Skill Matrix: Junior Developer - Backend

Entry-level position focused on developing and maintaining server-side applications. Works under supervision to implement APIs, manage databases, and resolve technical issues. Familiarity with server environments, version control, and basic database management is important.

Required Skills:

- **Framework**
  - **Nest Js**
- **Common**
  - **Core**
  - **Database**
  - **Code Versioning**
  - **Unit Testing**

Nest Js

Understands the basic concepts of NestJS, can create simple controllers and services, and familiar with dependency injection.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
NestJS Fundamentals - Modules, Controllers, and Services	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Understanding the core building blocks of a NestJS application: modules, controllers, and services.	Basic	**A Junior Developer can:** - Create simple NestJS controllers and services to handle basic API requests. - Can organize code into modules.	**Examples:** - Can create NestJS modules to organize application code. - Can define controllers to handle incoming HTTP requests. - Can implement services to encapsulate business logic.
NestJS Fundamentals - Dependency Injection	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Understands how NestJS uses dependency injection (DI) to manage dependencies between components.	Basic	**A Junior Developer can:** - Use dependency injection to inject services into controllers.	**Examples:** - Can inject dependencies into controllers, services, and other components using constructor injection. - Understands how to provide dependencies at different levels (module, global).
NestJS Fundamentals - Decorators	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Proficiency in using NestJS decorators for routing, request handling, validation, and other features.	Basic	**A Junior Developer can:** - Use basic NestJS decorators to define routes, request handlers, and simple data validation.	**Examples:** - Can use decorators like `@Get()`, `@Post()`, `@Body()`, `@Query()`, `@Param()` for routing and request handling. - Can use validation decorators (e.g., `@IsString()`, `@IsNumber()`) to validate incoming data. - Understands custom decorators and how to create them.
NestJS Fundamentals - Pipes, Guards, and Interceptors	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Understands and can effectively use NestJS pipes, guards, and interceptors to extend and customize functionality.	Intermediate	**A Junior Developer can:** - Use built-in pipes for common tasks like data transformation.	**Examples:** - Can create and use pipes for data transformation and validation. - Can implement guards for authorization and access

