Mastering Visual Studio: Coding and Git Integration

In the ever-evolving software development landscape, proficiency in seamlessly integrating coding environments and version control systems is essential. This guide is meticulously crafted to empower developers with the knowledge and skills needed to optimize Visual Studio (VS) and Git for a productive development workflow. It is structured into four core subtasks, each catering to a specific facet of the development workflow:

I) Install and Initialize Visual Studio

II) Write, Edit, and Compile Code

III) Integrate Git with Visual Studio

IV) Review and Merge Code

These subtasks are elaborated below.

I) Install and Initialize Visual Studio

This section lists the foundational steps of setting up Visual Studio on your development machine.

a) Download and Install Visual Studio

1. Visit the official website: https://visualstudio.microsoft.com/downloads/ and download Visual Studio Code.
2. Choose the appropriate version of VS for your Operating System (Windows, Linux or MacOS as shown in *Figure 1*).
3. Follow the on-screen instructions to complete the installation process.

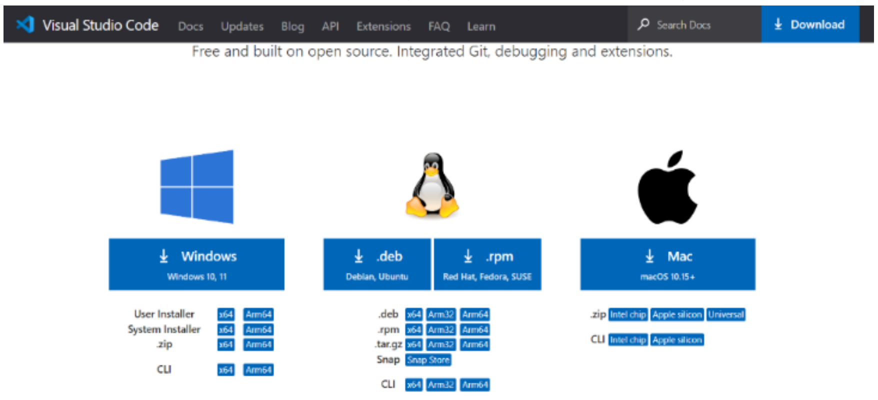
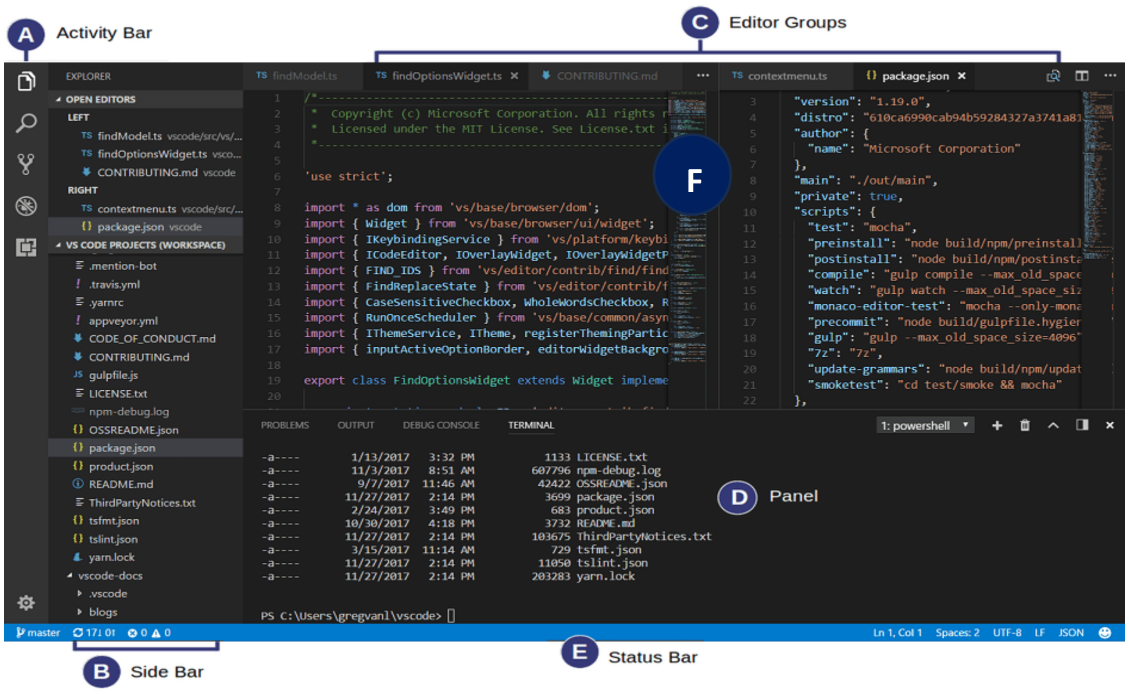


