Router module

> info **Hint** This chapter is only relevant to HTTP-based applications.

In an HTTP application (for example, REST API), the route path for a handler is determined by concatenating the (optional) prefix declared for the controller (inside the `@Controller` decorator), and any path specified in the method's decorator (e.g, `@Get('users')`). You can learn more about that in this section. Additionally, you can define a global prefix for all routes registered in your application, or enable versioning.

Also, there are edge-cases when defining a prefix at a module-level (and so for all controllers registered inside that module) may come in handy. For example, imagine a REST application that exposes several different endpoints being used by a specific portion of your application called "Dashboard". In such a case, instead of repeating the `/dashboard` prefix within each controller, you could use a utility `RouterModule` module, as follows:

```
@Module({
  imports: [
    DashboardModule,
    RouterModule.register([
      {
        path: 'dashboard',
        module: DashboardModule,
      },
    ]),
  ],
})
export class AppModule {}
```

> info **Hint** The `RouterModule` class is exported from the `@nestjs/core` package.

In addition, you can define hierarchical structures. This means each module can have `children` modules. The children modules will inherit their parent's prefix. In the following example, we'll register the `AdminModule` as a parent module of `DashboardModule` and `MetricsModule`.

```
@Module({
  imports: [
    AdminModule,
    DashboardModule,
    MetricsModule,
    RouterModule.register([
      {
        path: 'admin',
        module: AdminModule,
        children: [
          {
            path: 'dashboard',
            module: DashboardModule,
          },
```

```
            {
              path: 'metrics',
              module: MetricsModule,
            },
          ],
        },
      ])
    ],
  });
```

> info **Hint** This feature should be used very carefully, as overusing it can make code difficult to maintain over time.

In the example above, any controller registered inside the `DashboardModule` will have an extra `/admin/dashboard` prefix (as the module concatenates paths from top to bottom - recursively - parent to children). Likewise, each controller defined inside the `MetricsModule` will have an additional module-level prefix `/admin/metrics`.