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
				control. - Can use interceptors for logging, error handling, or modifying request/response objects.
NestJS Fundamentals - Exception Filters	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Understands exception filters and can use them to handle errors and exceptions gracefully in NestJS.	Basic	**A Junior Developer can:** - Use the built-in exception handling mechanisms in NestJS to handle common errors.	**Examples:** - Can create custom exception filters to catch and handle specific types of errors. - Understands how to use the `HttpException` class to throw HTTP exceptions with appropriate status codes. - Can provide custom error responses to clients.
NestJS Fundamentals - Configuration	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Understands how to manage configuration in NestJS applications using environment variables, configuration files, or other methods.	Basic	**A Junior Developer can:** - Access and use basic configuration settings from environment variables.	**Examples:** - Can access environment variables using the `process.env` object. - Can use the `@nestjs/config` package to manage configuration from files or other sources. - Can create configuration modules for better organization.
NestJS Fundamentals - CLI	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Familiarity with the NestJS CLI for generating code, managing projects, and running common tasks.	Basic	**A Junior Developer can:** - Use the NestJS CLI to generate basic application components (controllers, services, modules). - Can start and stop the application using CLI commands.	**Examples:** - Can use the CLI to create new NestJS projects (`nest new`). - Can generate controllers, services, and modules using CLI commands. - Can run the application in development mode (`nest start`).
ORMs and Data Access - TypeORM (or alternative)	**[Pre-requisite: Solid understanding of Node.js, TypeScript, and basic database concepts]** Proficiency in using TypeORM (or another chosen ORM like Sequelize) for database interactions in NestJS.	Basic	**A Junior Developer can:** - Define simple TypeORM entities and use repositories for basic database operations (CRUD).	**Examples:** - Can define entities using TypeORM decorators. - Can use repositories to perform CRUD operations. - Can build complex queries using the TypeORM query builder. - Can create and run database migrations.
ORMs and Data Access - Relationships	**[Pre-requisite: Solid understanding of Node.js, TypeScript, and basic database concepts]** Understands how to define and work with relationships between entities using TypeORM (or another chosen ORM).	Basic	**A Junior Developer can:** - Define simple one-to-many relationships between TypeORM entities.	**Examples:** - Can define one-to-many, many-to-one, and many-to-many relationships between entities. - Can use eager and lazy loading to manage data retrieval in relationships.
ORMs and Data Access - Migrations	**[Pre-requisite: Solid understanding of Node.js, TypeScript, and basic database concepts]** Understands database migrations and can use TypeORM (or another chosen ORM) to create and manage migrations.	Basic	**A Junior Developer can:** - Create and apply basic TypeORM migrations to add or modify database tables and columns.	**Examples:** - Can create new migrations to modify the database schema. - Can run migrations to apply changes to the database. - Can rollback migrations to undo changes.
Health Checks - Implementing Health Checks	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Can implement health check endpoints in NestJS applications to monitor service availability.	Basic	**A Junior Developer can:** - Implement a basic health check endpoint that returns a status code indicating the health of the service.	**Examples:** - Can create a simple health check endpoint that returns a 200 OK status if the service is healthy. - Can include checks for database connectivity, external service availability, or other critical dependencies.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
Testing - NestJS-Specific Testing	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Understands how to write unit and integration tests for NestJS applications using the NestJS testing module.	Basic	**A Junior Developer can:** - Write simple unit tests for NestJS controllers and services.	**Examples:** - Can use the `@nestjs/testing` module to create test modules and inject dependencies. - Can use the `Test` class to create test cases and run assertions. - Can mock dependencies using the NestJS testing module.
Authentication and Authorization (NestJS-Specific)	**[Pre-requisite: Solid understanding of Node.js, TypeScript, and authentication/authorization concepts]** Can implement authentication and authorization mechanisms within a NestJS application.	Basic	**A Junior Developer can:** - Set up basic authentication and authorization for a NestJS API.	**Examples:** - Can use Passport.js or other authentication libraries with NestJS. - Can implement JWT-based or session-based authentication. - Can use guards and roles to control access to resources.
Error Handling (NestJS-Specific)	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Understands NestJS-specific error handling mechanisms and can implement effective error handling strategies.	Basic	**A Junior Developer can:** - Handle common errors using NestJS's built-in exception handling mechanisms.	**Examples:** - Can use exception filters to catch and handle specific errors. - Can create custom error classes that extend `HttpException`. - Can use middleware to handle errors globally in a NestJS application.
Logging (NestJS-Specific)	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Understands how to implement logging in a NestJS application effectively.	Basic	**A Junior Developer can:** - Add basic logging statements to a NestJS application using the built-in logger.	**Examples:** - Can use the built-in NestJS logger or integrate with external logging libraries (e.g., Winston, Pino). - Can configure logging levels and output (e.g., console, file). - Can log relevant context, such as request IDs and user information.
Code Style and Linting (NestJS-Specific)	**[Pre-requisite: Solid understanding of Node.js and TypeScript]** Adheres to a consistent code style and uses linters to enforce code quality and maintainability within a NestJS application.	Basic	**A Junior Developer can:** - Understand the importance of writing clean and readable code. - Can run a linter and fix basic code style violations.	**Examples:** - Familiar with and follows a TypeScript style guide (e.g., Airbnb, Standard, Google). - Can configure and use a linter like ESLint with TypeScript support.