Figure 1

b) Explore the VS Interface



```
/*---------------------------------------------------------------
 *  Copyright (c) Microsoft Corporation. All rights r
 *  Licensed under the MIT License. See License.txt i
 *---------------------------------------------------------------

'use strict';

import * as dom from 'vs/base/browser/dom';
import { Widget } from 'vs/base/browser/ui/widget';
import { IKeybindingService } from 'vs/platform/keybi
import { ICodeEditor, IOverlayWidget, IOverlayWidgetP
import { FIND_IDS } from 'vs/editor/contrib/find/find
import { FindReplaceState } from 'vs/editor/contrib/f
import { CaseSensitiveCheckbox, WholeWordsCheckbox, R
import { RunOnceScheduler } from 'vs/base/common/asyn
import { IThemeService, ITheme, registerThemingPartic
import { inputActiveOptionBorder, editorWidgetBackgro

export class FindOptionsWidget extends Widget impleme
```

```
"version": "1.19.0",
"distro": "610ca6990cab94b59284327a3741a81
"author": {
  "name": "Microsoft Corporation"
},
"main": "./out/main",
"private": true,
"scripts": {
  "test": "mocha",
  "preinstall": "node build/npm/preinstall
  "postinstall": "node build/npm/postinsta
  "compile": "gulp compile --max_old_space
  "watch": "gulp watch --max_old_space_siz
  "monaco-editor-test": "mocha --only-mona
  "precommit": "node build/gulpfile.hygier
  "gulp": "gulp --max_old_space_size=4096"
  "7z": "7z",
  "update-grammars": "node build/npm/updat
  "smoketest": "cd test/smoke && mocha"
},
```

```
-a----        1/13/2017   3:32 PM           1133 LICENSE.txt
-a----        11/3/2017   8:51 AM         607796 npm-debug.log
-a----         9/7/2017  11:46 AM          42422 OSSREADME.json
-a----       11/27/2017   2:14 PM           3699 package.json
-a----        2/24/2017   3:49 PM            683 product.json
-a----       10/30/2017   4:18 PM           3732 README.md
-a----       11/27/2017   2:14 PM         103675 ThirdPartyNotices.txt
-a----        3/15/2017  11:14 AM            729 tsfmt.json
-a----       11/27/2017   2:14 PM          11050 tslint.json
-a----       11/27/2017   2:14 PM         203283 yarn.lock

PS C:\Users\gregvanl\vscode> 
```

Figure 2

4. After the installation, launch Visual Studio and explore its interface (*Figure 2*)
5. Check the folder structure (B) where your projects are organized.
6. Locate the Code Editor (C) and the terminal (D), which is required for writing code and executing commands respectively.
7. Identify the various panes within the Code Editor, such as the Minimap (F) and the Output Pane (D- same as the terminal)

c) Create a Project

8. Use the *'Create a new project'* feature to start a new development project.
9. Choose a project template based on the programming language and type (Example: *ng-template* for Angular)
10. Follow the prompts to set up project details like name, location and solution configuration.

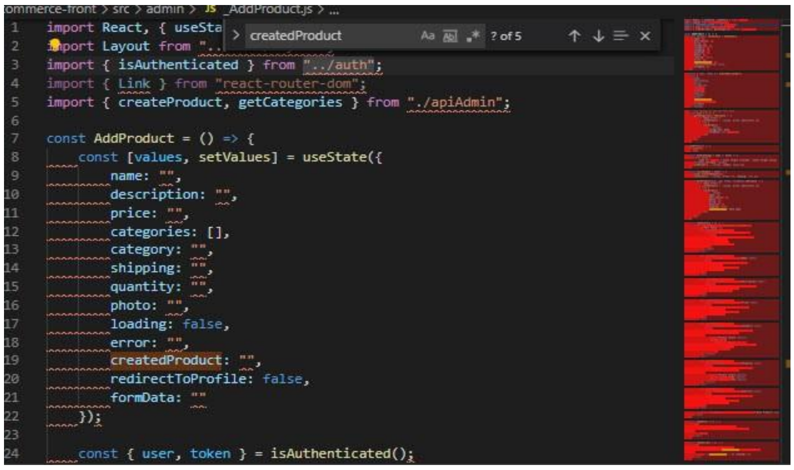
II) Write, Edit, and Compile Code

This section emphasizes the importance of efficient coding practices and syntax verification, ensuring that your code is not just functional, but also optimized.

