Skill Matrix: Junior Developer - Frontend

Entry-level role focused on building responsive, user-friendly web interfaces. Collaborates with design teams to implement layouts and improve user experience. Familiarity with basic front-end frameworks, accessibility, and browser compatibility is essential.

Required Skills:

- Framework
  - Angular
- Common
  - Core
  - Database
  - Code Versioning
  - Unit Testing

Angular

Understands the basic concepts of Angular, can create simple components and services, and familiar with data binding and directives.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
TypeScript Fundamentals - Types	Understanding of TypeScript's type system and the ability to use basic types effectively.	Basic	**A Junior Developer can:** - Write TypeScript code using basic types and understand the benefits of type safety.	**Examples:** - Can use basic types like `number`, `string`, `boolean`, `any`, `void`, `null`, and `undefined`. - Can work with arrays, tuples, and object types. - Can define custom types using enums, interfaces, and classes. - Understands type inference and type annotations.
Angular Fundamentals - Components	Understanding of Angular's component-based architecture and the ability to create and use components.	Basic	**A Junior Developer can:** - Create simple Angular components and use basic data binding techniques.	**Examples:** - Can explain the benefits of a component-based UI architecture. - Can create components using the Angular CLI (`ng generate component`). - Understands the component lifecycle and common lifecycle hooks (e.g., `ngOnInit`, `ngOnDestroy`). - Can use data binding techniques: interpolation (`{{ }}`), property binding (`[ ]`), event binding (`( )`), and two-way binding (`[( )]`).
Angular Fundamentals - Directives	Understands Angular directives and can use built-in directives and create custom directives.	Basic	"**A Junior Developer can:** - Use basic built-in directives to control the display of elements and iterate over data."	**Examples:** - Can use common built-in directives like `ngIf`, `ngFor`, `ngSwitch`, `ngClass`, and `ngStyle` to control the structure and appearance of the DOM. - Can create custom attribute directives to modify the behavior or appearance of elements. - Can create custom structural directives to add or remove elements from the DOM.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
Angular Fundamentals - Templates	Can write well-structured and efficient Angular templates using HTML, data binding expressions, and directives.	Basic	"**A Junior Developer can:** - Write basic Angular templates using interpolation and simple directives."	**Examples:** - Can create templates that display data from component properties using interpolation. - Can use property binding and event binding to interact with component logic. - Can use directives to control the structure and behavior of the template. - Can create reusable template fragments using `<ng-template>`.
Angular Fundamentals - Modules	Understands Angular modules (NgModule) and how to organize an Angular application using modules.	Basic	**A Junior Developer can:** - Understand the basic structure of an Angular module and can create simple modules.	**Examples:** - Can create modules using the Angular CLI (`ng generate module`). - Can declare components, directives, pipes, and services within modules. - Can import and export modules to share functionality. - Understands the concept of feature modules and shared modules.
Dependency Injection (DI)	Understands the principles of Dependency Injection (DI) and how Angular's DI system works.	Basic	"**A Junior Developer can:** - Inject services into components using constructor injection."	**Examples:** - Can explain the benefits of DI (e.g., testability, maintainability, loose coupling). - Can provide services using the `@Injectable` decorator. - Can inject services into components and other services using constructor parameters. - Understands the different provider scopes (e.g., root, module).
Services and Data Access - Creating Services	Can create Angular services using the `@Injectable` decorator and provide them at different levels.	Basic	**A Junior Developer can:** - Create basic Angular services and inject them into components.	**Examples:** - Can create a service using the Angular CLI (`ng generate service`). - Can use the `@Injectable` decorator to make a class injectable as a service. - Can provide services at the root level or at the module level. - Understands the difference between providing services in `providedIn: 'root'` and providing them in a module's `providers` array.
Services and Data Access - Fetching Data with HttpClient	Understands how to use the `HttpClient` to make HTTP requests to APIs in Angular applications.	Basic	"**A Junior Developer can:** - Use `HttpClient` to make basic API requests to fetch data."	**Examples:** - Can use the `HttpClient` to make GET, POST, PUT, and DELETE requests to REST APIs. - Can set request headers (e.g., for authorization). - Can handle responses, including parsing JSON data. - Can handle errors during HTTP requests.
Services and Data Access - Observables (rxjs)	Understands how to work with observables from the `rxjs` library in Angular, particularly when fetching data with `HttpClient`.	Basic	"**A Junior Developer can:** - Subscribe to observables returned by `HttpClient` and display the data."	**Examples:** - Can subscribe to observables to receive data streams. - Can use RxJS operators (e.g., `map`, `filter`, `switchMap`) to transform and manipulate data streams. - Can handle errors and completion events in observables.
Forms - Template-Driven Forms	Can create and work with template-driven forms in Angular.	Basic	"**A Junior Developer can:** - Create simple template-driven forms with basic validation."	**Examples:** - Can create a form using the `ngForm` directive. - Can bind form controls to component properties using `ngModel`.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
				- Can use built-in directives like `ngSubmit`, `ngModelGroup`, and `ngIf` to control form submission and display. - Can implement basic form validation using directives like `required`, `minlength`, and `maxlength`.
Forms - Form Validation	Understands and can implement form validation in Angular, using both built-in and custom validators.	Basic	"**A Junior Developer can:** - Use built-in validators to implement basic form validation."	**Examples:** - Can use built-in validators like `required`, `email`, `minlength`, and `maxlength`. - Can create custom validators to enforce specific validation rules. - Can display validation errors to the user. - Can conditionally enable or disable form controls based on validation status.