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
Back-End Programming Language	Proficiency in at least one backend programming language, including knowledge of syntax, data structures, algorithms, and object-oriented programming concepts	Basic	**A Junior Developer can:** - Demonstrate a fundamental understanding of core back-end concepts. - Write basic code in languages like Python, JavaScript, or Java. - Create simple APIs with basic functionality.	Java, NodeJS, Python, Ruby, PHP

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
			- Connect to databases and perform simple data operations. - Show a willingness to learn and apply best practices.	
Backend Frameworks	Familiarity with at least one backend framework for building web applications	Basic	**A Junior Developer can:** - Understand the basic concepts of back-end frameworks (e.g., Django, Express, Flask). - Use a framework to build a simple application. - Follow tutorials and documentation to implement features. - Demonstrate a willingness to learn and explore different frameworks.	Django, Spring, Express.js, Rails, Flask
Front-End Basics	Basic understanding of front-end technologies to improve collaboration with front-end developers	Basic	**A Junior Developer can:** - Demonstrate basic understanding of HTML, CSS, and JavaScript. - Create simple user interfaces and integrate them with back-end logic. - Work collaboratively with front-end developers.	HTML, CSS, JavaScript, React, Angular
Version Control	Basic knowledge of Git for tracking changes and collaborating with team members	Basic	**A Junior Developer can:** - Understand the basics of Git and version control. - Clone repositories, create branches, commit changes, and push to remote repositories. - Resolve simple merge conflicts. - Follow established version control workflows within the team.	Git, GitHub, GitLab, Bitbucket
Database Knowledge	Understanding of how databases work and how to interact with them using SQL or NoSQL	Basic	**A Junior Developer can:** - Write basic SQL queries (SELECT, INSERT, UPDATE, DELETE). - Understand database relationships (one-to-one, one-to-many, many-to-many). - Interact with databases using basic commands and tools. - Understand basic database design principles.	MySQL, PostgreSQL, MongoDB, Redis, SQLite
API	Basic skills in creating and consuming APIs for communication between software components	Basic	**A Junior Developer can:** - Understand the basics of RESTful API design principles (HTTP methods, status codes). - Build simple APIs that handle basic CRUD (Create, Read, Update, Delete) operations. - Use tools to test API endpoints. - Be familiar with API documentation formats (e.g., Swagger, Postman).	RESTful APIs, GraphQL, gRPC, SOAP
ORM	Familiarity with Object-Relational Mapping tools to interact with databases using objects	Basic	**A Junior Developer can:** - Understand the basic concept of Object-Relational Mapping (ORM). - Use an ORM to interact with a database in a simple application. - Follow tutorials and documentation to perform basic CRUD operations with an ORM.	Hibernate, Sequelize, SQLAlchemy, TypeORM
Testing and Debugging	Basic debugging skills to identify and fix issues in the code	Basic	**A Junior Developer can:** - Understand the importance of testing and different types of tests (unit, integration, end-to-end). - Write basic unit tests to verify the functionality of their code. - Use testing frameworks and tools to run tests and interpret results. - Follow testing guidelines and best practices within the team.	Postman, Insomnia, SoapUI

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
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
Unit Testing	Basic knowledge of unit testing and the ability to write simple unit tests	Basic	**A Junior Developer can:** - Write basic unit tests to verify code functionality. - Use a testing framework (e.g., JUnit, pytest) to structure and run tests. - Understand the purpose and benefits of unit testing.	JUnit, Mocha, pytest, Jest
Cloud Knowledge	Basic understanding of cloud services and deployment processes	Basic	**A Junior Developer can:** - Have a basic understanding of cloud computing concepts (IaaS, PaaS, SaaS). - Be familiar with popular cloud providers (AWS, Azure, GCP). - Deploy a simple application to a cloud platform using basic services.	AWS, Azure, GCP, Heroku, DigitalOcean
Documentation	Ability to write clear and concise documentation for code, APIs, and project details	Basic	**A Junior Developer can:** - Contribute to basic documentation tasks. - Understand the importance of clear and concise technical writing. - Follow established documentation standards within the team.	Markdown, Javadoc, Sphinx, Swagger
CI/CD	Basic understanding of continuous integration and continuous deployment practices	Basic	**A Junior Developer can:** - Understand the basic concepts of CI/CD (Continuous Integration/Continuous Delivery). - Be familiar with CI/CD tools (e.g., Jenkins, GitLab CI/CD, Travis CI). - Have participated in a team that uses CI/CD, 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 security threats (e.g., SQL injection, XSS). - Implement basic security measures in code (e.g., input validation). - Be aware of secure coding best practices.	OWASP Top 10, HTTPS, basic authentication and authorization
Containerization	Basic knowledge of using containers for application deployment and development environments	Basic	**A Junior Developer can:** - Understand the basic concepts of containerization (e.g., Docker). - Build and run a Docker image. - Deploy a simple application using Docker.	Docker, Kubernetes
Code Quality Tools	Familiarity with tools for ensuring code quality	Basic	**A Junior Developer can:** - Understand the importance of code quality. - Write clean and readable code. - Follow coding conventions and style guides.	ESLint, Prettier, SonarQube
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

