**TypeScript**

* **Static Typing**
* **تسمح لك بتحديد نوع البيانات Data Type لل variables , array , function وده بيساعد في تقليل الاخطاء وقت الdev .**
* **Object-Oriented**
* **بتوفر دعم قوي لمافهيم ال OOP زي ال Interface , Enums , Inheritance**
* **Custom** **Type**
* من الممكن انشاء ال type الخاص بك عن طريق Interface , type , enum , class
* **Generic**
* ميزة قوية تُستخدم لجعل الكود أكثر **مرونة** و **قابلية لإعادة الاستخدام حيث يتم تعريف الأنواع بشكل ديناميكي، بدلاً من الالتزام بنوع واحد ثابت**
* **Utility Types (**Omit, Pick , Partial, Required**)** 
  + هي أدوات قوية تساعد في تعديل الأنواع بسهولة وفعالية و تُستخدم كثيرًا لجعل الكود أكثر وضوحًا وقابلية لإعادة الاستخدام.
    - Omit : يتستخدم لإنشاء نوع جديد يحتوي على جميع الحقول من النوع الأصلي **باستثناء الحقول المحددة**.
    - Pick : يستخدم لإنشاء نوع جديد يحتوي على **مجموعة محددة من الحقول** من النوع الأصلي.
    - Partial : يستخدم لجعل **جميع الحقول اختيارية (Optional)**.
    - Required : تستخدم لجعل **جميع الحقول إلزامية (Required)**.

**Nest JS**

* **NestJS  هو إطار عمل (Framework) لبناء تطبيقات Node.js باستخدام TypeScript. يعتمد على البرمجة الكائنية (OOP) والبرمجة الموجهة للأحداث (Event-driven) والبرمجة الوظيفية (Functional Programming)، ويستخدم معمارية موديولات (Modules) مشابهة لـ Angular، مما يجعله أداة قوية لإنشاء تطبيقات خوادم متقدمة وسهلة التنظيم والصيانة.**
* **مميزات ال NestJs** 
  + **يعتمد على TypeScript بالكامل و يوفر كتابة كود آمنة وقابلة للصيانة و يساعد في الكشف عن الأخطاء أثناء الكتابة .**
  + **بنية معمارية منظمة يعمل على مبدأ الوحدات (Modules)، مما يساعد في تقسيم المشروع إلى أجزاء صغيرة يسهل التحكم فيها.**

**يعتمد على مفاهيم مثل Controllers و Services لتنسيق الكود.**

|  |  |
| --- | --- |
| Controllers | المسؤول عن التعامل مع طلبات المستخدمين (HTTP Requests) |
| Providers | **تستخدم لإدارة منطق العمل (Business Logic).** |
| Modules | **حجر الأساس في أي تطبيق NestJS , الوسيط الذي يربط بين ال Controllers , Providers (Services)** |
| Middleware | **عبارة عن function يتم استدعاؤها قبل معالج route و ايضا لديها القدره علي الوصول الي request and response** |
| Exception filters | **تستخدم في معالجة الأخطاء والاستثناءات (Exceptions) التي تحدث أثناء معالجة request** |
| Pipes | **تستخدم في معالجة البيانات أو التحقق من صحتها (Validation) أو تحويلها (Transformation) بعد وصول request إلى controllers** |
| Guards | **تستخدم في معالجة الأخطاء والاستثناءات (Exceptions) التي تحدث أثناء معالجة request** |
| Interceptors | **التعامل مع الطلب أو الاستجابة قبل أو بعد تنفيذ method في controllers و لها استخدمات في ال Caching , Performance , logging** |