Routing and Navigation - Angular Router	Understands how to use the Angular Router to manage navigation and routing within an Angular application.	Basic	"**A Junior Developer can:** - Define basic routes and navigate between them."	**Examples:** - Can define routes using the `RouterModule` and the `routes` array. - Can use the `routerLink` directive for declarative navigation in templates. - Can access route parameters using the `ActivatedRoute`. - Can use route guards to protect routes based on authentication or other conditions. - Can use the `Router` service for programmatic navigation.
Testing - Jasmine and Karma	Can write unit tests for Angular components, services, and other code artifacts using Jasmine and Karma.	Basic	"**A Junior Developer can:** - Write basic unit tests for components and services using Jasmine and Karma."	**Examples:** - Can use Jasmine's syntax to write test cases (`describe`, `it`, `expect`). - Can use Karma to run tests in a browser environment. - Can write tests that cover different scenarios, including edge cases. - Can use mocking to isolate units of code.
Testing - Angular Testing Utilities	Understands and can use Angular testing utilities to create test environments and simplify the testing process.	Basic	"**A Junior Developer can:** - Use basic Angular testing utilities to set up simple component tests."	**Examples:** - Can use `TestBed` to configure and create a testing module. - Can use `ComponentFixture` to interact with a component instance in tests. - Can use `DebugElement` to query and interact with DOM elements. - Can use mocking features provided by Angular testing utilities.
UI Development - HTML and CSS Fundamentals	Has a solid understanding of HTML for structuring content and CSS for styling web pages. This is a pre-requisite for styling Angular components effectively.	Basic	**A Junior Developer can:** - Create basic HTML pages with simple CSS styling.	**Examples:** - Can create HTML elements and structure a basic web page using semantic HTML tags. - Can write CSS rules to style HTML elements using selectors, properties, and values. - Understands CSS box model, positioning, and layout concepts. - Can use CSS preprocessors (e.g., Sass, Less) for more organized and maintainable stylesheets.
UI Development - Styling Angular Components	Understands different ways to style Angular components, including component styles, view encapsulation, and CSS frameworks.	Basic	**A Junior Developer can:** - Apply basic styling to components using component styles or inline styles.	**Examples:** - Can define styles for a component using the `styles` property in the component decorator. - Can use inline styles or link to external stylesheets. - Can use CSS preprocessors (e.g., Sass) for component styles.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
				- Understands view encapsulation and how it affects component styles.
UI Development - CSS Frameworks (Optional)	Can integrate and use popular CSS frameworks in Angular applications.	Basic	**A Junior Developer can:** - Use basic CSS framework components and styles.	**Examples:** - Can install and configure Bootstrap, Materialize, Bulma, or other CSS frameworks. - Can use framework components and utilities to style Angular components. - Can customize framework styles to fit project design requirements.
UI Development - Angular Material (Optional)	Can integrate and use the Angular Material library for UI components in Angular applications.	Basic	"**A Junior Developer can:** - Use basic Angular Material components to build UIs."	**Examples:** - Can install and configure Angular Material. - Can use Angular Material components (e.g., buttons, cards, dialogs, forms). - Can customize Angular Material components using theming and styles.
UI Development - Responsive Design	Understands the principles of responsive design and can create Angular applications that adapt to different screen sizes and devices.	Basic	"**A Junior Developer can:** - Understand the basic concepts of responsive design."	**Examples:** - Can use CSS media queries to apply different styles based on screen size. - Can use CSS flexbox or grid layout for responsive layouts. - Can use viewport meta tags to control how the page is displayed on mobile devices. - Can test responsive designs on different devices and browsers.
Advanced Concepts - Change Detection	Understands how Angular's change detection mechanism works and can use different change detection strategies.	Basic	**A Junior Developer can:** - Understand the basic concept of change detection in Angular.	**Examples:** - Can explain how Angular detects changes in component data and updates the view. - Understands the default change detection strategy and how it works. - Can use the `OnPush` change detection strategy to optimize performance. - Can manually trigger change detection when necessary.
Advanced Concepts - Pipes	Understands Angular pipes and can use built-in pipes and create custom pipes for data transformation.	Basic	"**A Junior Developer can:** - Use common built-in pipes to format data in templates."	**Examples:** - Can use built-in pipes like `date`, `currency`, `uppercase`, `lowercase`, and `json`. - Can create custom pipes using the `@Pipe` decorator. - Can use pipes in templates and in component code.
Advanced Concepts - Content Projection	Understands content projection in Angular and can use `<ng-content>` to project external content into components.	Basic	"**A Junior Developer can:** - Use `<ng-content>` to create simple components that accept content."	**Examples:** - Can use `<ng-content>` to create components that accept dynamic content from parent components. - Can use `select` attribute on `<ng-content>` to project specific content based on selectors. - Can use multiple `<ng-content>` tags to project content to different areas within a component.
Advanced Concepts - Performance Optimization	Understands techniques for optimizing the performance of Angular applications.	Basic	"**A Junior Developer can:** - Understand the importance of performance and can identify potential performance issues."	**Examples:** - Can identify performance bottlenecks using profiling tools. - Can use change detection strategies (`OnPush`) to reduce unnecessary re-renders. - Can use lazy loading to load modules on demand. - Can use Ahead-of-Time (AOT) compilation to improve initial

