Update `module` and `target` compiler options migration

What does this migration do?

This migration adjusts the `target` and `module` settings within the TypeScript configuration files for the workspace. The changes to each option vary based on the builder or command that uses the TypeScript configuration file. Unless otherwise noted, changes are only made if the existing value was not changed since the project was created. This process helps ensure that intentional changes to the options are kept in place.

TypeScript Configuration File(s)	Changed Property	Existing Value	New Value
`<workspace base>/tsconfig.json`	`"module"`	`"esnext"`	`"es2020"`
Used in browser builder options (`ng build` for applications)	`"module"`	`"esnext"`	`"es2020"`
Used in `ng-packagr` builder options (`ng build` for libraries)	`"module"`	`"esnext"`	`"es2020"`
Used in `karma` builder options (`ng test` for applications)	`"module"`	`"esnext"`	`"es2020"`
Used in `server` builder options (universal)	`"module"`	`"commonjs"`	*removed*
Used in `server` builder options (universal)	`"target"`	*any*	`"es2016"`
Used in `protractor` builder options (`ng e2e` for applications)	`"target"`	`"es5"`	`"es2018"`

Why is this migration necessary?

This migration provides improvements to the long-term supportability of projects by updating the projects to use recommended best practice compilation options.

For the functionality that executes on Node.js, such as Universal and Protractor, the new settings provide performance and troubleshooting benefits as well. The minimum Node.js version for the Angular CLI (v10.13) supports features in ES2018 and earlier. By targeting later ES versions, the compiler transforms less code and can use newer features directly. Since zone.js does not support native async and `await`, the universal builds still target ES2016.

Why `"es2020"` instead of `"esnext"`?

In TypeScript 3.9, the behavior of the TypeScript compiler controlled by `module` is the same with both `"esnext"` and `"es2020"` values. This behavior can change in the future, because the `"esnext"` option could evolve in a backwards incompatible ways, resulting in build-time or run-time errors during a TypeScript update. As a result, code can become unstable. Using the `"es2020"` option mitigates this risk.