Questions:

1. User & Profile (One-to-One)

📝 Scenario:

Create a feature where each user has one profile containing bio and avatar URL. Save both when creating a user.

🧩 Tasks:

- Define User and UserProfile entities with a one-to-one relation.
- Create a DTO to accept user + profile data.
- Implement a service method to save the data.
- Add a controller POST route to handle it.

✅ 2. Blog Post & Comments (One-to-Many)

📝 Scenario:

Create a blog feature where one post can have many comments.

🧩 Tasks:

- Entity for Post and Comment with a one-to-many relationship.
- DTO for creating a post with initial comments.
- Service to save post and cascade comments.
- Controller POST /post to create it.

✅ 3. Product & Category (Many-to-Many)

📝 **Scenario:**

A product can belong to many categories, and a category can contain many products.

🧩 **Tasks:**

- Define many-to-many relationship with @JoinTable() on one side.
- Create DTO with category IDs in product creation.
- Handle saving product with categories in the service.
- Create endpoint to list all products by category.

✅ 4. Search by Name Using ILike

📝 **Scenario:**

Build an endpoint to search organizers by full name (case-insensitive).

🧩 **Tasks:**

- Write service method using ILike.
- Add a GET endpoint like /organizer/search?name=polash
- Return matching results.

✅ 5. UUID Generation with @BeforeInsert()

📝 **Scenario:**

Each new customer should get a unique UUID on creation.

🧩 **Tasks:**

- Use uuid npm package.
- Add @BeforeInsert() in the Customer entity to generate it.
- Test with POST /customer.

✅ 6. Soft Delete (isActive flag)

📝 **Scenario:**

Instead of deleting users, mark them as inactive with an isActive: boolean.

🧩 **Tasks:**

- Add isActive column to User entity.
- Create a DELETE route that sets isActive = false.
- Modify findAll() to only return active users.

✅ 7. Admin Auth - Login with DTO and Controller

📝 **Scenario:**

Admins log in with username and password. On success, return a success message.

🧩 **Tasks:**

- Create AdminLoginDto with username and password.

- Write controller POST /admin/login.
- Write service to match credentials.
- Throw UnauthorizedException if wrong.

✅ 8. Upload with File Path (No actual file handling)

📝 Scenario:

When creating a document, save its name and file path (just as strings).

🧩 Tasks:

- Create Document entity with name and path columns.
- Write DTO and POST controller.
- Save record with dummy file path (/uploads/doc.pdf).

Solutions:

// ✅ Scenario 1: User & Profile (One-to-One)