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
				load time. - Can optimize data binding and DOM manipulation for better performance.
Advanced Concepts - Tree Shaking	Understands the concept of tree shaking in Angular and how it helps reduce bundle size by removing unused code during the build process.	Basic	"**A Junior Developer can:** - Understand the concept of tree shaking and its benefits."	**Examples:** - Can explain how tree shaking works in the context of Angular and Webpack. - Understands that using ES modules and properly importing dependencies is important for tree shaking to work effectively. - Can analyze bundle sizes and identify opportunities for reducing bundle size through tree shaking.
Advanced Concepts - Linting	Understands the importance of linting and can use linters like TSLint or ESLint to enforce code style and quality standards in Angular projects.	Basic	"**A Junior Developer can:** - Understand the purpose of linting and can fix basic linting errors."	**Examples:** - Can configure a linter for an Angular project (using TSLint or ESLint). - Can define custom linting rules. - Can integrate linting into the development workflow (e.g., using a code editor plugin or running linting as part of the build process).
Advanced Concepts - Logging	Understands how to set up logging in Angular applications and can choose and use appropriate logging libraries.	Basic	"**A Junior Developer can:** - Use `console.log` for basic debugging."	**Examples:** - Can use console logging for basic debugging in development. - Can set up a logging library like Winston or Pino for more structured logging. - Can configure logging levels (e.g., debug, info, warn, error). - Can log relevant context information (e.g., timestamps, user IDs, request data).
Tooling and Workflow - Angular CLI	Proficiency in using the Angular CLI for creating, building, testing, and managing Angular projects.	Basic	"**A Junior Developer can:** - Use basic Angular CLI commands to create projects, generate components, and run the development server."	**Examples:** - Can create new Angular projects using `ng new`. - Can generate components, services, modules, and other artifacts using `ng generate`. - Can build the application for development or production using `ng build`. - Can serve the application locally using `ng serve`. - Can run unit tests using `ng test`. - Can use the CLI to add dependencies and perform other project management tasks.
Tooling and Workflow - npm or yarn	Can use npm (Node Package Manager) or yarn to manage project dependencies in Angular projects.	Basic	"**A Junior Developer can:** - Install and update packages using npm or yarn."	**Examples:** - Can install, update, and remove packages using npm or yarn. - Understands the `package.json` file and its role in managing dependencies. - Can use semantic versioning to manage package versions. - Can use npm or yarn scripts to automate tasks.
Tooling and Workflow - Webpack (understanding helpful)	Has a basic understanding of Webpack, the module bundler used by the Angular CLI to bundle and optimize Angular applications.	Basic	"**A Junior Developer can:** - Understand that Webpack is used to bundle the application."	**Examples:** - Understands the purpose of module bundlers in web development. - Can explain basic Webpack concepts (e.g., entry points, output, loaders, plugins). - Can troubleshoot common Webpack-related errors or warnings.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
				- Can analyze Webpack bundle statistics to identify optimization opportunities.

Core

Has a basic understanding of common IDEs, can write simple code, understands the importance of clear communication, and can follow documentation. Still developing knowledge in performance optimization and databases.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
HTML	**HTML syntax and semantics:** This includes knowing how to properly structure elements, use attributes, and ensure that code is valid and readable. **HTML5 features:** Understanding and utilizing features like `<canvas>`, `<audio>`, `<video>`, Web Components, and Service Workers. **Accessibility:** Creating web pages that are accessible to users with disabilities, adhering to guidelines like WCAG. **Performance optimization:** Writing efficient HTML code to improve page load times and overall performance. **SEO best practices:** Using HTML elements and attributes to optimize web pages for search engines.	Basic	**A Junior Developer can:** - Write semantic HTML that adheres to best practices. - Understand and use different HTML elements appropriately. - Create accessible web pages that meet basic accessibility guidelines.	
CSS	**CSS syntax and selectors:** This includes knowing how to use different selectors (e.g., class, ID, attribute) to target specific elements and apply styles. **CSS properties:** Understanding the various properties that can be used to control the appearance of elements, such as color, font, layout, and positioning. **CSS values:** Knowing the different data types and units used for CSS values (e.g., pixels, percentages, colors). **CSS layout techniques:** Understanding how to use CSS to create different types of layouts, such as grid, **flexbox**, and float-based layouts. **CSS preprocessors:** Familiarity with preprocessors like Sass or Less can improve code organization and maintainability. **Responsive design principles:** Creating websites that adapt to different screen sizes and devices. **Browser compatibility:** Ensuring that CSS styles work consistently across different browsers and versions.	Basic	**A Junior Developer can:** - Write CSS to style web page elements (fonts, colors, layout). - Understand basic CSS concepts (selectors, inheritance, box model). - Create responsive layouts using media queries.	
CSS Frameworks	Using CSS frameworks, Deciding on which CSS framework works best for a project.	Basic	**A Junior Developer can:** - Understand the purpose and benefits of CSS frameworks. - Use a CSS framework (e.g., Bootstrap, Tailwind CSS) to style a web page. - Customize basic framework components.	Bootstrap,Material UI, Antd Design, Semanti UI
Programming Language	Proficiency in at least one frontend programming language, including knowledge of syntax, ES6+ features, and OOP	Basic	**A Junior Developer can:** - Write basic JavaScript code. - Understand variables, data types, operators, and control flow in JavaScript. - Manipulate the DOM (Document Object Model) to create interactive elements.	Vanilla JavaScript, JavaScript with ES6, TypeScript
Frontend Frameworks	Familiarity with at least one frontend framework for building web applications	Basic	**A Junior Developer can:** - Understand the basic concepts of front-end frameworks (e.g., React, Vue, Angular).	ReactJS, Angular, VueJS, Svelte

