
iX
Embrace Opportunity
iX
iXPERIENCE

Software Engineering Day 17

Advanced Software Development Practices

i_X

Today's Overview

- Pull requests using github
- Code Reviews.
- Testing with Jest.

- Cheat Sheet:
  - iX Cheat Sheet
  - iX Cheat Sheet - Day 17

Pull Requests (PR)

GitHub.

i_X

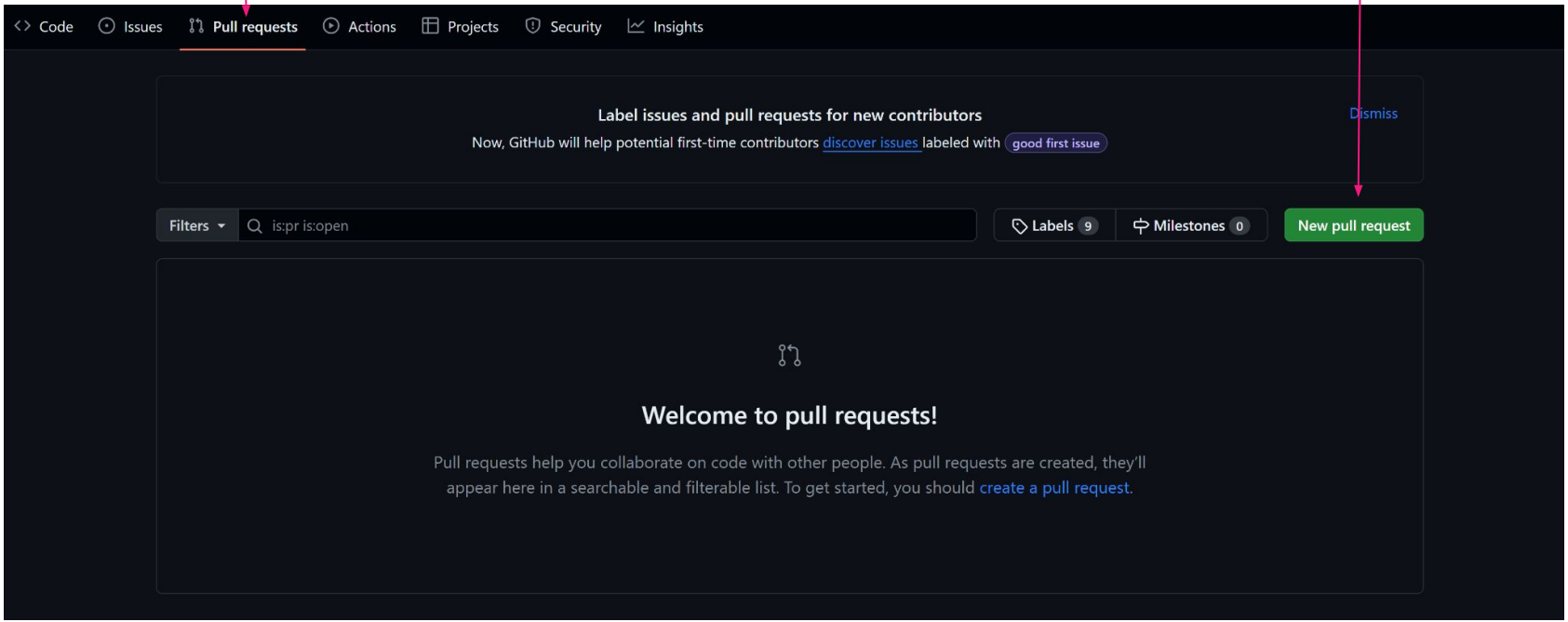
Pull Requests - Introduction

- A pull request (PR) is a proposal to merge a set of changes from one branch into another.
- They display the differences between the commits of the two branches.
- Allows collaborators to review the differences:
  - They can approve / deny or suggest changes to the author.
- Powerful tool when working in a team, to keep the main branches up to standard and minimizes problems.