// ---

// user.entity.ts

```
@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @OneToOne(() => UserProfile, profile => profile.user, { cascade:
true })
  @JoinColumn()
  profile: UserProfile;
}

// user-profile.entity.ts
@Entity()
export class UserProfile {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  bio: string;

  @Column()
  avatarUrl: string;

  @OneToOne(() => User, user => user.profile)
  user: User;
}

// user.dto.ts
export class CreateUserDto {
  name: string;
  profile: {
    bio: string;
    avatarUrl: string;
  };
}

// user.service.ts
async create(dto: CreateUserDto): Promise<User> {
  const user = this.userRepo.create(dto);
  return this.userRepo.save(user);
}

// user.controller.ts
@Post()
createUser(@Body() dto: CreateUserDto) {
  return this.userService.create(dto);
}



// ✅ Scenario 2: Blog Post & Comments (One-to-Many)
// -----------------------------------------------

// post.entity.ts
@Entity()
export class Post {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  title: string;
  @OneToMany(() => Comment, comment => comment.post, {
cascade: true })
  comments: Comment[];
}

// comment.entity.ts
@Entity()
export class Comment {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  text: string;

  @ManyToOne(() => Post, post => post.comments)
  post: Post;
}

// create-post.dto.ts
export class CreatePostDto {
  title: string;

  comments: { text: string }[];

}


// post.service.ts

async create(dto: CreatePostDto): Promise<Post> {

  const post = this.postRepo.create(dto);

  return this.postRepo.save(post);

}


// post.controller.ts

@Post()

createPost(@Body() dto: CreatePostDto) {

  return this.postService.create(dto);

}



// ✅ Scenario 3: Product & Category (Many-to-Many)

// ------------------------------------------------


// product.entity.ts
```

```
@Entity()
export class Product {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @ManyToMany(() => Category, category => category.products, {
cascade: true })
  @JoinTable()
  categories: Category[];
}

// category.entity.ts
@Entity()
export class Category {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @ManyToMany(() => Product, product => product.categories)
  products: Product[];
}

// create-product.dto.ts
export class CreateProductDto {
  name: string;
  categoryIds: number[];
}

// product.service.ts
async create(dto: CreateProductDto): Promise<Product> {
  const categories = await
this.categoryRepo.findByIds(dto.categoryIds);
  const product = this.productRepo.create({ ...dto, categories });
  return this.productRepo.save(product);
}

// product.controller.ts
```

```
@Post()
createProduct(@Body() dto: CreateProductDto) {
  return this.productService.create(dto);
}



// ✅ Scenario 4: Search by Name Using ILike
// ----------------------------------------

// organizer.service.ts
async searchByName(name: string): Promise<Organizer[]> {
  return this.organizerRepo.find({ where: { fullName: 
ILike(`%${name}%`) } });
}

// organizer.controller.ts
@Get('search')
searchByName(@Query('name') name: string) {
  return this.organizerService.searchByName(name);
}
```

```
// ✅ Scenario 5: UUID Generation with @BeforeInsert()
// ---------------------------------------------------

// customer.entity.ts
@Entity()
export class Customer {
  @PrimaryColumn()
  uuid: string;

  @Column()
  name: string;

  @BeforeInsert()
  generateUUID() {
    this.uuid = uuidv4();
  }
}


// ✅ Scenario 6: Soft Delete with isActive Flag
// --------------------------------------------

// user.entity.ts
@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @Column({ default: true })
  isActive: boolean;
}

// user.service.ts
async softDelete(id: number): Promise<void> {
  const user = await this.userRepo.findOneBy({ id });
  if (!user) throw new NotFoundException('User not found');
  user.isActive = false;
  await this.userRepo.save(user);
```

```
}

// user.controller.ts
@Delete(':id')
softDelete(@Param('id', ParseIntPipe) id: number) {
  return this.userService.softDelete(id);
}



// ✅ Scenario 7: Admin Login with DTO
// ----------------------------------

// admin-login.dto.ts
export class AdminLoginDto {
  username: string;
  password: string;
}

// admin.service.ts
async login(dto: AdminLoginDto): Promise<string> {
  const admin = await this.adminRepo.findOneBy({ username:
dto.username });
  if (!admin || admin.password !== dto.password) {
    throw new UnauthorizedException('Invalid credentials');
  }
  return 'Login successful';
}

// admin.controller.ts
@Post('login')
login(@Body() dto: AdminLoginDto) {
  return this.adminService.login(dto);
}



// ✅ Scenario 8: File Upload (Storing Path Only)
// --------------------------------------------

// document.entity.ts
@Entity()
export class Document {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  name: string;

  @Column()
  filePath: string;
}

// create-document.dto.ts
export class CreateDocumentDto {
  name: string;
  filePath: string;
}

// document.service.ts
async create(dto: CreateDocumentDto): Promise<Document> {
  const doc = this.documentRepo.create(dto);
  return this.documentRepo.save(doc);
}
```

```
// document.controller.ts
@Post()
create(@Body() dto: CreateDocumentDto) {
  return this.documentService.create(dto);
}
```

//

"Create a simple Quiz Management feature where an Admin can create quizzes. Each quiz has a title and a list of questions. Each question has text and four options, with one correct answer."

🧩 **Task:**

Implement the following:

1. **Entity files** for Quiz and Question with a **One-to-Many** relationship.
2. A **DTO** for creating a quiz with questions.
3. A **Service** method to save a quiz with its questions.
4. A **Controller** endpoint to handle the POST request.

Ans:

```
// quiz.entity.ts
@Entity()
export class Quiz {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  title: string;

  @OneToMany(() => Question, question => question.quiz, { cascade:
true })
  questions: Question[];
}

// question.entity.ts
@Entity()
export class Question {
  @PrimaryGeneratedColumn()
  id: number;
  @Column()
  text: string;

  @Column("simple-array")
  options: string[];

  @Column()
  correctAnswer: string;

  @ManyToOne(() => Quiz, quiz => quiz.questions)
  quiz: Quiz;
}


// create-quiz.dto.ts
export class CreateQuizDto {
  title: string;
  questions: {
    text: string;
    options: string[];
    correctAnswer: string;
  }[];
}

// quiz.service.ts
async create(dto: CreateQuizDto): Promise<Quiz> {
  const quiz = this.quizRepo.create(dto);
  return this.quizRepo.save(quiz);
}

// quiz.controller.ts
@Post()
createQuiz(@Body() dto: CreateQuizDto) {
  return this.quizService.create(dto);
}
```