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
			- Create simple components using a framework. - Follow tutorials and documentation to build basic applications.	
Version Control	Basic knowledge of Git for tracking changes and collaborating with team members	Basic	**A Junior Developer can:** - Understand basic Git commands and workflows (clone, branch, commit, push, pull). - Use version control to collaborate with other developers on front-end projects. - Resolve simple merge conflicts.	Git, GitHub, GitLab, Bitbucket
Basic Backend Knowledge	Understanding of basic backend concepts to improve collaboration with backend developers	Basic	**A Junior Developer can:** - Understand the basic interaction between front-end and back-end systems. - Make simple API calls to retrieve and display data. - Handle basic data manipulation and display logic on the front end.	Node.js, RESTful APIs
UI/UX Design Principles	Basic understanding of design principles, user experience, and user interface design	Basic	**A Junior Developer can:** - Have a basic understanding of UI/UX design principles. - Create simple and user-friendly interfaces. - Be open to feedback on design and usability.	Figma, Sketch, Adobe XD
Responsive Design	Ability to create responsive web designs that work on various screen sizes	Basic	**A Junior Developer can:** - Understand the importance of responsive design. - Create web pages that adapt to different screen sizes using basic media queries. - Test their designs on different devices and browsers.	CSS Flexbox, CSS Grid, Bootstrap, Tailwind CSS
Testing and Debugging	Basic debugging skills to identify and fix issues in the code. Should able to validate backend API's using a client tool before intergrating with UI code	Basic	**A Junior Developer can:** - Write basic unit tests for front-end components using a testing framework. - Understand the purpose and importance of testing front-end applications. - Follow testing guidelines and best practices within the team.	Chrome DevTools, Jest, Mocha, Postman
Accessibility	Understanding of web accessibility standards and practices to ensure inclusivity	Basic	**A Junior Developer can:** - Have a basic understanding of accessibility principles (WCAG guidelines). - Implement basic accessibility features using semantic HTML and ARIA attributes. - Test their code for basic accessibility issues.	WCAG, ARIA, Axe
Performance Optimization	Techniques for optimizing web performance for faster load times and better user experience	Basic	**A Junior Developer can:** - Understand the importance of front-end performance. - Implement basic performance optimization techniques (e.g., image optimization, minification). - Use browser developer tools to identify potential performance bottlenecks.	Lighthouse, Webpack, Bundle Analyzer
Documentation	Ability to write clear and concise documentation for code, UI flows, and project details.	Basic	**A Junior Developer can:** - Contribute to basic documentation tasks for front-end projects. - Understand the importance of clear and concise technical writing. - Follow established documentation standards within the team.	Markdown, JSDoc

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
CI/CD	Basic understanding of continuous integration and continuous deployment practices	Basic	**A Junior Developer can:** - Understand the basic concepts of CI/CD for front-end development. - Be familiar with CI/CD tools commonly used for front-end projects. - Have participated in a team that uses CI/CD for front-end development, even in a limited capacity.	Jenkins, Travis CI, CircleCI, GitHub Actions
Security Basics	Awareness of basic security principles to ensure secure coding practices	Basic	**A Junior Developer can:** - Understand common front-end security threats (e.g., XSS, CSRF). - Implement basic security measures in code (e.g., input validation, output encoding). - Be aware of secure coding best practices for front-end development.	OWASP Top 10, HTTPS, Content Security Policy
State Management	Techniques for managing state in frontend applications	Basic	**A Junior Developer can:** - Understand the basic concept of state management in front-end applications. - Use a simple state management solution (e.g., local component state). - Follow best practices for managing state within a component.	Redux, Vuex, MobX
Communication and Collaboration	Soft skills for effectively working within a team, including communication and collaboration tools	Basic	**A Junior Developer can:** - Communicate clearly and concisely in written and verbal forms. - Participate actively in team meetings and discussions. - Ask clarifying questions when needed.	Slack, Microsoft Teams, Zoom
IDE Knowledge	Basic proficiency in at least one integrated development environment (IDE) for efficient coding	Basic	**A Junior Developer can:** - Use an IDE effectively for front-end development tasks (HTML, CSS, JavaScript). - Understand the IDE's basic features for front-end development (code completion, syntax highlighting, error checking). - Install and configure plugins to enhance the IDE's front-end development capabilities.	VSCode, WebStorm, Sublime Text
Project Domain	Basic understanding of the specific domain relevant to the project	Basic	**A Junior Developer can:** • Have a basic understanding of the specific domain relevant to the project (e.g., E-commerce, Healthcare, FinTech). • Perform simple tasks under supervision. • Learn domain-specific concepts.	E-commerce, Healthcare, FinTech

Database

Can write basic SQL queries, understands database concepts like tables, rows, and columns. Familiar with at least one type of database (Relational or Document).