a) Learn Keyboard Shortcuts and Use Code Helper Tools

11. Refer to *Appendix A* for a comprehensive list of keyboard shortcuts to navigate and optimize your workflow.

12. Utilize the built-in Code Helper tool, *IntelliSense* for auto-completion and error highlighting.

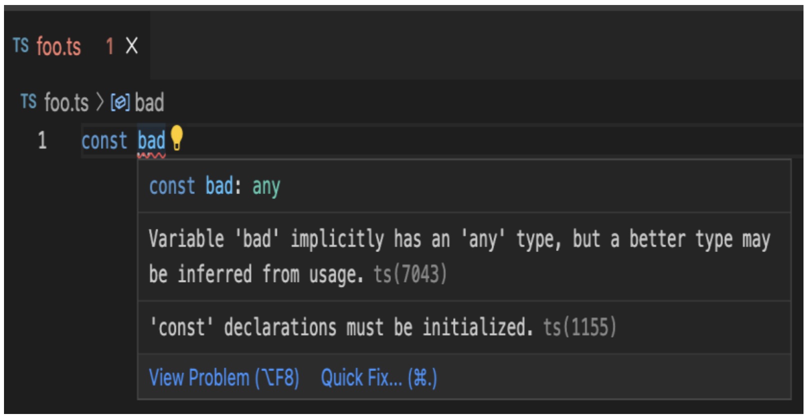


```
import React, { useSta
import Layout from "..
import { isAuthenticated } from "../auth";
import { Link } from "react-router-dom";
import { createProduct, getCategories } from "./apiAdmin";

const AddProduct = () => {
    const [values, setValues] = useState({
        name: "",
        description: "",
        price: "",
        categories: [],
        category: "",
        shipping: "",
        quantity: "",
        photo: "",
        loading: false,
        error: "",
        createdProduct: "",
        redirectToProfile: false,
        formData: ""
    });

    const { user, token } = isAuthenticated();
```

Figure 3

Red squiggly lines *(Figure 3)* highlight errors related to syntax or code compilation.

The Minimap on the right end *(Figure 3)* displays in red, parts of code that contain errors.

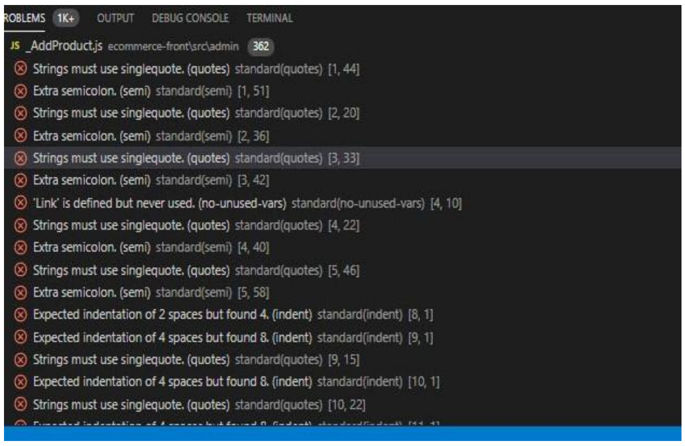


```
const bad
```

```
const bad: any

Variable 'bad' implicitly has an 'any' type, but a better type may
be inferred from usage. ts(7043)

'const' declarations must be initialized. ts(1155)
```