Pull Requests - GitHub - UI

Pull Request - TAB

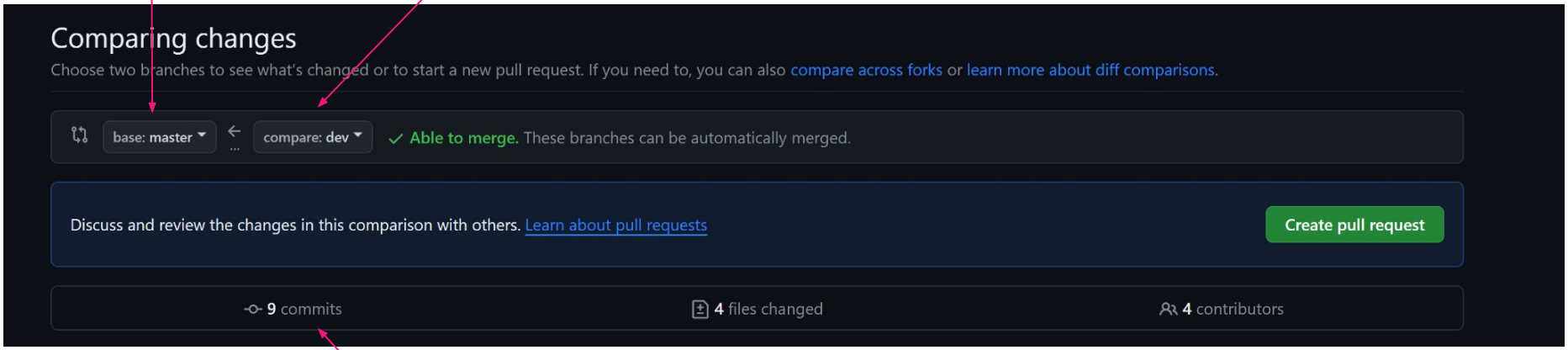
Creation



Pull Requests - GitHub - UI

Main Branch

Feature Branch



Commit Differences Between Branches

i_X

Pull Requests - GitHub - UI

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base: master ← compare: myFeatureBranch ✓ Able to merge. These branches can be automatically merged.

Add a title

New feature added

Add a description

Write | Preview

Ticket:

Work done:

Markdown is supported | Paste, drop, or click to add files

Create pull request

Reviewers
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Use Closing keywords in the description to automatically close issues

Helpful resources

Pull Requests - Creation

- After pushing your feature branch to the repository.
- Go to the pull request tab.
- Create pull request, between the feature branch and main branch.
- Pull Request includes:
  - Title
  - Description:
    - Include the ticket that was worked on
      - If working with management system, such as Agile methodology.
    - Include the explanation of the work done for the ticket to be completed.

Pull Request - Best Practices

- Keep one feature/fix/removal/ticket to one PR.
  - Fulfil a single purpose.
- Provide clear context and guidance for reviewers.
  - Purpose of the pull request.
  - Overview of what changed.
  - Links to any additional context such as tracking issues or previous conversations.
- Have clear commit messages.
- As repository maintainer:
  - Utilize protected branches to prevent unwanted merges.
  - Apply PR rules in order to protect branches, eg:
    - The amount of reviews needed to be able to merge the PR.

Code Review

Source Code QA

i_X

Code Review - Introduction

- Code review is an important part of the development process.
  - Also known as peer reviews.
- Acts as quality assurance of the code base.
  - Allows for a second set of trained eyes to identify any shortcomings or possible bugs that may happen in the future.
- Setting a code review process sets a foundation for continuous improvements.
  - Allows the main branch to keep a quality code base.
- Advantages:
  - Share knowledge
  - Bug discovery
  - Maintain coding standards
  - Increase security

Code Review - Methods

- Ensure the highest quality code and standard:
  - Tool-assisted reviews:
    - Automated tools to enforce code standards, identify vulnerabilities, gather metrics and gather files.
    - These tools do have to be maintained by the developers.
    - Team reviews are still suggested to fully ensure highest standard.