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
SQL	**[Pre-requisite: Basic understanding of databases]**	Understanding	**A Junior Developer can:** - Understands the basic purpose of SQL and its role in	**Examples:** - SELECT * FROM Products WHERE Category = 'Electronics' ORDER BY Price

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
	Ability to write queries to retrieve, manipulate, and filter data.		web development. - Can read and understand simple SQL queries.	DESC LIMIT 10; - UPDATE Users SET LastLogin = CURRENT_TIMESTAMP WHERE UserID = 123;
DB Fundamentals	Knowledge of core database concepts.	Understanding	**A Junior Developer can:** - Familiar with basic database terminology like tables, columns, and rows. - Understands different data types at a high level.	**Examples:** - Relational databases vs. NoSQL databases. - ACID properties of transactions. - Basic database design principles.
Data Modeling	**[Pre-requisite: Basic DB Fundamentals]** Understanding of Entity-Relationship Diagrams for data modeling.	Understanding	**A Junior Developer can:** - Can interpret simple Entity-Relationship Diagrams (ERDs). - Understands the concept of entities and relationships in a database.	**Examples:** - Understanding how user data, product data, and order data are related in an e-commerce application. - Creating a simplified ERD to represent these relationships.

Code Versioning

Understands basic Git commands (clone, add, commit, push, pull), can work with branches, and understands the importance of commit messages.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
Basics - Version control fundamentals	Understands the importance of version control systems for tracking changes in code, collaborating with other developers, and reverting to previous versions.	Intermediate	**A Junior Developer can:** - Explain what version control is and why it is important. - Make basic commits and push/pull changes to a remote repository.	**Examples:** - Can explain the benefits of using version control. - Understands the difference between local and remote repositories.
Basics - Core Git commands (init, add, commit, push, pull)	Proficiency in using essential Git commands for local and remote repository interactions.	Intermediate	**A Junior Developer can:** - Initialize a Git repository. - Stage and commit changes with messages. - Push and pull changes from a remote repository.	**Examples:** - Initializes new Git repositories (git init). - Stages changes for commit (git add). - Commits changes with meaningful messages (git commit -m). - Pushes changes to a remote repository (git push). - Pulls changes from a remote repository (git pull).
Basics - Understanding Git file states (staged, modified, untracked)	Understands the different states a file can be in within the Git workflow and how these states impact version control operations.	Basic	**A Junior Developer can:** - Stage files for a commit using git add. - Recognize the difference between staged, modified, and untracked files using git status.	**Examples:** - Can explain the difference between a staged, modified, and untracked file. - Uses `git status` to check the state of files in the working directory.
Basics - .gitignore usage	Understands the purpose and use of .gitignore files for excluding specific files and directories from version control.	Intermediate	**A Junior Developer can:** - Explain what a .gitignore file is used for. - Add entries to a .gitignore file to exclude files from tracking.	**Examples:** - Can create and configure a .gitignore file to exclude common files (e.g., log files, temporary files, build artifacts). - Understands the use of wildcards and patterns in .gitignore entries.
Basics - Viewing history: Using `git log` effectively	Capable of using the `git log` command effectively with various options to view and analyze commit history.	Intermediate	**A Junior Developer can:** - View commit history using git log.	**Examples:** - Can use `git log` with basic filtering options like `--oneline`, `--author`, and `--grep`.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
			- Understand the basic structure of a git log entry.	- Understands how to view history for a specific file or directory.
Workflows - Basic workflow (local commits, push/pull)	Follows a standard Git workflow for making changes, committing locally, and syncing with a remote repository.	Intermediate	**A Junior Developer can:** - Create a branch. - Make commits on a branch. - Push a branch to a remote repository. - Pull changes from a remote branch.	**Examples:** - Can create a new branch, make changes, commit them, and push the branch to a remote repository. - Understands the importance of keeping the local branch up-to-date with the remote branch (using `git pull`).
Workflows - Branching and merging	Understands the concept of branching in Git and can perform basic branching and merging operations.	Intermediate	**A Junior Developer can:** - Explain why branching is useful in Git. - Create a new branch and switch to it. - Merge one branch into another.	**Examples:** - Creates branches for new features or bug fixes. - Merges branches back into the main development branch.
Workflows - Cloning repositories	Can clone existing Git repositories from remote sources, setting up a local copy for development.	Basic	**A Junior Developer can:** - Clone a repository using HTTPS. - Explain the difference between cloning and forking a repository.	**Examples:** - Clones repositories using different protocols (HTTPS, SSH). - Understands the difference between cloning and forking.
Workflows - Managing remote branches (fetch, pull, push)	Understands how to interact with remote branches, fetching changes, pulling updates, and pushing local branches.	Intermediate	**A Junior Developer can:** - Fetch branches from a remote repository. - Pull changes from a remote branch. - Push changes to a remote branch. - Understand the difference between fetch and pull.	**Examples:** - Fetches changes from a remote repository without merging them immediately. - Understands the order of operations when pulling changes. - Pushes local branches to the remote repository.
Workflows - Pull Requests: Creating clear and well-structured pull requests	Can create well-structured and informative pull requests that facilitate effective code reviews and collaboration.	Intermediate	**A Junior Developer can:** - Create a pull request from a branch. - Write a clear and concise description for a pull request.	**Examples:** - Writes clear and concise pull request descriptions that explain the purpose and scope of changes. - Includes relevant information in pull requests (e.g., issue numbers, screenshots, testing instructions).
Workflows - Pull Requests: Reviewing pull requests with constructive feedback	Actively participates in code review by providing constructive feedback on pull requests, ensuring code quality and consistency.	Intermediate	**A Junior Developer can:** - Review a pull request and provide feedback on the code. - Understand the importance of code review.	**Examples:** - Reviews code for clarity, style, and adherence to standards. - Provides specific and actionable feedback to improve code quality.
Collaboration - Centralized Workflow (e.g., GitLab, Bitbucket)	Can effectively contribute to projects using a centralized Git workflow, typically involving a platform like GitLab or Bitbucket.	Intermediate	**A Junior Developer can:** - Explain the difference between a local and remote repository in the context of a centralized workflow. - Contribute code to a project hosted on a platform like GitLab or Bitbucket.	**Examples:** - Understands the role of the central repository in a centralized workflow. - Can fork repositories, create branches, commit changes, and open pull requests for review.
Collaboration - Conflict resolution	Can identify, understand, and resolve merge conflicts that occur when integrating code changes from different branches.	Intermediate	**A Junior Developer can:** - Identify a merge conflict. - Resolve a simple merge conflict.	**Examples:** - Understands the causes of merge conflicts. - Can manually resolve conflicts by editing the affected files.
Collaboration - Strategies for minimizing merge conflicts	Understands and implements strategies to reduce the likelihood of merge conflicts during collaborative development.	Intermediate	**A Junior Developer can:** - Understand the importance of pulling	**Examples:** - Communicates effectively with team members about code changes to avoid working on the same

