To implement the quiz feature properly, we will need the following new entities:

New Entities:

1. **Question**
    - Each quiz consists of multiple questions.
2. **Option**
    - Each question has multiple answer choices.
3. **QuizAttempt**
    - Tracks which student attempted which quiz.
4. **Answer**
    - Stores student's answers for each question.

Updated Database Schema:

We'll update the existing `Quiz` entity and add new tables.

Quiz Table (Updated)

```
@Entity()
export class Quiz {
  @PrimaryGeneratedColumn()
  quizId: number;

  @Column()
  quizName: string;

  @Column()
  courseId: number;

  @Column()
  description: string;

  @Column()
  totalMarks: number;

  @Column()
  createdBy: number; // Tutor who created the quiz
```

```
  @ManyToOne(() => Course, (course) => course.quizzes)
  @JoinColumn({ name: "courseId" })
  course: Course;
}
```

Question Table

```
@Entity()
export class Question {
  @PrimaryGeneratedColumn()
  questionId: number;

  @Column()
  quizId: number;

  @Column()
  questionText: string;

  @Column()
  correctOptionId: number;

  @ManyToOne(() => Quiz, (quiz) => quiz.questions)
  @JoinColumn({ name: "quizId" })
  quiz: Quiz;
}
```

Option Table

```
@Entity()
export class Option {
  @PrimaryGeneratedColumn()
  optionId: number;

  @Column()
  questionId: number;

  @Column()
  optionText: string;

  @ManyToOne(() => Question, (question) => question.options)
  @JoinColumn({ name: "questionId" })
  question: Question;
}
```

QuizAttempt Table

```
@Entity()
```

```
export class QuizAttempt {
  @PrimaryGeneratedColumn()
  attemptId: number;

  @Column()
  userId: number;

  @Column()
  quizId: number;

  @Column({ type: "float" })
  score: number;

  @Column()
  attemptDate: Date;

  @ManyToOne(() => User, (user) => user.quizAttempts)
  @JoinColumn({ name: "userId" })
  user: User;

  @ManyToOne(() => Quiz, (quiz) => quiz.attempts)
  @JoinColumn({ name: "quizId" })
  quiz: Quiz;
}
```

Answer Table

```
@Entity()
export class Answer {
  @PrimaryGeneratedColumn()
  answerId: number;

  @Column()
  attemptId: number;

  @Column()
  questionId: number;

  @Column()
  selectedOptionId: number;

  @ManyToOne(() => QuizAttempt, (attempt) => attempt.answers)
  @JoinColumn({ name: "attemptId" })
  attempt: QuizAttempt;

  @ManyToOne(() => Question, (question) => question.answers)
  @JoinColumn({ name: "questionId" })
  question: Question;
```

}

Step-by-Step Implementation:

1. Backend - Quiz Creation

- Tutors can create quizzes with multiple questions and options.
- Implement an API to handle quiz creation.

Controller:

```
@Post('/create')
async createQuiz(@Body() quizDto: CreateQuizDto): Promise<Quiz> {
  return this.quizService.createQuiz(quizDto);
}
```

Service:

```
async createQuiz(quizDto: CreateQuizDto): Promise<Quiz> {
  const quiz = this.quizRepository.create(quizDto);
  return this.quizRepository.save(quiz);
}
```

---

2. Backend - Taking the Quiz

- Students fetch quiz questions and submit answers.

Fetching Quiz Questions:

```
@Get('/:quizId/questions')
async getQuizQuestions(@Param('quizId') quizId: number): Promise<Question[]> {
  return this.quizService.getQuizQuestions(quizId);
}
```

Submitting Answers:

```
@Post('/submit')
async submitAnswers(@Body() attemptDto: QuizAttemptDto): Promise<QuizAttempt> {
  return this.quizService.submitAttempt(attemptDto);
}
```

---

3. Backend - Results Calculation

- Once a student submits a quiz, we calculate their score.

Service Logic:

```
async submitAttempt(attemptDto: QuizAttemptDto): Promise<QuizAttempt> {
  let score = 0;

  for (const answer of attemptDto.answers) {
    const correctAnswer = await this.questionRepository.findOne(answer.questionId);
    if (correctAnswer.correctOptionId === answer.selectedOptionId) {
      score += 1; // Each correct answer gives 1 mark
    }
  }

  const attempt = this.quizAttemptRepository.create({
    userId: attemptDto.userId,
    quizId: attemptDto.quizId,
    score,
    attemptDate: new Date(),
  });

  return this.quizAttemptRepository.save(attempt);
}
```

---

4. Frontend Implementation using PrimeNG

1. **Tutor Dashboard**
    - Create quizzes
    - View students' scores
2. **Student Dashboard**
    - Take quizzes
    - View results

Using PrimeNG

You need to install PrimeNG in Angular:

```
npm install primeng --save
npm install primeicons --save
```

Import PrimeNG modules in `app.module.ts`:

```
import { TableModule } from 'primeng/table';
import { ButtonModule } from 'primeng/button';
import { DialogModule } from 'primeng/dialog';
```

Example Quiz Form (Tutor creates quiz):

```
<p-dialog [(visible)]="display" header="Create Quiz">
  <div>
    <label>Quiz Name:</label>
    <input type="text" [(ngModel)]="quiz.quizName" />
  </div>

  <div>
    <label>Description:</label>
    <textarea [(ngModel)]="quiz.description"></textarea>
  </div>

  <button pButton type="button" label="Save" (click)="saveQuiz()"></button>
</p-dialog>
```

Payment System Enhancement

- **Subscription-based Model:** Students must pay to access quizzes and courses.
- **Integration:** Use Stripe/Razorpay API.

Backend API for Payment:

```
@Post('/payment')
async makePayment(@Body() paymentDto: PaymentDto): Promise<Payment> {
  return this.paymentService.processPayment(paymentDto);
}
```

Payment Processing Logic:

```
async processPayment(paymentDto: PaymentDto): Promise<Payment> {
```

```
  const payment = this.paymentRepository.create(paymentDto);
  return this.paymentRepository.save(payment);
}
```

Final Steps

- Create API routes for all quiz functionalities.
- Integrate PrimeNG tables and buttons for user interaction.
- Secure APIs with authentication.

Let me know if you need frontend integration with PrimeNG step-by-step! 🚀