PART 4: WRITTEN QUESTIONS (ARCHITECTURE & DECISIONS)

Q1: Scalability & Performance:

For optimizing /products/search and /products/trending endpoints for scalability and performance with 100K+ products:

1. Database Indexing

- **Search:** Add indexes on columns frequently searched.
- **Trending:** Index fields used for sorting/filtering (createdAt).

2. Full-Text Search.

- Integrating Elasticsearch for fast and flexible search.

3. Pagination and Limits

- Always paginate results (limit and offset).
- Never return all products at once.

4. Caching

- Cache popular search queries and their results.

Q2: Security

- **How do you protect token refresh endpoints?**
- **What measures do you take for secure file uploads?**

1. Protecting Token Refresh Endpoints

- Store refresh tokens in HTTP only cookies to prevent JavaScript access.
- Keep refresh tokens short lived and rotate them on every use.
- Maintain a blacklist for revoked tokens (after logout or password change).
- Apply rate limiting to prevent brute force attacks.
- Always use HTTPS to transmit tokens.

2. Secure File Uploads

- Only allow specific types and file extensions.
- Set maximum file size limits.
- Restrict who can upload and access files (authentication/authorization).
- Apply rate limiting to upload endpoints.

Q3: Internationalization

- How would you implement multi-language support for product fields (name, description)?

1. Create a Product Translation Entity

Each translation is a row linked to a product and a language code.

```
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from 'typeorm';

import { Product } from './product.entity';

@Entity()

export class ProductTranslation {

  @PrimaryGeneratedColumn()

  id: number;

  @Column()

  language: string; // 'en', 'fr'

  @Column()

  name: string;

  @Column('text')

  description: string;

  @ManyToOne(() => Product, product => product.translations, { onDelete: 'CASCADE' })

  product: Product;

}
```

2. API

- Accept translations as an array in your DTOs.
- When fetching a product, return the translation matching the requested language

Q4: Analytics

- How would you track product views, analyze trends, and compute ratings efficiently?

1. Tracking Product Views

- Create a ProductView entity/table with fields: productId, userId, timestamp.
- For high traffic use an in-memory store to increment view counts in real time.

2. Analyzing Trends

- Define a trending using recent views, sales, ratings.
- Store trending product IDs

3. Computing Ratings

- Store each user's rating in a Review entity
  with productId, userId, rating, comment, createdAt.
- Use SQL aggregate functions (AVG, COUNT) if you need to recalculate on demand

Q5: Recommended Stack – Justify Your Toolset

- **What tools, frameworks, and services would you use to build this project from scratch, and why?**

  Please specify and justify your choices for:

Area	Your Stack	Why

Backend Framework	NestJS, Express.js, etc.(exmpl)	Modular, scalable, built in DI, great TypeORM support.
Frontend Framework	React, Vue, etc.(exmpl)	Component-based, strong community, easy integration.
Database	PostgreSQL, MongoDB, etc.(exmpl)	Strong relational features, JSON support, scalable.
File Storage	Cloudinary, Firebase, Local, etc.(exmpl)	Secure, scalable, fast CDN delivery
Auth & Security	JWT, 2FA, etc.(exmpl)	JWT for stateless auth, refresh tokens for security
DevOps	GitHub Actions, Docker, etc.(exmpl)	Docker for consistency, GitHub Actions for CI/CD,

==➔Justification Details

- **NestJS:** Clean architecture, TypeScript, easy testing, scalable for microservices.
- **React:** Fast UI development with Next.js, huge ecosystem.
- **PostgreSQL:** Handles complex queries, supports indexing and is widely used in production.
- **Cloudinary/S3:** Offloads file storage, handles image/video transformations, secure uploads.
- **JWT & 2FA:** JWT is standard for APIs, refresh tokens for session security, 2FA for user protection.
- **Docker & GitHub Actions:** Ensures reproducible builds, automated testing/deployment.