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
			changes frequently to minimize merge conflicts.	code sections simultaneously. - Uses frequent pulls and pushes to keep branches up-to-date and identify potential conflicts early on.
Collaboration - Using different merge strategies (recursive, ours, theirs)	Understands different merge strategies available in Git and can choose the appropriate one based on the desired outcome.	Intermediate	**A Junior Developer can:** - Understands that there are different merge strategies available in Git, even if they haven't used them all.	**Examples:** - Can perform a standard recursive merge (the default). - Understands the use cases for "ours" and "theirs" merge strategies, preserving changes from one branch over the other.
Advanced Topics - Rebasing vs. merging	Understands the difference between rebasing and merging, and can choose the appropriate method based on the situation.	Intermediate	**A Junior Developer can:** - Understand the basic difference between rebasing and merging.	**Examples:** - Can explain the conceptual difference between rebasing and merging. - Understands when rebasing is preferred (e.g., to maintain a cleaner commit history) and when merging is more suitable.
Advanced Topics - Stashing and tagging	Can effectively use Git stash to temporarily save changes and Git tags to mark specific points in history.	Intermediate	**A Junior Developer can:** - Stash changes using git stash. - Apply stashed changes using git stash pop. - Create a tag.	**Examples:** - Uses `git stash` to save uncommitted changes when switching branches or addressing urgent issues. - Understands how to apply stashed changes and manage multiple stashes.
Advanced Topics - Git hooks	Has a basic understanding of Git hooks and how they can be used to automate tasks at specific points in the Git workflow.	Basic	**A Junior Developer can:** - Explain what a Git hook is and give an example of how it could be used.	**Examples:** - Aware of common use cases for Git hooks (e.g., running code linters before commits).
Advanced Topics - Conventional Commits	Understands and utilizes conventional commit messages to improve clarity and facilitate automation.	Intermediate	**A Junior Developer can:** - Write commit messages that follow a defined convention.	**Examples:** - Follows a consistent commit message format (e.g., "feat: add new feature" or "fix: resolve bug"). - Understands how to structure commit messages for easy parsing by tools.
Advanced Topics - Git LFS	Understands the purpose and basic usage of Git Large File Storage (LFS) for managing large files within a Git repository.	Understanding	**A Junior Developer can:** - Explain what Git LFS is and why it's useful.	**Examples:** - Aware of the challenges of storing large files directly in Git. - Understands that Git LFS can be used to manage large files more efficiently.
Advanced Topics - Git Pruning	Aware of the concept of Git pruning and how it can be used to clean up unreachable objects in the repository's history.	Understanding	**A Junior Developer can:** - Understands that Git pruning is a maintenance task that can help reduce the size of a repository.	**Examples:** - Understands that Git pruning removes objects that are no longer referenced by any branches or tags.
Advanced Topics - Git Branching Strategies	Understands different branching strategies and can recommend or implement a suitable strategy for a project.	Basic	**A Junior Developer can:** - Can explain the basic concept of a branching strategy and why it's important.	**Examples:** - Familiar with common branching strategies like Gitflow, feature branching, and trunk-based development. - Understands the trade-offs of each strategy.
Advanced Topics - Git Bisect	Understands the purpose and basic usage of `git bisect` for identifying the commit that introduced a bug.	Understanding	**A Junior Developer can:** - Explain the purpose of `git bisect`.	**Examples:** - Aware that `git bisect` is a powerful tool for debugging and finding the root cause of issues.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
Advanced Topics - Git Cherry-pick	Understands the purpose of `git cherry-pick` and can use it to apply specific commits from one branch to another.	Basic	**A Junior Developer can:** - Understand the basic concept of cherry-picking.	**Examples:** - Aware of situations where cherry-picking commits can be useful (e.g., applying hotfixes).
Advanced Topics Git Sub modules	Basic awareness of Git submodules, their purpose, and how they can be used to manage dependencies within a Git repository.	Understanding	**A Junior Developer can:** - Have a basic understanding of what a Git submodule is.	**Examples:** - Aware that submodules allow for embedding external repositories into a project.
Advanced Topics - Reverting commits (git revert)	Understands how to use `git revert` to undo changes introduced by specific commits in a safe and controlled manner.	Basic	**A Junior Developer can:** - Understand that git revert is a safe way to undo changes.	**Examples:** - Aware that `git revert` creates a new commit that undoes the changes of a previous commit, preserving the original commit history.
