**1.**

**What is GitHub?**

GitHub is an online platform which allows users to store their code, track changes to their codes, and allows users to also work with others.

It is like a virtual workspace where programmers can share their work, get feedback, and build software together.

**Primary features**

The primary features include the following:

1. Repository Hosting
2. Version control
3. Branches and merging
4. Pull requests
5. Issues and bug tracking
6. GitHub actions

**Explain how it supports collaborative software development**

It uses a system called Git to keep track of all the changes made to the code, making it easier for teams to collaborate efficiently and effectively.

**2.**

**What is a GitHub Repository?**

A GitHub Repository is a place where you can upload, store and maintain files such as codes and documentation of a specific project on GitHub.

**Describe how to create a new repository**

1. Sign into your GitHub account

2. Click the "+" icon in the upper-right corner

3. Select "New repository"

4. Name your repository

5. Choose visibility (public or private)

6. Click "Create repository"

**Describe the essential elements that should be included in it.**

1. README file: Overview of the project

2. LICENSE: Legal permissions for using the code

3. .gitignore file: Specifies files to ignore in version control

4. Code files: The main project files and source code

5. Documentation: Instructions and explanations for using the project

6. Contributing guidelines: Instructions for contributing to the project

7. Issues and Pull Requests: For tracking bugs, features, and code contribution

**3.**

**Explain the concept of version control in the context of Git.**

Version control in Git refers to a tracking system that enables you to manage changes to files over time. One can go back and see what those changes were and can revert to those versions if it need be. In addition, it enables people to work on a project simultaneously and later merge those different versions of the project efficiently.

**How does GitHub enhance version control for developers?**

* It hosts repositories by providing a central place for storing and managing codes.
* Through collaborative tools such as pull requests and code reviews, it facilitates team collaboration.
* Through branch management, branching and merging workflows are simplified
* It helps manage bugs, tasks, and feature requests.
* There’s continuous integration with CI/CD tools for automated testing and deployment.
* In addition, manages permissions and contributions from multiple users.

**4.**

**What are branches in GitHub**

Branches are development lines within a repository which allow developers to work on different features, fix bugs, or have experiments without affecting the main codebase.

**Why are they important?**

Branches in GitHub are essential in that they allow developers to make changes to codes in isolation enabling them to work on different features, fix bugs, or have experiments without affecting the main codebase. This ensures the projects stability while new developments are being tested and refined.

Through branches, developers can work in teams or collaboratively i.e., one person can handle fixing bugs in the code, while another works on features.

**Describe the process of creating a branch**

Two methods can be used to create a branch

1. Go to your repository on GitHub.

2. Navigate to Branch Selector: Click on the branch selector dropdown (usually says "main" or "master").

3. Create New Branch: Type the name of your new branch in the text box.

4. Confirm: Press "Create branch" or hit Enter.

Alternatively, if one is using Git on their local machine:

1. Open Terminal: Navigate to your repository folder.

2. Create Branch: Using the command 🡪 git branch BranchName

**How to make changes on branch?**

* Switch to the branch you want to work on by using the command 🡪

git switch BranchName

* Make the changes through IDE or text editor
* Track and stage the changes 🡪 git add .
* Commit changes with a descriptive message 🡪 git commit -m “Message”
* Push the changes 🡪 git push origin BranchName

This will update the branch with the latest changes, making them available for collaboration or review.

**How to merge?**

* Check current branch 🡪 git branch
* Switch to the main branch if not on main branch
* Git fetch changes made to the repository 🡪 git fetch origin
* Pull the latest changes to ensure the local branch is up to date with the remote repository 🡪 git pull origin main

**5.**

**What is a pull request in GitHub?**

A pull request in GitHub is a request for changes made in one branch to be merged into another in a repository.

**How does it facilitate code reviews and collaboration?**

A pull request enables developers to look at proposed changes, make comments, and discuss on them. It enables them to can go through the proposed changes in the code line by line.

A pull request contains a comment section where team members contribute questions and remarks regarding the code changes, and one can propose enhancements for improvement, thus facilitating collaboration.

Everything is documented from the group discussions to the feedback thus ensuring that everyone is in agreement.

Only the changes approved by the reviewer are merged after having passed the review, which checks the viability of the code before it is merged into the main branch/project thus facilitating code reviews.

**Steps to Create a Pull Request:**

Scenario:

Let’s say you had to develop a new feature for the main software/application

1. Create a new branch where the changes will be made.

🡪 git branch NewFeature

1. Modify the code, add new features, fix bugs, etc.
2. Stage and commit the changes with a descriptive message.

Stage 🡪 git add .

Commit 🡪 git commit -m "Add new feature”

4. Push your branch to the remote repository.

🡪 git push origin new-feature

5. Open a Pull Request:

- Go to the GitHub repository and click the “Compare and pull request" button.

