Extensions

> warning **Warning** This chapter applies only to the code first approach.

Extensions is an **advanced, low-level feature** that lets you define arbitrary data in the types configuration. Attaching custom metadata to certain fields allows you to create more sophisticated, generic solutions. For example, with extensions, you can define field-level roles required to access particular fields. Such roles can be reflected at runtime to determine whether the caller has sufficient permissions to retrieve a specific field.

Adding custom metadata

To attach custom metadata for a field, use the `@Extensions()` decorator exported from the `@nestjs/graphql` package.

```
@Field()
@Extensions({ role: Role.ADMIN })
password: string;
```

In the example above, we assigned the `role` metadata property the value of `Role.ADMIN`. `Role` is a simple TypeScript enum that groups all the user roles available in our system.

Note, in addition to setting metadata on fields, you can use the `@Extensions()` decorator at the class level and method level (e.g., on the query handler).

Using custom metadata

Logic that leverages the custom metadata can be as complex as needed. For example, you can create a simple interceptor that stores/logs events per method invocation, or a field middleware that matches roles required to retrieve a field with the caller permissions (field-level permissions system).

For illustration purposes, let's define a `checkRoleMiddleware` that compares a user's role (hardcoded here) with a role required to access a target field:

```
export const checkRoleMiddleware: FieldMiddleware = async (
  ctx: MiddlewareContext,
  next: NextFn,
) => {
  const { info } = ctx;
  const { extensions } = info.parentType.getFields()[info.fieldName];

  /**
   * In a real-world application, the "userRole" variable
   * should represent the caller's (user) role (for example,
"ctx.user.role").
   */
  const userRole = Role.USER;
  if (userRole === extensions.role) {
    // or just "return null" to ignore
    throw new ForbiddenException(
      `User does not have sufficient permissions to access
"${info.fieldName}" field.`,
    );
  }
  return next();
};
```

With this in place, we can register a middleware for the `password` field, as follows:

```
@Field({ middleware: [checkRoleMiddleware] })
@Extensions({ role: Role.ADMIN })
password: string;
```