Figure 4

When hovered, Visual Studio displays a message on why the code highlighted in red is an error and suggests ways using which it can be fixed *(Figure 4)*



```
JS _AddProduct.js ecommerce-front\src\admin 362
Strings must use singlequote. (quotes) standard(quotes) [1, 44]
Extra semicolon. (semi) standard(semi) [1, 51]
Strings must use singlequote. (quotes) standard(quotes) [2, 20]
Extra semicolon. (semi) standard(semi) [2, 36]
Strings must use singlequote. (quotes) standard(quotes) [3, 33]
Extra semicolon. (semi) standard(semi) [3, 42]
'Link' is defined but never used. (no-unused-vars) standard(no-unused-vars) [4, 10]
Strings must use singlequote. (quotes) standard(quotes) [4, 22]
Extra semicolon. (semi) standard(semi) [4, 40]
Strings must use singlequote. (quotes) standard(quotes) [5, 46]
Extra semicolon. (semi) standard(semi) [5, 58]
Expected indentation of 2 spaces but found 4. (indent) standard(indent) [8, 1]
Expected indentation of 4 spaces but found 8. (indent) standard(indent) [9, 1]
Strings must use singlequote. (quotes) standard(quotes) [9, 15]
Expected indentation of 4 spaces but found 8. (indent) standard(indent) [10, 1]
Strings must use singlequote. (quotes) standard(quotes) [10, 22]
```

Figure 5

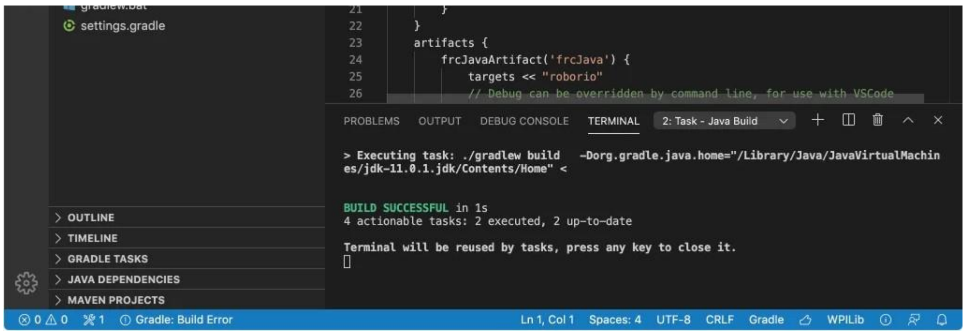
Output window *(Figure 5)* shows the compilation errors that need to be corrected before the code can be pushed to the remote branch.

b) Verify Code Syntax and Functionality

13. Review your code for syntax compliance and expected functionality.
14. Address any issues identified or apply suggestions offered by VS during code review *(refer to Figure 4)*

c) Compile Code

15. Execute the compilation command relevant to your project type (Example: *ng build* for Angular projects)
16. Monitor the output window for any compilation errors or warnings *(see Figure 5)*
17. Confirm that the compilation is successful. You should see the **Build Successful** message in the terminal as seen below in *Figure 6*.



```
    }
artifacts {
    frcJavaArtifact('frcJava') {
        targets << "roborio"
        // Debug can be overridden by command line, for use with VSCode
```

```
> Executing task: ./gradlew build   -Dorg.gradle.java.home="/Library/Java/JavaVirtualMachin
es/jdk-11.0.1.jdk/Contents/Home" <

BUILD SUCCESSFUL in 1s
4 actionable tasks: 2 executed, 2 up-to-date

Terminal will be reused by tasks, press any key to close it.
```

Figure 6

III) Integrate Git with Visual Studio

In this segment, learn to seamlessly configure Git settings, authenticate and compile through basic Git commands.

a) Configure Git Settings

18. Access Visual Studio's Git settings to configure authentication using your credentials- *Username* and *Password* and set up proxy servers *(localhost:8080)*
19. Restart Visual Studio to apply the configured Git settings.
20. When initializing a new project, select the appropriate branch type and adhere to the naming conventions.

- feature/<feature-name>: For adding a new feature or functionality. *Example: feature/user-authentication*
- bugfix/<issue-number>-<short-description>: For fixing defects or bugs. *Example: bugfix/123-fix-login-issue*
- hotfix/<version-number>-<short-description>: For addressing critical issues in production. *Example: hotfix/1.2.1-security-patch*
- release/<version-number>: For preparing a release. *Example: release/1.3.0*
- development or main or master: For ongoing development or the main branch. *Example: main*

- task/<task-name>: For smaller tasks or miscellaneous work. *Example: task/update-readme*

b) Learn Git Commands

21. Refer to the list of basic Git commands provided in *Appendix B - Part1*.
22. Implement essential commands such as git clone, git pull, and git push while syncing the local repository with the master branch.

