Lesson | Wednesday

Intermediate JavaScript (/intermediate-javascript) / Asynchrony and APIs (/intermediate-javascript/asynchrony-and-apis) / Pull Requests and Submitting Great Work

Text

Coding in the Real World

Outside of Epicodus, developers don't generally work in pairs on a daily basis, although some companies use pair programming (or even mob programming) frequently. But they usually don't work entirely alone either. Instead, they work in small, specialized teams dedicated to one issue or project. But how does code get created in teams? While workflows differ, we'll discuss how software development generally works in a dev team.

Working with Senior Devs

Many teams follow the junior/senior dev hierarchy. (In some cases, teams may include mid-level developers too). Less experienced developers are **juniors** and seasoned developers with years of experience are **seniors**. In fact, if you're working on an open source project, you may be working with designated **maintainers** of the codebase that are essentially fulfilling the same job as a senior dev.

Working on Existing Code

Development teams almost always work on existing codebases, not new projects begun from scratch. These codebases may contain **legacy code**, which is code that may not be compatible with newer updates or may utilize outdated technologies or methods. It's not always possible to revamp a large project completely, especially in an agile environment.

This means you'll need to learn how to jump into existing code quickly while ensuring any changes you make don't break features already finished and deployed. Imagine merging the cool-yet-niche feature you have been working on, only to get a frantic message that something important is now intensely broken! That's a bad day at the office, right?

For this very reason, **code reviews** should occur on all **pull requests**. Code reviews at Epicodus are modeled after reviews in the workplace. So let's learn a little more about them!

Pull Requests and Code Reviews

An existing codebase will likely have several branches — a main branch, a development branch, and various feature branches. Say you're tasked with integrating a calendar into an existing application. To do so, you must add a `startTime` property to `Calendar` objects. However, you wouldn't add this code to either the main branch (you don't want to break the code in production) or the development branch. Instead, you'd create a feature branch for the code. The next step would be to write (and test) the code in this branch. But then what happens once you are done?

Well, in order to catch any conflicts with existing code or other potential issues, the next step is to submit a pull request. A **pull request** is a request to "pull" your new code into the main (or development) version of the existing codebase. When you submit a

pull request, a senior developer (or equivalent) reviews the new code carefully, offering comments with constructive feedback or detailing any potential issues.

If your code is accepted, the pull request will be merged into the codebase. If more revision is needed, your senior developer will likely reject your pull request and may ask you to review code according to feedback. If the new feature being implemented affects many other areas of the codebase, your pull request may be reviewed by more than one senior dev.

Pull requests can be intimidating but they don't have to be! Here are tips for making a successful first pull request.

Overarching Principle

Your primary goal is to minimize the amount of time your senior must spend reviewing your work.

This does **not** mean you should avoid doing work, developing new features, and so on! It **does** mean you should spend time making your code DRY, clear, and efficient. An experienced developer can quickly tell the amount you have accomplished and how polished your code is. Similarly, an Epicodus teacher can quickly differentiate a fantastic project from a mediocre one. Your code must be error free, complete, and easy to evaluate. It should be as perfect as you can reasonably get it.

The less time a senior developer needs to spend fixing your code, the more valuable you are as an employee. Conversely, the more time your senior developer has to spend pointing out bad code, bugs, flaws, projects that don't build, commits that mix up broken and working code, typos, missing tests, tests that don't pass, forgotten API keys, obvious copy/pasting, and so on, the **less** valuable you'll be as an employee because you need **more** supervision. Supervision takes time. Time costs money. This is

especially important if you or your senior developer are blocked from completing additional work while your pull request is pending. Stand out through your consistency, simplicity, and hard work.

- **One feature per pull request** (One feature per commit). Avoid rolling fixes on different issues into one commit (and therefore one pull request). It's easier for your senior developer to review code if they know it's dedicated to fixing one issue or implementing one feature. Imagine a commit containing both working and broken code! A nightmare to unravel — and you will likely be tasked with redoing it. Redoing means that you are costing your employer money. Make separate commits instead.
- **Write clear code documentation.** Make clear Git commit messages and clear comments for confusing code segments. But do not bury your code in a wall of comments, either. Comments should stand out and serve a purpose. Ideally, well-written code with well-named variables, parameters and function names should be self-documenting — no comments needed!
- **Carefully review code for scalability.** What happens if I try to add a new property or functionality to an object? Or add a second type of event that needs a different kind of calendar? Should I already be thinking of an end date? These are the things your senior will notice. Make sure you *indicate that you have been thinking about these issues*.
- **Review your code** for typos, inconsistencies in naming, spacing, indentation, and formatting.

If your pull request is rejected:

- **Don't get angry**, focus on your self-perceived inadequacies, or quit your job. It's not personal, it's about maintaining the integrity of the codebase to prevent issues down the line.

- **Read the feedback carefully** before implementing recommended changes. Understand what you are being asked to do and why.
- **Ask for additional explanations** from your senior developer if their explanations are not 100% clear.
- **Try to continually improve the code's quality.** Do not compare yourself to others in your team. Focus instead on understanding what you must learn to improve and advance. Ask for guidance from your team to pinpoint specific areas for improvement.

We strongly recommend this enlightening blog post (https://medium.com/turbine-labs/theres-no-hell-in-team-4f5d6f3ff511) by Portland Developer Relations Manager Brook Shelley. It covers many topics, not just pull requests, and is a quick, insightful, and entertaining read!

During your time at Epicodus: Try and implement the guidelines for effective pull requests described above into your individual work and group projects as much as possible. Focus on making your projects quick and efficient to grade. You can do this by having a thorough commit history, commenting as needed, making sure you have a great README, and treating the instructors reviewing your code as if they are senior devs looking at a pull request — which means you should have high standards for the work you submit.

During your internship or job: Ask if there are any specific things you should know about your workflow or submitting pull requests before you submit your first PR.

Previous (/intermediate-javascript/asynchrony-and-apis/further-exploration-chaining-promises)
Next (/intermediate-javascript/asynchrony-and-apis/sprucing-up-your-github)

disable dark mode

(http://www.epicodus.com)