**Hint :**

**Explanation of the Execution Order:**

1. **Middleware: Executes first, handling raw requests (e.g., logging, modifying headers).**
2. **Guard: Executes next, determining whether the request is allowed to proceed.**
3. **Pipe: Processes incoming data, ensuring validation or transformation before the Controller method executes.**
4. **Interceptor: Runs before and after the Controller method to modify requests, responses, or measure performance.**
5. **Filter: Only executes if an exception is thrown, handling and customizing error responses.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature | Middleware | Guard | Pipe | Interceptor | Filter |
| Purpose | Process requests before reaching controllers. | Control access by allowing or denying requests. | Validate or transform request data. | Modify requests/responses or log performance. | Handle exceptions and errors. |
| Execution Stage | Before anything else in the request lifecycle. | After Middleware, before Pipes. | After Guards, before Controller methods. | Before and after Controller methods. | After execution or during exceptions. |
| Use Case | Logging, request parsing, authentication. | Authentication, authorization. | Validation (e.g., DTO validation), type conversion | Response mapping, caching, performance monitoring. | Centralized error handling. |
| Request Access | Yes | Yes | Yes | Yes | On exceptions only. |
| Response Access | Yes | No | No | Yes | Yes |
| Applied To | Application, Module, Route | Application, Controller, Route | Application, Controller, Method | Application, Controller, Method | Application, Controller, Method |
| Key Example | Modify headers or log incoming requests. | Ensure only authorized users access a route. | Validate incoming query or body data. | Modify outgoing response or log execution time. | Send custom error responses to the client. |

**Database In Nest JS**

**أولًا: TypeORM مع قواعد بيانات SQL**

* **TypeORM** هو ORM (Object-Relational Mapping) يُستخدم للتعامل مع قواعد بيانات SQL مثل PostgreSQL, MySQL, SQLite, وغيرها.

الخطوات الأساسية :

* **تثبيت المكتبه الخاصه بال ORM و الخاصه بنوع قاعده البيانات المستخدمه مثل typeorm و pg**
* **أضافه ال configuration الخاص بالاتصال بقاعدة البيانات**
* **إنشاء الكيانات (Entities) الخاصه و ربطها مع ال module الخاص بها**
* **اضافه ال Operation الخاصه بالتعامل مع قواعد البيانات CRUD و اضافه الController , Service الخاصه بها**

**ثانيًا: Mongoose مع قواعد بيانات MongoDB**

* **Mongoose  هو ODM (Object-Document Mapping) يُستخدم للتعامل مع MongoDB.**

**الخطوات الأساسية :**

* **تثبيت المكتبه الخاصه بال ODM و الخاصه بنوع قاعده البيانات المستخدمه مثل mongoose**
* **أضافه ال configuration الخاص بالاتصال بقاعدة البيانات**
* **إنشاء الـ Schema**
  + **الـ Schema تمثل البيانات في MongoDB**
* **ربط الـ Schema بال module الخاص بها**
* **اضافه ال Operation الخاصه بالتعامل مع قواعد البيانات CRUD و اضافه الController , Service الخاصه بها**

|  |  |  |
| --- | --- | --- |
| الميزة | TypeORM (SQL) | Mongoose (MongoDB) |
| نوع قاعدة البيانات | Relational (PostgreSQL, MySQL, etc.) | NoSQL (MongoDB) |
| تصميم البيانات | الجداول والعلاقات | الوثائق (Documents) |
| لغة الاستعلام | SQL / QueryBuilder | MongoDB Queries |
| المرونة في البيانات | أقل مرونة (Schema ثابت) | مرونة عالية (Dynamic Schema) |
| الأداء | ممتاز في العلاقات المعقدة | ممتاز مع البيانات الكبيرة وغير المرتبة |

**الخلاصة**

* **TypeORM مناسب أكثر لـ SQL Databases حيث تحتاج إلى بناء علاقات معقدة وجداول مترابطة.**
* **Mongoose مناسب لـ MongoDB (NoSQL) حيث البيانات مرنة ولا تتطلب تصميمًا صارمًا.**

**Use JWT in Authentication in Nest JS**

* باختصار يتم استخدام ال JWT (JSON Web Token) في ال **NestJs** لتنفيذ عملية المصادقة (**Authentication**) بشكل آمن وفعال.
* 1. خطوات العمل:
  + إنشاء التوكن عند تسجيل الدخول:
    - يتم إنشاء توكن JWT يحتوي على بيانات المستخدم (مثل id وusername) كمحتوى.
    - يتم توقيع التوكن بمفتاح سري.
  + إرسال التوكن إلى العميل:
    - يتم إرجاع التوكن للعميل كـ Bearer Token.
  + التحقق من صحة التوكن:
    - في كل طلب محمي، يرسل العميل التوكن.
    - يتم فك التوكن والتحقق من صحة التوقيع والبيانات.