c) Fix Issues During Git Integration

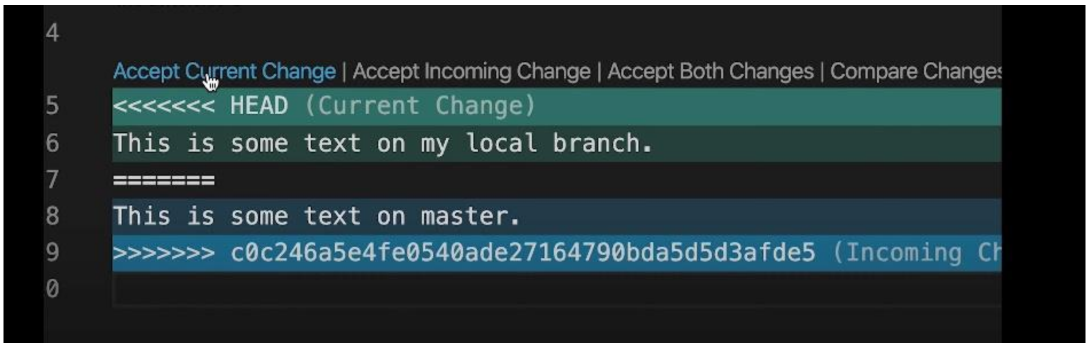
23. Consult *Appendix B - Part 2* for a list of common issues that may arise during Git integration.
24. Follow the suggested solutions to troubleshoot and resolve integration problems.

d) Commit code changes

25. Commit the code changes made with clear and descriptive commit messages.
    Example Commit Message: DML-7090:DML-913 UI Updates on Login Screen
    DML-7090: Story/Task number
    DML-913: Sub-task number
    UI Updates on Login Screen: Descriptive message that sums up why the code changes in that specific commit were made, to let reviewers make informed decisions on Pull Request approvals
26. Use commands like *git add .* to stage changes and *git commit -m 'Your commit message'* to commit them.

e) Handle code conflicts during code push

27. When pushing code changes, be prepared to resolve conflicts if any.
28. If conflicts exist, utilize Git features like Accept Current Change, Accept Incoming Change, Compare Changes, and Accept Both Changes accordingly.


Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
<<<<<<< HEAD (Current Change)
This is some text on my local branch.
=======
This is some text on master.
>>>>>>> c0c246a5e4fe0540ade27164790bda5d5d3afde5 (Incoming Ch
```

Figure 7

In *Figure 7*,

- **Accept Current Change:** Choose this option to keep the changes made in the current branch.
- **Accept Incoming Change:** Choose this option to keep the changes from the incoming branch.
- **Compare Changes:** Visually compare conflicting changes to make informed decisions.
- **Accept Both Changes:** Choose this option to combine and accept changes from both branches.

f) Switch Branches for Different Tasks

29. Use the *git checkout* command to switch between branches based on the tasks you are working on. The task type can be one of the following:
    - Isolating Work
    - Parallel Development
    - Testing and Experimentation
    - Versioning and Release
    - Collaboration
    - Bug Fix
    - Code Reviews
30. Ensure a smooth transition between different branches in your version control workflow.

IV) Review and Merge Code

This final subsection delves into the critical aspects of code review and collaboration.

a) Create a Pull Request

31. Draft a Pull Request (PR) through Git with a clear title and description *(see Figure 8)*
32. Specify the feature branch and the target branch for the merge.
33. Attach any relevant documentation or files to assist reviewers.
34. Add tags to the PR that identify it to be a part of a specific change *(Example Tag: Jan 2024 UI revamp)*

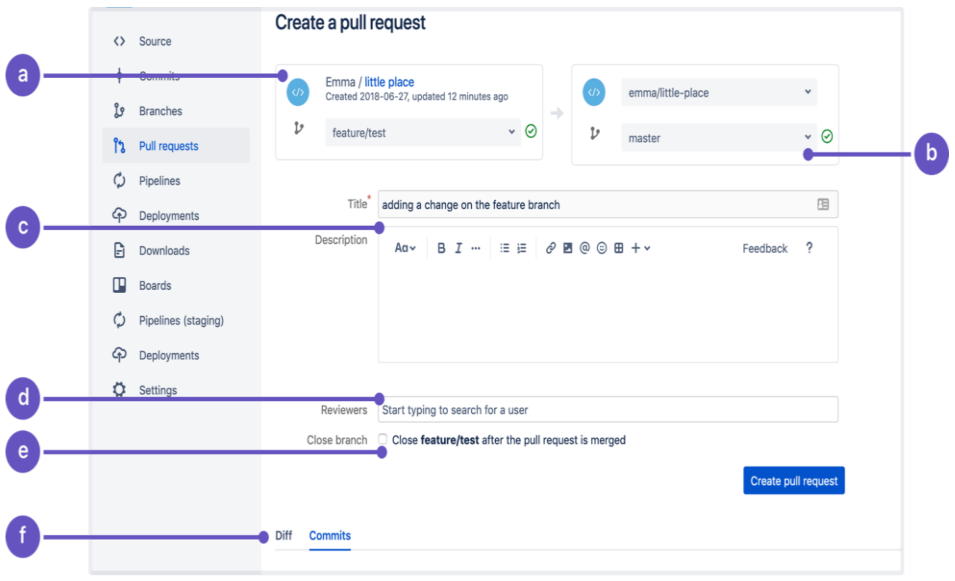


Figure 8

a) Source branch
b) Destination branch
c) Description of the code changes in the Pull Request
d) Reviewers List
e) Checkbox to delete the branch after merge
f) Pane to check the commits and differences in code from previous versions

b) Seek Peer Approval

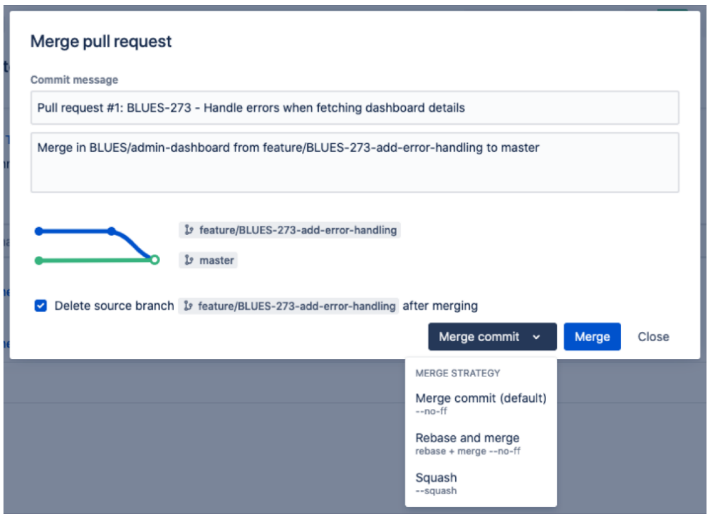
35. Add appropriate reviewers to your Pull Request.
36. Understand the different response types: Approved, Unapproved, Needs Work, Declined, Re-Open (*refer to Table 1*)
37. Act on the reviewer's feedback to enhance code quality.

Table 1: Pull Request Review Responses

Approved	Proceed with the merging process as all changes are approved.
Unapproved	Address the reviewer's concerns and modify the code accordingly before seeking another review.
Needs work	Make necessary improvements based on feedback and resubmit the pull request for review.
Declined	Reassess the feedback, make substantial changes, and resubmit the pull request for a fresh review.
Re-open	Address specific concerns or requested changes to move forward with the review process again.

c) Merge Code

38. Fulfill any specified merge conditions.
39. Confirm the feature and target branch names. When there are no conflicts, click on the Merge button *(as seen in Figure 9)* to merge the Pull Request.



Merge pull request

Commit message

Pull request #1: BLUES-273 - Handle errors when fetching dashboard details

Merge in BLUES/admin-dashboard from feature/BLUES-273-add-error-handling to master

feature/BLUES-273-add-error-handling

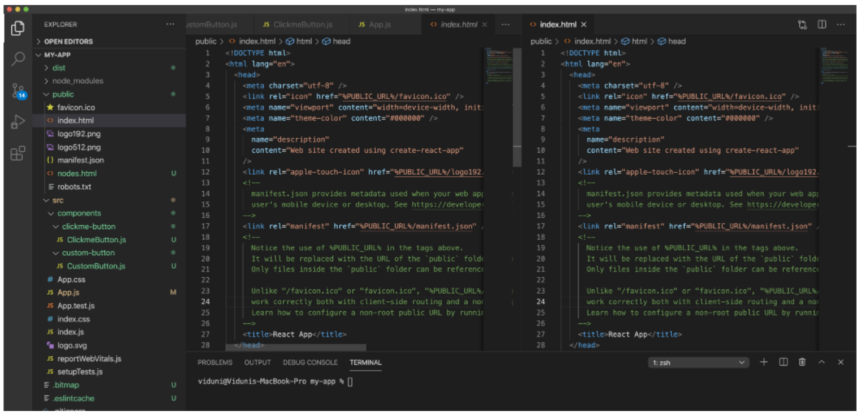
master

☑ Delete source branch feature/BLUES-273-add-error-handling after merging

Figure 9

d) Inspect Code Changes

40. Use the built-in comparison features, like Code Split *(Figure 10)* to thoroughly check and compare the code changes made.
41. Examine the refactored code.



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, init
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.
    <!--
      manifest.json provides metadata used when your web app
      user's mobile device or desktop. See https://develope
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" /
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` fold
      Only files inside the `public` folder can be referenc

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/
      work correctly both with client-side routing and a non
      Learn how to configure a non-root public URL by runnin
    -->
    <title>React App</title>
  </head>
```

```
viduni@Vidunis-MacBook-Pro my-app % []
```

