Contributing to Nest

We would love for you to contribute to Nest and help make it even better than it is today! As a contributor, here are the guidelines we would like you to follow:

- Question or Problem?
- Issues and Bugs
- Feature Requests
- Submission Guidelines
- Development Setup
- Coding Rules
- Commit Message Guidelines

Got a Question or Problem?

Do not open issues for general support questions as we want to keep GitHub issues for bug reports and feature requests. You've got much better chances of getting your question answered on Stack Overflow where the questions should be tagged with tag `nestjs`.

Stack Overflow is a much better place to ask questions since:

- questions and answers stay available for public viewing so your question / answer might help someone else
- Stack Overflow's voting system assures that the best answers are prominently visible.

To save your and our time, we will systematically close all issues that are requests for general support and redirect people to Stack Overflow.

If you would like to chat about the question in real-time, you can reach out via our discord channel.

Found a Bug?

If you find a bug in the source code, you can help us by submitting an issue to our GitHub Repository. Even better, you can submit a Pull Request with a fix.

Missing a Feature?

You can *request* a new feature by submitting an issue to our GitHub Repository. If you would like to *implement* a new feature, please submit an issue with a proposal for your work first, to be sure that we can use it. Please consider what kind of change it is:

- For a **Major Feature**, first open an issue and outline your proposal so that it can be discussed. This will also allow us to better coordinate our efforts, prevent duplication of work, and help you to craft the change so that it is successfully accepted into the project. For your issue name, please

prefix your proposal with `[discussion]`, for example "[discussion]: your feature idea".
- **Small Features** can be crafted and directly submitted as a Pull Request.

Submission Guidelines

Submitting an Issue

Before you submit an issue, please search the issue tracker, maybe an issue for your problem already exists and the discussion might inform you of workarounds readily available.

We want to fix all the issues as soon as possible, but before fixing a bug we need to reproduce and confirm it. In order to reproduce bugs we will systematically ask you to provide a minimal reproduction scenario using a repository or Gist. Having a live, reproducible scenario gives us wealth of important information without going back & forth to you with additional questions like:

- version of NestJS used
- 3rd-party libraries and their versions
- and most importantly - a use-case that fails

Unfortunately, we are not able to investigate / fix bugs without a minimal reproduction, so if we don't hear back from you we are going to close an issue that doesn't have enough info to be reproduced.

You can file new issues by filling out our new issue form.

Submitting a Pull Request (PR)

Before you submit your Pull Request (PR) consider the following guidelines:

1. Search GitHub for an open or closed PR that relates to your submission. You don't want to duplicate effort.
2. Fork the nestjs/nest repo.
3. Make your changes in a new git branch:

   ```
   git checkout -b my-fix-branch master
   ```

4. Create your patch, **including appropriate test cases**.
5. Follow our Coding Rules.
6. Run the full Nest test suite (see common scripts), and ensure that all tests pass.
7. Commit your changes using a descriptive commit message that follows our commit message conventions. Adherence to these conventions is necessary because release notes are automatically generated from these messages.

   ```
   git commit -a
   ```

Note: the optional commit `-a` command line option will automatically "add" and "rm" edited files.

8. Push your branch to GitHub:

   ```
   git push origin my-fix-branch
   ```

9. In GitHub, send a pull request to `nestjs:master`.

- If we suggest changes then:
  - Make the required updates.
  - Re-run the Nest test suites to ensure tests are still passing.
  - Rebase your branch and force push to your GitHub repository (this will update your Pull Request):

    ```
    git rebase master -i
    git push -f
    ```

That's it! Thank you for your contribution!

After your pull request is merged After your pull request is merged, you can safely delete your branch and pull the changes from the main (upstream) repository:

- Delete the remote branch on GitHub either through the GitHub web UI or your local shell as follows:

  ```
  git push origin --delete my-fix-branch
  ```

- Check out the master branch:

  ```
  git checkout master -f
  ```

- Delete the local branch:

  ```
  git branch -D my-fix-branch
  ```

- Update your master with the latest upstream version:

  ```
  git pull --ff upstream master
  ```

Development Setup

You will need Node.js version 8.9.0+.

1. After cloning the repo, run:

```
$ npm i # (or yarn install)
```

2. In order to prepare your environment run `prepare.sh` shell script:

```
$ sh scripts/prepare.sh
```

That will compile fresh packages and afterward, move them all to `sample` directories.

Commonly used NPM scripts

```
# build all packages and move to "sample" directories
$ npm run build

# run the full unit tests suite
$ npm run test

# run integration tests
# docker is required(!)
$ sh scripts/run-integration.sh

# run linter
$ npm run lint

# build all packages and put them near to their source .ts files
$ npm run build:prod
```

Coding Rules

To ensure consistency throughout the source code, keep these rules in mind as you are working:

- All features or bug fixes **must be tested** by one or more specs (unit-tests).
- We follow Google's JavaScript Style Guide, but wrap all code at **100 characters**. An automated formatter is available (`npm run format`).

Commit Message Guidelines

We have very precise rules over how our git commit messages can be formatted. This leads to **more readable messages** that are easy to follow when looking through the **project history**. But also, we use the git commit messages to **generate the Nest change log**.

Commit Message Format

Each commit message consists of a **header**, a **body** and a **footer**. The header has a special format that includes a **type**, a **scope** and a **subject**:

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

The **header** is mandatory and the **scope** of the header is optional.

Any line of the commit message cannot be longer than 100 characters! This allows the message to be easier to read on GitHub as well as in various git tools.

Footer should contain a closing reference to an issue if any.

Samples: (even more samples)

```
docs(changelog): update change log to beta.5
fix(core): need to depend on latest rxjs and zone.js
```

Revert

If the commit reverts a previous commit, it should begin with `revert:`, followed by the header of the reverted commit. In the body it should say: `This reverts commit <hash>.`, where the hash is the SHA of the commit being reverted.

Type

Must be one of the following:

- **build**: Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
- **chore**: Updating tasks etc; no production code change
- **ci**: Changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)
- **docs**: Documentation only changes
- **feat**: A new feature
- **fix**: A bug fix
- **perf**: A code change that improves performance
- **refactor**: A code change that neither fixes a bug nor adds a feature
- **style**: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)
- **test**: Adding missing tests or correcting existing tests
- **sample**: A change to the samples

Scope

The scope should be the name of the npm package affected (as perceived by person reading changelog generated from commit messages.

The following is the list of supported scopes:

- **common**
- **core**
- **sample**
- **microservices**
- **testing**
- **websockets**

There are currently a few exceptions to the "use package name" rule:

- **packaging**: used for changes that change the npm package layout in all of our packages, e.g. public path changes, package.json changes done to all packages, d.ts file/format changes, changes to bundles, etc.
- **changelog**: used for updating the release notes in CHANGELOG.md
- **sample/#**: for the example apps directory, replacing # with the example app number
- none/empty string: useful for `style`, `test` and `refactor` changes that are done across all packages (e.g. `style: add missing semicolons`)

Subject

The subject contains succinct description of the change:

- use the imperative, present tense: "change" not "changed" nor "changes"
- don't capitalize first letter
- no dot (.) at the end

Body

Just as in the **subject**, use the imperative, present tense: "change" not "changed" nor "changes". The body should include the motivation for the change and contrast this with previous behavior.

Footer

The footer should contain any information about **Breaking Changes** and is also the place to reference GitHub issues that this commit **Closes**.

Breaking Changes should start with the word `BREAKING CHANGE:` with a space or two newlines. The rest of the commit message is then used for this.

A detailed explanation can be found in this document.