- Fill out the pull request form, providing a title and description of your changes.

- Submit the pull request.

**Steps to Review a Pull Request:**

1. Open the Pull Request:

- Navigate to the repository on GitHub and go to the "Pull requests" tab.

- Select the pull request you want to review.

2. Review the Changes:

- Look at the "Files changed" tab to see the diffs.

- Leave comments on specific lines or general feedback.

3. Discuss and Request Changes:

- Engage in discussions with the contributor to clarify any doubts or request changes.

- Use the "Request changes" option if modifications are needed.

4. Approve the Changes:

- Once satisfied with the changes, approve the pull request.

- Click the "Approve" button.

5. Merge the Pull Request:

- If you have merge permissions, click the "Merge pull request" button.

- Choose the merge method (e.g., merge commit, squash, or rebase) and confirm the merge.

6. Close the Branch (optional):

- After merging, delete the branch if it is no longer needed. This can be done directly on GitHub or through the terminal.

git branch -d new-feature

git push origin --delete new-feature

**6.**

**Explain what GitHub Actions are**

Git actions, is a GitHub feature that enables the user to define, automate and execute workflows for software development.

**How can they be used to automate workflows**

Workflows can be also automated with help of GitHub Actions through defining scripts in YAML files located in the repository. These workflows are initiated by events such as code commits, Git pull requests or on a time-based schedule.

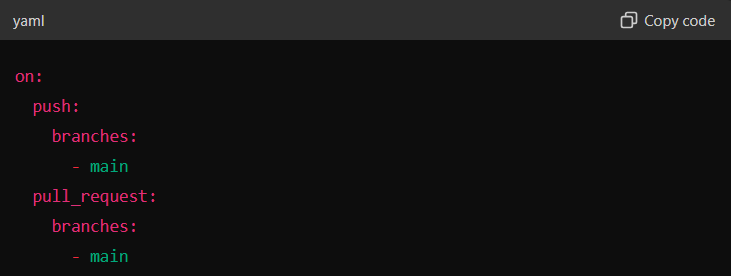
**Example of a simple CI/CD:**

1. Define a Workflow

Create a YAML file in the `. github/workflows` folder.

1. Specify Triggers

Define only events to which this workflow will respond, for instance, `push` or `pull\_request`.



1. Set-up jobs

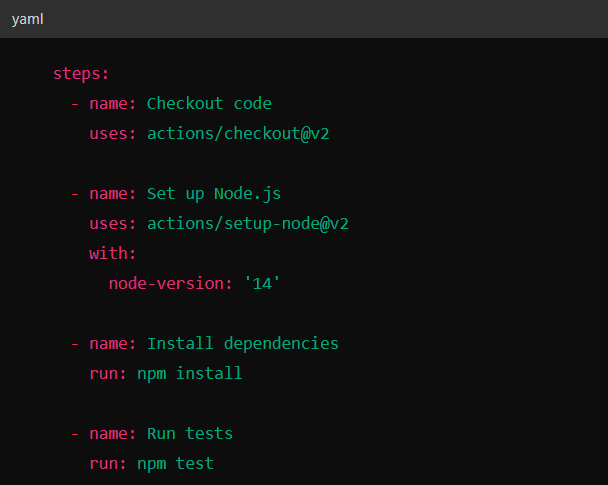
Define jobs with what steps to do the build, test, and deploy of code.

A black and grey screen with green text

Description automatically generated

1. Add steps to jobs

Each job can have multiple steps, such as checking out the code, setting up dependencies, running tests, and deploying



1. Use Predefined Actions

Add actions from community or provided by GitHub to optimize actions used in several projects.

A screen shot of a computer

Description automatically generated

These workflows run automatically, ensuring consistent and efficient project management.

**7.**

**What is Visual Studio?**

Visual Studio is an integrated development environment (IDE) developed by Microsoft

**Key features of Visual Studio**

**Code Editing:** It has an IntelliSense based code editor and swift code completion and syntax editor with supporting color tones for different code.

**Debugging**: It has debugging tools that provides you with the ability to examine the values of the variables and the possibility to place, as well as, remove breakpoints.

**Version Control Integration:** It can be integrated with Git, GitHub, Azure DevOps and other similar systems of version control.

**Extensibility**: It endorses plenty of add-ons and plug-ins to expand the feature sets of the browser.

**Built-in Tools:** It Comprises of Application Development, Application Testing and Application Deployment tools.

**Collaboration:** It contains functions like Live Share, which enables developers to work together in the identical moment.