Code Reviews - Best Practices

- Limit review sessions
  - Single team member shouldn't spend too much time reviewing
  - Limit the amount of lines to review per member.
- Team inclusivity
  - Code reviews can have both junior and seniors review
  - Can be used as a great tool to help newer developers understand best practices and how problems are solved.
  - This is also gets the whole team into a single processes.
- Comment and questions:
  - Explain suggestions so the author has a clear plan to work off.
  - Ask questions during review to have better understanding of the changes you are making before making suggestions.

Code Review - Key Points

- When reviewing code, the key points to look for:
  - Design
    - Does the change make sense
  - Functionality
    - Does it fulfil the requirements
  - Complexity
    - Does the complexity match the problem being solved
  - Tests
  - Naming
    - Logical naming scheme
  - Style
    - Keep the coding standard set for the team

Testing

With Jest

i_X

Testing - Jest

- Jest is a JavaScript unit testing framework (Works with Node / React)
  - Official Documentation
- Unit testing:
  - Verify small parts of applications in complete isolation.
  - Ensuring expected outcomes.
  - Cannot interact with external dependencies
- Jest doesn't rely on third-party tools.
- Due to Jest being a framework and not a library, it offers:
  - Test runner
  - Assertion library
  - CLI tool

Testing - Jest

- Advantages:
  - Automatically runs affected tests for commits
  - Syntax to skip tests or run a single test, helpful during debugging a single component.
  - Brings easy mocking to be utilised.

Testing - Jest - Installation

- Global installation:

```sh
npm install -g jest
```

- Project installation:

```sh
npm install --save-dev jest
```

Testing - Jest - Installation

- React Installation:

```sh
npm install --save-dev react-test-renderer
```

- Utilizing jest:
  - Inside *package.json*
    - Make the change below:

```
"scripts": {
    "test": "jest"
}
```

Testing - Jest - Basics

- Creating test files:
  - Jest automatically identifies *.test* suffix (as well as *.spec* suffix)
    - Testing your *index.js* page, you'll create *index.test.js*
- Testing syntax:
  - *test()* function:
    - To start off defining the test
    - Takes two parameters
      - Test description - distinctive name that is shown when running the test
      - Callback - containing the actual test code
  - expect() function:
    - Callback function - takes the function as a parameter.

Testing - Jest - Basics - Example

```javascript
// index.js
function isPalindrome(string) {
    let left = 0;
    let right = string.length - 1;

    while (left < right) {
        if (string[left] === string[right]) {
            left += 1;
            right -+ 1;
        }
        else return false;
    }
    return true;
}
```

```javascript
// index.test.js
isPalindrome = require('./index,js');

test('kayak is palindrome', () => {
    expect(isPalindrome('kayak')).toBe(true);
})

test('lesson is not palindrome', () => {
    expect(isPalindrome('lesson')).toBe(false);
})
```

i_X

Testing - Jest - Basics - Example Output

```sh
# console logs
> jest PASS
./index.test.js
✓ kayak is palindrome (1 ms)
✓ lesson is not palindrome

Test Suites: 1 passed, 1
total Tests:       2 passed, 2
total Snapshots:   0
total Time:        0.252 s
Ran all test suites.
```

Test file

Distinctive names

Success rate

UI serialized value generated

Test elapsed time

Testing - Jest - Mock Functions

- Mock functions allow you to test the links between code by erasing the actual implementation of a function, capturing calls to the function (and the parameters passed in those calls), capturing instances of constructor functions when instantiated with new, and allowing test-time configuration of return values.
- Two mock function methods:
  - Creating a mock function
  - Writing a manual mock to override a module dependency

Testing - Jest - Mock Functions - Example

```
// forEach.js
export function forEach(items, callback) {
    for (const item of items) {
        callback(item);
    }
}
```

Testing - Jest - Mock Functions - Example

```
// forEach.test.js
const forEach = require ('./forEach');
const mockFn = jest.fn(x => 42 + x);

test('forEach mock function', () => {
    forEach([0, 1], mockFn);
    //Expect mock function to be called twice
    expect(mockFn.mock.calls).toHaveLength(2);
})
```

