**: Set an expiry time for the JWT token to limit its lifespan and ensure regular re-authentication.
- **Input Validation**: Validate email format and sanitize input to prevent injection attacks.

3. **Courses CRUD Operations (`/api/courses`)**:
   - **Functionality**:
     - **GET** `/api/courses`: Retrieves a list of available courses.
     - **GET** `/api/courses/{id}`: Retrieves a single course by its ID.
     - **POST** `/api/courses/create`: Allows a teacher to create a new course.
     - **PUT** `/api/courses/{id}/update`: Allows updating a course's details.
   - **Security**:
     - **JWT Authentication**: Every request to CRUD operations should include a valid JWT token in the **Authorization header**. This ensures only authenticated users can access and modify course data.
     - **Role-Based Access Control**: Only users with specific roles (e.g., **teacher**, **admin**) should be able to create or update courses. Admins should have full control, while teachers may only edit their own courses.
     - **Input Validation**: All inputs should be sanitized to prevent SQL injection and ensure only valid data is accepted.
     - **Authorization Check**: For sensitive operations like updating or deleting courses, verify the user's identity and role before allowing them to proceed.
4. **General Security Measures**:
   - **Input Sanitization**: Sanitize all user inputs to prevent malicious input (e.g., SQL injection, XSS attacks). Use PHP functions like `htmlspecialchars()` and `filter_var()` for sanitization.
   - **JWT Token Expiry and Revocation**: Implement a refresh token mechanism to issue new JWT tokens and revoke old ones after expiry or logout.
5. **Password Security**:
   - Use **bcrypt** for hashing passwords in PHP. Example:

```
php
CopyEdit
// Hash password during registration
```

```
$hashed_password = password_hash($password, PASSWORD_BCRYPT);

// Verify password during login
if (password_verify($password, $stored_hashed_password)) {
    // Generate JWT token
}
```

- **Do not** store plaintext passwords in the database.
- Enforce **strong password policies** (e.g., minimum length, special characters, numbers) to enhance user security.

6. **JWT Token Management**:
   - On successful login, generate a JWT token using a library such as `firebase/php-jwt`.
   - Include the JWT token in the **Authorization header** of each request from the client to authenticate the user:

```
http
CopyEdit
Authorization: Bearer <JWT_TOKEN>
```

   - The JWT token should contain the user ID and role, which can be used for authorization checks on each request.

## Example Flow:

1. **User registers**:
   - The user provides their details, and the password is hashed using bcrypt before storing in the database.
2. **User logs in**:
   - The login request compares the entered password with the hashed password in the database.
   - On successful login, a **JWT token** is generated and returned to the client.
3. **Authenticated requests**:

- The client sends the JWT token in the **Authorization header** for protected routes (e.g., creating or updating courses).
- The server verifies the token on each request, checking if it’s valid and if the user has the appropriate role to access the resource.

4. **Role-based access**:
    - Admins and teachers have different permissions (e.g., admins can manage all courses, teachers can only manage their own).