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
IDE Knowledge	Basic proficiency in at least one integrated development environment (IDE) for efficient coding	Basic	**A Junior Developer can:** - Use an IDE to write, edit, and debug code. - Navigate and understand the IDE's basic features (editor, debugger, project management). - Install and configure plugins to extend IDE functionality.	IntelliJ, VSCode, Eclipse, PyCharm

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]** Ability to write queries to retrieve, manipulate, and filter data.	Basic	**A Junior Developer can:** - Understand basic SQL syntax for querying data (SELECT, FROM, WHERE). - Can retrieve data from a single table based on simple conditions. - Understands basic data types and how to use them in queries.	**Examples:** - SELECT * FROM Customers WHERE Country = 'USA'; - SELECT CustomerID, SUM(OrderAmount) FROM Orders GROUP BY CustomerID;
DB Fundamentals	Knowledge of core database concepts.	Basic	**A Junior Developer can:** - Familiar with basic database terminology (tables, columns, rows, data types). - Understands the purpose of a database and its role in applications.	**Examples:** - Tables: Customers, Products, Orders - Columns: CustomerID, CustomerName, ProductName, OrderDate - Data Types: INTEGER, VARCHAR, DATE, BOOLEAN
Data Modeling	**[Pre-requisite: Basic DB Fundamentals]** Understanding of Entity-Relationship Diagrams for data modeling.	Basic	**A Junior Developer can:** - Can understand and interpret basic Entity-Relationship Diagrams (ERDs). - Can identify entities and relationships in a simple data model.	**Examples:** - Creating an ERD to represent the relationships between Customers, Orders, and Products in an e-commerce database.
ETL	**[Pre-requisite: Intermediate SQL, Basic Programming]** Understanding of the process for moving data between systems.	Understanding	**A Junior Developer can:** - Understands the basic concepts of Extract, Transform, Load (ETL). - Can assist in developing simple ETL processes.	**Examples:** - Building an ETL pipeline to extract data from a CRM system, transform it, and load it into a data warehouse. - Using tools like Apache Kafka or Apache Nifi for real-time data streaming and integration.
Version Control	**[Pre-requisite: None]** Familiarity with version control systems to manage database changes.	Understanding	**A Junior Developer can:** - Understands the importance of version control for managing database changes. - Can use basic version control commands to track changes.	**Examples:** - Using Git to track changes to database schemas, scripts, and other artifacts. - Creating branches for new features or bug fixes. - Merging changes and resolving conflicts.
Advanced DB Techs	**[Pre-requisite: Intermediate SQL, Basic DB Admin (Basic)]** Expertise in specific database platforms and their advanced features.	Understanding	**A Junior Developer can:** - Awareness of advanced database features like stored procedures, triggers, and functions. - Can assist in implementing simple database automation tasks.	**Examples:** - Working with stored procedures, functions, and triggers. - Implementing and managing database replication and clustering. - Utilizing database partitioning or sharding techniques.

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. - Understand the basic structure of a git log entry.	**Examples:** - Can use `git log` with basic filtering options like `--oneline`, `--author`, and `--grep`. - 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).

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
				- 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 changes frequently to minimize merge conflicts.	**Examples:** - Communicates effectively with team members about code changes to avoid working on the same 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.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
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.
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.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
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.
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.
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.

SKILL	DESCRIPTION	LEVEL	RESPONSIBILITIES	EXAMPLES
				- 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.

Name	Description
Expert	Deep Knowledge And Can Teach Or Lead Others In This Area.