Testing - Jest - Matchers

- Matchers:
  - Lets you test values in different ways
  - Creating assertions in combination with the *expect* keyword
    - Compare the output of our test with a value that expected to be returned by the function.
  - Official Documentation
  - The matcher follows the *expect()* function
    - In the basic example the matcher is *toBe()*

Testing - Jest - Matchers

```
// Compare primitive values
.toBe()

// Compare recursively all properties of object instances
.toEqual()

// Compare the truthiness of result (Doesn't care what the value is)
// All values are truthy unless defined as falsy below
.toBeTruthy() //Ensure the value is true
// Falsy = false, 0, '', null, undefined, NaN
.toBeFalsy() //Ensure the value is false

// Compare mathematical operations
.ToBeGreaterThan()
.ToBeLessThanOrEqual()
.ToBeGreaterThanOrEqual()

// Compare string to match a regular expression
.toMatch()
```

Testing - Jest - Setup and Teardown

- Setup and Teardown:
  - Jest provides helper functions to handle setup work before tests run, and finishing work after tests are run.
    - Prepare and cleanup.
  - Functions:
    - For repeat testing:
      - *beforeEach* and *afterEach* hooks
    - Only run once:
      - *beforeAll* and *afterAll* hooks
    - Scoping:
      - Hooks declared inside a *describe* block only apply to the tests within that block.
      - Allows to create groups for the testing report

Testing - Jest - Setup and Teardown

- Repeat and single runs:

```
// Runs before and after each test in a suite
beforeEach(() => {});
afterEach(() => {});
// Performed once per test suite
beforeAll(() => {});
afterAll(() => {});
```

Testing - Jest - Setup and Teardown

- Scoping test file:

```
// index.test.js

isPalindrome = require('./index,js');

describe('isPalindrome function', () => {
    test('kayak is palindrome', () => {
     expect(isPalindrome('kayak')).toBe(true);
    });

    test('lesson is not palindrome', () => {
        expect(isPalindrome('lesson')).toBe(false);
    });
});
```

Testing - Jest - Setup and Teardown

- Scoping report:

```sh
# console logs
> jest
PASS  ./index.test.js
  isPalindrome function
    ✓ kayak is palindrome (1 ms)
    ✓ lesson is not palindrome

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.221 s
Ran all test suites.
```

Testing - Jest - Snapshot

- Snapshot testing in Jest is used to detect UI changes, applies for React front ends, checking large objects, even JSON responses.
- Typical snapshot test case:
  - Renders a UI component, takes a snapshot, then compares it to a reference snapshot file stored alongside the test.
- Snapshots are useful:
  - Instead of rendering the graphical UI, which required building the app.
  - The test will generate a serializable value for your React tree.

Testing - Jest - Snapshot

- Create snapshot testing:
  - On first test run the snapshot will be generated
  - On Subsequent tests, Jest will compare the rendered output with the previous output.

```
import renderer from 'react-test-renderer';
import Link from '../Link';

test('renders correctly', () => {
  const tree = renderer
    .create(<Link page="http://www.jestjs.io">Jest</Link>)
    .toJSON();
  expect(tree).toMatchSnapshot();
});
```

Testing - Jest - Snapshot

- Generated snapshot
  - File directory: */__tests__/__snapshots__/*

```
exports[`renders correctly 1`] = `
<a
  className="normal"
  href="http://www.jestjs.io"
  onMouseEnter={[Function]}
  onMouseLeave={[Function]}
>
  Jest
</a>
`;
```

Testing - Jest - Extra

- Useful cheat sheet for Jest testing:
  - Cheat Sheet

Homework

Apply what we have learned

Homework

- Add unit tests to capstone project.
- Pre-reading Chapter 2, 4 and 6
  - Cracking the coding interview by Gayle Laakmann McDowell
    - Found on Beeline.

Next Class

Resume Building & Interview Preparation