Advanced Topics - Resetting to previous commits (git reset)	Understands the use of `git reset` to move the HEAD and potentially the branch to a specific commit, undoing changes.	Intermediate	**A Junior Developer can:** - Understand that git reset is a way to undo changes (though should be used with caution).	**Examples:** - Can use `git reset` to undo local changes that haven't been committed yet.
Advanced Topics - Understanding and using the reflog for recovering lost commits	Understands the concept of the reflog and can utilize it to recover lost commits or revert to previous states of the repository.	Basic	**A Junior Developer can:** - Understand that the reflog exists and can be used to potentially recover lost commits.	**Examples:** - Aware of the reflog as a safety net for tracking changes to the HEAD of branches.
Understanding the Bigger Picture - Distributed vs. centralized VCS	Understands the key differences between distributed and centralized version control systems and their implications.	Intermediate	**A Junior Developer can:** - Can explain the basic difference between a distributed and centralized VCS.	**Examples:** - Can explain the benefits and drawbacks of both distributed and centralized models.
Understanding the Bigger Picture - Data structures in Git (.git folder)	Has a basic understanding of the internal structure of a Git repository, particularly the contents of the .git folder.	Understanding	**A Junior Developer can:** - Be aware that the .git folder contains important information about the Git repository.	**Examples:** - Aware of the key components within the .git folder (e.g., objects, refs, HEAD). - Understands that Git stores data as snapshots rather than deltas.
Understanding the Bigger Picture Git Command Line	Prefers and is comfortable using the Git command line interface for most, if not all, Git operations.	Understanding	**A Junior Developer can:** - Navigate directories and list files using the command line. - Execute basic Git commands from the command line.	**Examples:** - Familiar with common command-line shells (e.g., bash, zsh) and basic shell commands.
Understanding the Bigger Picture Git Client Tools	Understands the purpose and benefits of using Git client tools for visualizing and managing repositories.	Basic	**A Junior Developer can:** - Be familiar with at least one GUI Git client (like GitHub Desktop).	**Examples:** - Familiar with popular Git client tools (e.g., Sourcetree, GitKraken, GitHub Desktop). - Has used at least one Git client tool for basic operations.
Security - Setting up and managing SSH keys for secure authentication	Understands the importance of SSH keys for secure authentication with Git repositories and can generate, manage, and use them effectively.	Basic	**A Junior Developer can:** - Use SSH keys to authenticate with a remote repository.	**Examples:** - Can generate SSH key pairs. - Understands the difference between public and private keys and how they are used for authentication.
Security - Protecting sensitive information: Best practices	Understands and follows best practices to avoid accidentally committing sensitive information (like passwords, API keys) to version control.	Basic	**A Junior Developer can:** - Understand that sensitive information should never be committed directly to version control.	**Examples:** - Aware of the risks of storing sensitive information in version control.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
Performance - Optimizing Git for large repositories	Aware of the challenges associated with large Git repositories and understands basic optimization techniques.	Understanding	**A Junior Developer can:** - Be aware of the potential performance issues associated with large repositories.	**Examples:** - Understands that large repositories can impact performance.
Performance - Garbage Collection	Understands the concept of garbage collection in Git and how it helps optimize repository size and performance.	Understanding	**A Junior Developer can:** - Understand that Git has a garbage collection mechanism.	**Examples:** - Aware that Git performs garbage collection to remove unreachable objects.
Tooling - Advanced use of GitKraken for complex merging (or similar for chosen GUI)	Can effectively use the advanced features of GitKraken or a similar GUI client for handling complex merging scenarios.	Basic	**A Junior Developer can:** - Perform basic Git operations using a GUI client.	**Examples:** - Familiar with the basic interface and features of GitKraken or a chosen Git GUI client.
Tooling - Integrating Git with CI/CD pipelines	Understands how Git integrates with Continuous Integration/Continuous Deployment (CI/CD) pipelines and can set up basic integrations.	Basic	**A Junior Developer can:** - Have a general understanding of CI/CD and how Git is involved.	**Examples:** - Aware of the role of Git in CI/CD workflows.