**Cross-platform Development:** Can be used on Microsoft Windows, macOS, Android, iOS, and as a service.

**The difference between Visual Studio and Visual Studio Code**

Visual Studio is a comprehensive IDE suited **for large-scale projects and enterprise-level projects.**

Visual Studio Code is a free, open source, cross-platform, advanced, integrated and powerful code editor **for the web and smaller projects**, designed for faster and efficient editing.

**8.**

**Describe the steps to integrate a GitHub repository with Visual Studio.**

**Installation of apps and connecting to GitHub**

1. Install Git and Visual Studio.

2. Open Visual Studio and go to "Team Explorer".

3. Click "Manage Connections" and select “Connect to GitHub".

4. Sign in with your GitHub credentials.

**Cloning a GitHub Repository:**

1. In Team Explorer, click "Clone".

2. Enter the repository URL.

3. Choose a local directory.

4. Click "Clone".

**Creating a New GitHub Repository:**

1. In Team Explorer, click "Manage Connections".

2. Click "New"\*\* under "Local Git Repositories".

3. Fill in the "Repository Name", "Local Path", and "Description".

4. Click "Create".

5. Go to "Changes" in Team Explorer.

6. Commit initial changes (if any).

7. Click "Sync”.

8. Click "Publish to GitHub".

**Managing the Repository:**

1. View and stage changes in "Changes".

2. Commit changes with a message.

3. Use "Sync" to pull and fetch changes.

4. Create a new branch in "Branches".

5. Switch branches by double-clicking the branch name.

**How does this integration enhance the development workflow?**

Integrating a GitHub repository with Visual Studio makes the development process more efficient as version control, collaboration and automation can all be accessed through a single environment.

This integration makes it easier for developers to access Git commands within the IDEs such as cloning repositories, commit changes or managing branches.

It contributes to better cooperation through pull requests and the feature called code review, where discussions are held, and changes merged.

Additionally, Visual Studio's integration with GitHub Actions automates tasks like building, testing, and deploying applications, ensuring continuous integration and continuous deployment (CI/CD).

**9.**

**Explain the debugging tools available in Visual Studio**

Visual Studio provides powerful debugging tools, including:

**How can developers use debugging tools to identify and fix issues in their code?**

**- Breakpoints**: Markers that pause code execution at specified lines.

**- Data Tips:** Pop-up windows that display the current values of variables when you hover over them.

**- Watch Windows:** Panels where you can add variables or expressions to monitor their values continuously.

**- Locals and Autos Windows:** Displays local variables and recently used variables within the current scope.

**- Call Stack:** Shows the sequence of function calls that led to the current execution point.

**- Immediate Window:** A console where you can execute commands and evaluate expressions during a debugging session.

**- Output Window:** Displays debugging messages, status updates, and logs.

**- Exception Settings:** Allows configuration of how the debugger handles exceptions, specifying which exceptions to break on.

**- IntelliTrace:** Records execution flow and allows playback to see historical states of the application.

**- Edit and Continue:** Enables modification of code during a debugging session without needing to restart it.

**- Memory and Registers Windows**: Provides views of the program’s memory and CPU registers for low-level inspection.

**- Diagnostic Tools:** Offers performance and memory usage data, helping to identify bottlenecks and leaks.

**- Step Commands:** Controls for navigating code execution:

* + Step Into (F11): Moves into the next function call.
* Step Over (F10): Executes the next line of code without entering functions.
* Step Out (Shift+F11): Continues execution until the current function exits.

**How debugging tools can be used to identify and fix issues in their code**

**- Breakpoints:** Set breakpoints at critical sections of your code to pause execution and inspect the state of the application at specific points. Use conditional breakpoints to pause only when certain conditions are met, making it easier to isolate issues.

**- Data Tips:** Hover over variables while debugging to quickly check their values. This helps in verifying whether variables hold expected values at different execution points.

**- Watch Windows:** Add variables or complex expressions to the Watch window to keep an eye on their values throughout the debugging session. This is particularly useful for tracking how values change over time and identifying where they diverge from expected behavior.

**- Locals and Autos Windows:** Use these windows to automatically view local variables and recently accessed variables. This provides immediate context for the current scope and helps in understanding the current state of the program.

**-Call Stack:** Analyze the call stack to trace the sequence of function calls. This is crucial for understanding the execution flow and identifying the source of errors or unexpected behavior.

**- Immediate Window:** Execute commands and evaluate expressions dynamically to test hypotheses and make quick checks without altering the code. This can help in pinpointing issues interactively.

**- Output Window:** Monitor the Output window for debugging messages and logs. This can give insights into what the application is doing at different stages and help identify errors or performance issues.

**- Exception Settings:** Configure exception settings to break on specific exceptions, allowing you to catch and diagnose errors early. This helps in identifying issues that may not be immediately apparent.

