Full Stack Overview and Git Exercises

1 Full Stack Overview

Full-stack development refers to the practice of working on both the front-end and back-end of a web application. The front-end involves creating user interfaces using technologies like HTML, CSS, and JavaScript frameworks (React, Angular, Vue). The back-end includes server-side languages (Node.js, Django, Flask, Spring Boot) and databases (SQL, NoSQL). A full-stack developer also manages APIs, deployment, security, and performance optimization. Full-stack development enables developers to work on entire applications, reducing dependencies on separate teams and improving efficiency.

2 Interpreter vs Compiler

- **Interpreter**: Translates and executes code line by line (e.g., Python, JavaScript). This allows for immediate execution but can be slower than compiled languages.
- **Compiler**: Translates the entire source code into machine code before execution (e.g., C, Java). This improves performance but requires a separate compilation step.
- **Key Differences**: Interpreters provide real-time execution useful for scripting, while compilers optimize code execution speed and efficiency for large-scale applications.

3 REPL Jupyter

- **REPL (Read-Eval-Print Loop)**: An interactive shell that allows real-time execution of code (e.g., Python, Node.js REPL). Useful for testing and debugging small code snippets.
- **Jupyter Notebook**: A web-based interactive environment for coding, visualization, and documentation. Widely used for Python, data science, and machine learning applications, enabling users to execute code in an interactive manner.

4 Agile for Developers

Agile is a software development methodology emphasizing iterative progress, collaboration, and flexibility. Developers work in short iterations called sprints, adjust to changing requirements, and prioritize working software over extensive documentation. Agile methodologies, such as Scrum and Kanban, promote continuous integration, team communication, and fast delivery of features.

5 Introduction to SDLC (Software Development Life Cycle)

SDLC is a structured process for software development, including phases like Planning, Analysis, Design, Development, Testing, Deployment, and Maintenance. It ensures systematic development with quality assurance, reducing risks and improving project efficiency. Different SDLC models, such as Waterfall, Agile, and Spiral, cater to different project needs.

6 Waterfall Model

A linear and sequential SDLC approach where each phase depends on the deliverables of the previous one. It is beneficial for projects with well-defined requirements but lacks flexibility. Each phase must be completed before moving to the next, making it less adaptable to changing needs.

7 Agile Methodology

Agile is an iterative approach with continuous feedback loops. It emphasizes collaboration, adaptability, and incremental delivery through sprints. Agile promotes customer involvement, faster development cycles, and a flexible response to changes. Unlike Waterfall, Agile allows modifications at any stage of development.

8 Agile vs Waterfall

- **Waterfall**: Sequential, rigid, documentation-heavy, best for well-defined projects with clear objectives.
- **Agile**: Iterative, flexible, feedback-driven, best for evolving projects with changing requirements. Encourages collaboration between stakeholders.

9 Story Pointing

A technique in Agile used to estimate the complexity of a task. Points are assigned based on difficulty, dependencies, and effort rather than time. It helps teams prioritize and plan workloads effectively, ensuring efficient sprint planning.

10 Scrum Ceremonies

Scrum involves four key ceremonies:

1. **Sprint Planning**: Define sprint goals, backlog items, and expected outcomes.
2. **Daily Standup**: Short daily meeting to discuss progress, blockers, and next steps.
3. **Sprint Review**: Demo the completed work and collect stakeholder feedback.
4. **Sprint Retrospective**: Reflect on what went well, what went wrong, and how to improve future sprints.

11 Git Fundamentals

Git is a distributed version control system that helps manage code changes collaboratively. Key concepts include repositories, branches, commits, merges, and remotes. Git enables tracking changes, collaboration, and rollback capabilities in case of errors. It allows multiple developers to work on a project simultaneously without conflicts.

12 Source Control Management (SCM)

- **VCS (Version Control System)**: Tracks code changes over time, enabling collaboration and history tracking.
- **CVCS (Centralized VCS)**: Single central repository (e.g., SVN). Requires a stable network connection.
- **DVCS (Distributed VCS)**: Each user has a complete repository copy (e.g., Git), allowing offline work and better redundancy.

13 Initializing a Git Repository

```
git init
```

Creates a new Git repository in the current directory.

14 Pushing to a Remote Repository

```
git remote add origin <repository_url>
git push -u origin main
```

Connects a local repository to a remote repository and uploads changes.

15 Git Commit, Branch, Merge, Push, Pull

- **Commit**: Saves changes locally `git commit -m "message"`. Commits should be atomic and meaningful.
- **Branch**: Creates an independent line of development `git branch feature-branch`. Useful for feature isolation.
- **Merge**: Combines branches `git merge feature-branch`. Ensures features are integrated into the main branch.
- **Push**: Uploads local commits to a remote repository `git push origin main`. Keeps remote repositories updated.
- **Pull**: Fetches and merges changes from a remote repository `git pull origin main`. Synchronizes local and remote branches.

16 Git Exercises

1. Initialize a new Git repository and create a README file.
2. Create a new branch, make changes, and merge it back into the main branch.
3. Clone a repository, create a new file, commit the changes, and push it to GitHub.
4. Resolve a merge conflict by modifying files manually.
5. Set up a GitHub repository, push code from a local repository, and collaborate with teammates using pull requests.
6. Revert a commit and reset the repository to a previous state.
7. Use `git stash` to save and retrieve uncommitted changes.