Figure 10

e) Understand Code Revision and Git Version Control

42. Grasp the fundamentals of code revision and Git's version control capabilities.

- **Versioning:**
  Git allows developers to track changes to their code over time. Each modification, or "commit," is recorded, providing a complete history of the project's evolution.

- **Distributed System:**
  Git is a distributed version control system, meaning that every developer working on a project has a complete copy of the repository on their local machine. This decentralization enables offline work and faster access to project history.

- **Branching and Merging:**
  Branching in Git allows developers to work on different features or fixes simultaneously without interfering with each other.
  Merging combines these separate branches back into the main codebase. This feature encourages parallel development and facilitates collaboration.

43. Recognize the importance of version history in tracking changes and collaborating with a development team.
    - Use tools like *git log* to review commits. Version history aids in identifying when and by whom specific changes were made. A well-maintained version history is crucial for effective collaboration, debugging, and maintaining code integrity.

References

[1] Graphic/Image. MergePR. https://confluence.atlassian.com/bitbucketserver/merge-a-pull-request-808488562.html, Accessed January 28, 2024

[2] Graphic/Image.PR. https://stackoverflow.com/questions/26678718/issue-with-bitbucket-pull-request , Accessed January 28, 2024

[3] Graphic/Image.Merge Conflict. https://stackoverflow.com/questions/73277690/how-to-change-vs-codes-merge-conflict-layout , Accessed Januar 28, 2024

[4] Graphic/Image.Build Success. https://www.codelikethewind.org/2021/10/13/codeready-workspaces-consistent-development-environments/ ,Accessed January 28, 2024

[5] Graphic/Image.VS Error. https://stackoverflow.com/questions/63419399/red-squiggly-lines-in-vs-code ,Accessed January 28, 2024

[6] Microsoft. VS Tips and Tricks. https://code.visualstudio.com/docs/getstarted/tips-and-tricks , Accessed January 28, 2024

Appendix A

Keyboard Shortcuts for Visual Studio

General Navigation:

- Go to Definition: F12
- Navigate Backward: Ctrl + -
- Navigate Forward: Ctrl + Shift + -
- Quick Find: Ctrl + F
- Quick Replace: Ctrl + H

Code Editing:

- Cut Line: Ctrl + X
- Copy Line: Ctrl + C
- Paste Line: Ctrl + V
- Comment/Uncomment: Ctrl + K, Ctrl + C/U
- Duplicate Line: Ctrl + D

Code Navigation:

- Navigate to Solution Explorer: Ctrl + Alt + L
- Find All References: Shift + F12
- Navigate to Previous/Next Member: Alt + Up/Down

Code Selection:

- Select Current Line: Ctrl + L
- Expand/Shrink Selection: Alt + Shift + .

Code Formatting:

- Format Document: Ctrl + K, Ctrl + D
- Format Selection: Ctrl + K, Ctrl + F

Build and Debug:

- Build Solution: Ctrl + Shift + B
- Start Debugging: F5
- Stop Debugging: Shift + F5
- Set Next Statement: Ctrl + Shift + F10

Version Control (Git):

- View Changes: Ctrl + 0, G
- Commit Changes: Ctrl + 0, C
- Push Changes: Ctrl + 0, P
- Pull Changes: Ctrl + 0, F

Refactoring:

- Rename: Ctrl + R, Ctrl + R
- Extract Method: Ctrl + R, Ctrl + M
- Organize Usings: Ctrl + R, Ctrl + G

Miscellaneous:

Open Command Prompt: `Ctrl + ``

Show/Hide Solution Explorer: Ctrl + Alt + L

Show/Hide Toolbox: Ctrl + Alt + X

Remember to explore and customize these shortcuts based on your workflow preferences. You can find and modify keyboard shortcuts in Visual Studio through the ‘Options’ menu under "Environment" -> "Keyboard."