**- IntelliTrace:** Use IntelliTrace to record the execution flow and replay it to see the historical state of your application. This is useful for diagnosing issues that are hard to reproduce, as you can see the exact sequence of events leading up to the problem.

**- Edit and Continue:** Modify your code during a debugging session and continue without restarting. This allows you to quickly test fixes and see their effects in real-time.

**- Memory and Registers Windows:** Inspect memory contents and CPU registers to debug low-level issues such as memory corruption or incorrect register values. This is essential for performance tuning and fixing low-level bugs.

**- Diagnostic Tools:** Use diagnostic tools to monitor performance metrics and memory usage. This helps in identifying performance bottlenecks, memory leaks, and other resource-related issues.

**- Step Commands:** Navigate through your code using Step Into, Step Over, and Step Out commands to control the flow of execution. This allows you to follow the program’s logic closely and identify the exact point where issues occur.

**10.**

**Discuss how GitHub and Visual Studio can be used together to support collaborative development.**

**- Version Control:** GitHub provides a central repository for version control. Developers can clone repositories, create branches, commit changes, and push updates directly from Visual Studio.

**- Pull Requests:** Developers can create pull requests in Visual Studio to propose changes to the codebase. This facilitates code reviews and discussions before merging changes.

**- Issue Tracking:** GitHub’s issue tracker can be used within Visual Studio to manage tasks, bug reports, and feature requests, keeping the team organized and focused.

**- Continuous Integration and Deployment (CI/CD**): GitHub Actions can be configured to automate testing and deployment workflows. Visual Studio’s integration with GitHub Actions allows for setting up and managing these workflows directly from the IDE.

**- Live Share:** Visual Studio’s Live Share feature enables real-time collaboration. Developers can share their coding session with teammates, allowing for pair programming and instant feedback.

**- Documentation and Wikis:** GitHub’s wiki and markdown files for documentation can be edited directly in Visual Studio, ensuring that project documentation stays up to date.

**Provide a real-world example of a project that benefits from this integration.**

Real-World Example: Open Source Software Project

**Project: "EduLearn"** - An Open Source E-Learning Platform

Overview: EduLearn is an open-source e-learning platform designed to provide educational resources and online courses to users. The project involves multiple developers working on different aspects such as user interface design, backend development, content creation, and more.

**Collaborative Workflow:**

1. Initial Setup:

- The project lead creates a GitHub repository for EduLearn.

- Developers clone the repository using Visual Studio to get the initial codebase on their local machines.

2. Branching Strategy:

- Developers create branches for new features or bug fixes. For example, a branch named `feature/login-system` for implementing the user login functionality.

- This branching strategy ensures that the main branch remains stable and that new features are developed and tested in isolation.

3. Development and Commit:

- Each developer works on their assigned tasks within their branches, using Visual Studio to write and debug code.

- Regular commits are made with descriptive messages, such as `Added login form validation`.

4. Pull Requests and Code Reviews:

- Once a feature is complete, a developer creates a pull request from their feature branch to the main branch on GitHub.

- Team members review the pull request, provide feedback, and discuss any necessary changes.

- Changes are made in Visual Studio, and the pull request is updated until it meets the team’s standards.

5. Continuous Integration:

- GitHub Actions are set up to run automated tests on each pull request. This ensures that new changes do not break existing functionality.

- Visual Studio’s integration with GitHub Actions allows developers to view test results and debug issues directly within the IDE.

6. Merging and Deployment:

- After a pull request is approved, it is merged into the main branch.

- A GitHub Action is triggered to deploy the latest version of EduLearn to a staging environment for further testing.

- Once validated, the changes are deployed to production.

7. Live Collaboration:

- Developers use Visual Studio Live Share for pair programming sessions to tackle complex issues or onboard new team members.

- This real-time collaboration speeds up problem-solving and knowledge sharing.

8. Issue Management:

- Bugs, new features, and improvements are tracked using GitHub Issues. Each issue is assigned to a developer and linked to corresponding pull requests.

- Visual Studio’s integration with GitHub allows developers to view and update issues without leaving the IDE.

9. Documentation:

- Project documentation is maintained in the repository’s wiki and markdown files.

- Developers update documentation as they implement new features, ensuring that users and contributors have up-to-date information.

**Benefits**

**- Efficiency:** Streamlined development and collaboration processes reduce bottlenecks and enhance productivity.

**- Quality:** Automated testing and code reviews help maintain high code quality and catch issues early.

**- Transparency:** Clear version history and issue tracking provide transparency and accountability within the team.

**- Real-time Collaboration:** Live Share facilitates real-time coding and problem-solving, enhancing teamwork and communication.

By leveraging the integration between GitHub and Visual Studio, the EduLearn project benefits from a cohesive development environment that supports efficient collaboration, high code quality, and streamlined workflows.

*Source:* ChatGPT