TEAM DATA - DELIVERABLE 3

20.03.2020

—

DHARMIK SHAH
YUN JIE (JEFFREY) LI
JEFFREY SO
ALVIN TANG
JIMMY PANG

Table of Contents

Promising Issues

Allow Selection of Multiple Editor Tabs

Github issue #89958: Allow Selection of Multiple Editor Tabs

This ticket was created as a feature request on February 3. The user requested to allow the selection of multiple tabs within an editor group, similar to the process of that in Google Chrome. There were around 10 upvotes for the ticket, but because of the lack of work on it, it has since been a candidate for the backlog.

We did not choose this feature for deliverable 2 because we had considered it to be too advanced for that deliverable, so we instead chose to work on other tickets reasonable at the time. We now consider working on this ticket for this deliverable, given that it provides us with an opportunity to further dive into VS Code's codebase and better understand how multiple components work together. The tasks in the previous deliverable only required us to mostly work in one major component, which is the editor.

Modification Explanation

The following is a list of files that we will most likely work with:

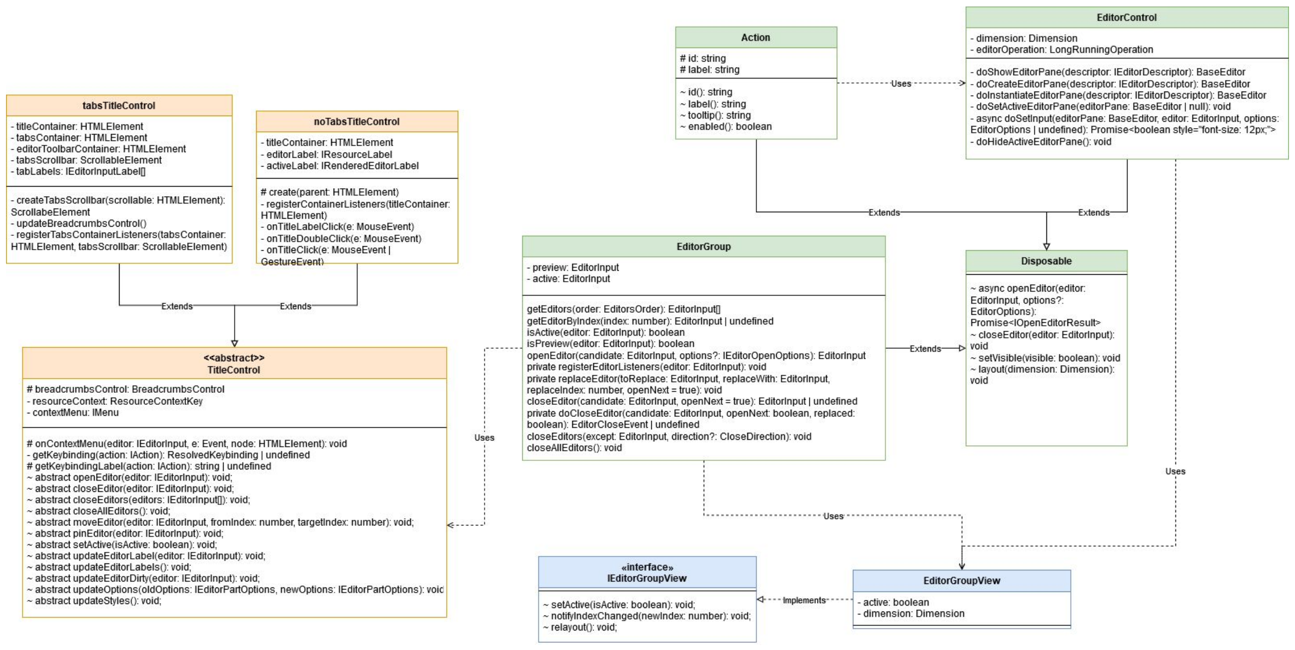
- **`workbench\common\editor\editorGroup.ts`**
- **`workbench\browser\parts\editor\editorActions.ts`**
- **`workbench\browser\parts\editor\editorControl.ts`**
- **`workbench\browser\parts\editor\editorGroupView.ts`**
- **`workbench\browser\parts\editor\tabsTitleControl.ts`**

We will need to go into where the tabs are being drawn. Most likely the editors/workspace component in VS Code, and find where in the editor is the tab drag being handled. From there, we would need to add the ability for multiple tabs to be selected. For this, we need to listen to a modifier key e.g. shift (this will likely also be done within the editor component), and apply the action that a single tab does. As well as allowing some context menu options to be done on multiple files. This will likely all be within the editor and workspace component and maybe some common components within the base folder.