Unit Testing

Understands the importance of unit testing, can write basic unit tests for simple functions and classes.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
Understanding Unit Testing Fundamentals - What is Unit Testing?	Understands the definition, purpose, and benefits of unit testing.	Basic	**A Junior Developer can:** - Define unit testing and its role in software development.	**Examples:** - Can explain what unit testing is and why it is important. - Can describe the advantages of unit testing, such as finding bugs early, improving code quality, and facilitating refactoring. - Can differentiate between unit tests, integration tests, and end-to-end tests.
Understanding Unit Testing Fundamentals - Test-Driven Development (TDD)	Understands the Test-Driven Development (TDD) process and its benefits and challenges.	Basic	**A Junior Developer can:** - Understand the basic concept of TDD.	**Examples:** - Can describe the Red-Green-Refactor cycle of TDD. - Can explain the advantages and disadvantages of using TDD.
Understanding Unit Testing Fundamentals - Anatomy of a Unit Test	Understands the typical structure of a unit test and the concepts of test fixtures and setup/teardown methods.	Basic	**A Junior Developer can:** - Write unit tests that follow the Arrange-Act-Assert structure.	**Examples:** - Can describe the Arrange-Act-Assert pattern of unit tests. - Understands the purpose of test fixtures and how to use them. - Can explain the role of setup and teardown methods in unit tests.
Writing Effective Unit Tests - Test Case Design	Can design effective test cases using various techniques to ensure comprehensive test coverage.	Intermediate	**A Junior Developer can:** - Write test cases that cover basic scenarios.	**Examples:** - Can apply equivalence partitioning to divide input data into representative classes for testing. - Can use boundary value analysis to test edge cases and boundary conditions. - Can apply error guessing to anticipate potential errors and write tests for them.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
Writing Effective Unit Tests - Assertions	Understands assertion libraries and can use them to verify expected outcomes in unit tests.	Basic	**A Junior Developer can:** - Use basic assertions to check expected values.	**Examples:** - Can use assertion methods from testing frameworks or libraries. - Understands different types of assertions (e.g., equality, truthiness, exceptions). - Can write clear and meaningful assertion statements.
Writing Effective Unit Tests - Code Coverage	Understands the importance of code coverage, can use tools to measure it, and can interpret coverage reports.	Basic	**A Junior Developer can:** - Understand the concept of code coverage and why it is important.	**Examples:** - Can explain what code coverage is and why it is valuable. - Can use code coverage tools (e.g., SonarQube, JaCoCo, Istanbul). - Can analyze code coverage reports to identify areas of the codebase that are not well-tested.
Working with Test Doubles - What are Test Doubles?	Understands different types of test doubles (mocks, stubs, fakes, spies) and can choose and use the appropriate type for a given testing scenario.	Intermediate	**A Junior Developer can:** - Understand the concept of test doubles and when they might be useful.	**Examples:** - Can explain the differences between mocks, stubs, fakes, and spies. - Can identify situations where using test doubles is beneficial. - Understands the trade-offs of using different types of test doubles.
Working with Test Doubles - Mocking Frameworks	Familiarity with popular mocking frameworks and can use them effectively to create and manage test doubles.	Intermediate	**A Junior Developer can:** - Create simple mocks using a mocking framework.	**Examples:** - Can use mocking frameworks like Mockito (Java), Moq (.NET), Rhino Mocks (.NET), or Jest (JavaScript). - Can create mocks and stubs with the framework. - Can verify interactions with mocked dependencies.
Working with Test Doubles - Mocking Techniques	Can apply various mocking techniques to isolate dependencies, simulate specific behaviors, and verify interactions.	Intermediate	**A Junior Developer can:** - Set up mocks with basic expectations.	**Examples:** - Can set up mock objects with predefined return values or exceptions. - Can verify that specific methods were called on mock objects. - Can configure mocks to throw exceptions under certain conditions.
Best Practices and Advanced Techniques - Writing Clean and Maintainable Tests	Understands the importance of writing clean, readable, and maintainable unit tests, and can apply best practices for doing so.	Intermediate	**A Junior Developer can:** - Write unit tests that are readable and follow basic naming conventions.	**Examples:** - Writes tests that are focused and test only one thing at a time. - Uses clear and descriptive names for test methods and variables. - Avoids code duplication in tests by using helper methods or setup/teardown methods effectively.
Best Practices and Advanced Techniques - Testing Asynchronous Code	Can effectively write unit tests for asynchronous code using techniques appropriate for the chosen language and testing framework.	Intermediate	**A Junior Developer can:** - Write basic tests for asynchronous code using callbacks or Promises.	**Examples:** - Can use async/await (in languages that support it) to write asynchronous tests. - Can use callbacks or Promises to handle asynchronous operations in tests. - Understands how to synchronize test execution with asynchronous operations.

Appendix

- **Level:**

Name	Description
Understanding	Know What It Is And Have A Basic Idea Of How It Works.
Basic	Perform Simple Tasks And Understand The Fundamental Concepts.
Intermediate	Handle More Complex Tasks And Have A Good Grasp Of The Subject.
Advanced	Manage Advanced Tasks And Solve Problems Independently.
Proficient	Very Skilled And Can Handle Almost Any Task Related To This Skill.
Expert	Deep Knowledge And Can Teach Or Lead Others In This Area.