Appendix B

Part 1: List of Git Commands for Visual Studio and Git Integration

Clone Repository:

git clone <repository_url>: Clone a remote repository to your local machine.

Configure Git:

git config --global user.name "Your Name": Set your global username.

git config --global user.email "your.email@example.com": Set your global email.

Branching:

git branch: List all branches in the repository.

git branch <branch_name>: Create a new branch.

git checkout <branch_name> or *git switch <branch_name>*: Switch to a specific branch.

git merge <branch_name>: Merge changes from one branch to another.

Status and Changes:

git status: View the status of your working directory and staging area.

git add <file>: Add changes to the staging area.

git commit -m "Your commit message": Commit changes with a descriptive message.

Pull and Push:

git pull origin <branch_name>: Fetch changes from a remote repository and merge them into the current branch.

git push origin <branch_name>: Push local changes to a remote repository.

Remote Repository:

git remote -v: List remote repositories.

git remote add <remote_name> <repository_url>: Add a new remote repository.

Log and History:

git log: Display the commit history.

git log --oneline: Display a concise commit history.

Undoing Changes:

git reset HEAD <file>: Unstage changes.

git checkout -- <file>: Discard changes in the working directory.

Tagging:

git tag -a <tag_name> -m "Tag message": Create an annotated tag.

Stashing:

git stash: Save changes that are not ready to be committed.

git stash apply: Apply stashed changes to your working directory.

Remember to replace *<branch_name>, <file>, <repository_url>*, and *<tag_name>* with the actual branch name, file, repository URL, and tag name, respectively. These commands provide a foundation for managing your Git workflow within Visual Studio or any Git-integrated development environment.

Part 2: Git Integration – Common Issues and Fixes

Authentication Issues:

Issue: Unable to authenticate with the remote repository.

Fix: Ensure that your Git credentials are correctly configured. Use the git config command to set your username and email. If using HTTPS, ensure that your credentials are saved or use SSH for authentication.

Proxy Configuration:

Issue: Working behind a firewall or proxy, leading to connection issues.

Fix: Set up proxy configuration in your Git settings using commands like git config --global http.proxy and git config --global https.proxy. Ensure the proxy allows Git traffic.

Branch Naming Conflicts:

Issue: Naming conflicts when creating or switching branches.

Fix: Follow a consistent branch naming convention to avoid conflicts. If switching branches, commit or stash changes in the current branch before switching.

Outdated Remote Tracking Branches:

Issue: Local branches not reflecting changes made in the remote repository.

Fix: Use git fetch to update your local repository with changes from the remote. Optionally, use git pull to fetch and merge changes in one step.

Merge Conflicts:

Issue: Conflicts arise when merging branches.

Fix: Manually resolve conflicts by editing conflicted files. Use git status, git diff, and git mergetool to identify and resolve conflicts. Commit changes after resolving.

Uncommitted Changes During Branch Switching:

Issue: Unable to switch branches due to uncommitted changes.

Fix: Commit or stash changes before switching branches. Use git stash to temporarily save changes if not ready for a commit.

Large Files and Git LFS:

Issue: Large files causing repository bloat and slow performance.

Fix: Use Git Large File Storage (LFS) for managing large files. Install Git LFS, track large files, and commit them to LFS.

Incomplete Git Installation:

Issue: Git commands not recognized or working.

Fix: Verify that Git is installed correctly on your machine. Ensure that the Git executable is in your system's PATH.

Repository Corruption:

Issue: Corruption or inconsistency in the Git repository.

Fix: In some cases, cloning the repository again may be necessary. Regularly backup your repository to prevent data loss.

SSH Key Issues:

Issue: SSH key not properly configured for remote access.

Fix: Ensure that your SSH key is added to your SSH agent (ssh-add) and associated with your Git account. Update your remote repository URL to use the SSH protocol.

Addressing these common Git integration issues and following the suggested fixes will help ensure a smoother workflow with Git in Visual Studio or any other Git-integrated environment.