The **`tabTitleControl.ts`** file under **`workbench/browser/parts/editor`** manages the behaviour of tabs when interacted with by the user. When you drag a tab and move it to a different tab group, in the file **`tabsTitleControl.ts`**, the function **`createTab()`** is called to create the tab in the new group, and then **`openEditor()`** is invoked to set focus to that tab. The function **`closeEditor()`** is then invoked to close the tab in the old group. This gives the impression that the user moves a tab from one group to another.

The **`noTabsTitleControl.ts`** file is responsible for the behaviour when tabs are disabled by the user in VS Code (i.e. **`workbench.editor.showTabs`** is set to **`false`**). There is an option to modify this file, should we choose to expand the feature to close multiple selected tabs at once.

UML Diagram


Action
# id: string
# label: string
~ id(): string
~ label(): string
~ tooltip(): string
~ enabled(): boolean
Uses
EditorControl
- dimension: Dimension
- editorOperation: LongRunningOperation
- doShowEditorPane(descriptor: IEditorDescriptor): BaseEditor
- doCreateEditorPane(descriptor: IEditorDescriptor): BaseEditor
- doInstantiateEditorPane(descriptor: IEditorDescriptor): BaseEditor
- doSetActiveEditorPane(editorPane: BaseEditor | null): void
- async doSetInput(editorPane: BaseEditor, editor: EditorInput, options: EditorOptions | undefined): Promise<boolean style="font-size: 12px;">
- doHideActiveEditorPane(): void
tabsTitleControl
- titleContainer: HTMLElement
- tabsContainer: HTMLElement
- editorToolbarContainer: HTMLElement
- tabsScrollbar: ScrollableElement
- tabLabels: IEditorInputLabel[]
- createTabsScrollbar(scrollable: HTMLElement): ScrollabeElement
- updateBreadcrumbsControl()
- registerTabsContainerListeners(tabsContainer: HTMLElement, tabsScrollbar: ScrollableElement)
noTabsTitleControl
- titleContainer: HTMLElement
- editorLabel: IResourceLabel
- activeLabel: IRenderedEditorLabel
# create(parent: HTMLElement)
- registerContainerListeners(titleContainer: HTMLElement)
- onTitleLabelClick(e: MouseEvent)
- onTitleDoubleClick(e: MouseEvent)
- onTitleClick(e: MouseEvent | GestureEvent)
Extends
Extends
Extends
Extends
EditorGroup
- preview: EditorInput
- active: EditorInput
getEditors(order: EditorsOrder): EditorInput[]
getEditorByIndex(index: number): EditorInput | undefined
isActive(editor: EditorInput): boolean
isPreview(editor: EditorInput): boolean
openEditor(candidate: EditorInput, options?: IEditorOpenOptions): EditorInput
private registerEditorListeners(editor: EditorInput): void
private replaceEditor(toReplace: EditorInput, replaceWith: EditorInput, replaceIndex: number, openNext = true): void
closeEditor(candidate: EditorInput, openNext = true): EditorInput | undefined
private doCloseEditor(candidate: EditorInput, openNext: boolean, replaced: boolean): EditorCloseEvent | undefined
closeEditors(except: EditorInput, direction?: CloseDirection): void
closeAllEditors(): void
Disposable
~ async openEditor(editor: EditorInput, options?: EditorOptions): Promise<IOpenEditorResult>
~ closeEditor(editor: EditorInput): void
~ setVisible(visible: boolean): void
~ layout(dimension: Dimension): void
Extends
<<abstract>>
TitleControl
# breadcrumbsControl: BreadcrumbsControl
- resourceContext: ResourceContextKey
- contextMenu: IMenu
# onContextMenu(editor: IEditorInput, e: Event, node: HTMLElement): void
- getKeybinding(action: IAction): ResolvedKeybinding | undefined
# getKeybindingLabel(action: IAction): string | undefined
~ abstract openEditor(editor: IEditorInput): void;
~ abstract closeEditor(editor: IEditorInput): void;
~ abstract closeEditors(editors: IEditorInput[]): void;
~ abstract closeAllEditors(): void;
~ abstract moveEditor(editor: IEditorInput, fromIndex: number, targetIndex: number): void;
~ abstract pinEditor(editor: IEditorInput): void;
~ abstract setActive(isActive: boolean): void;
~ abstract updateEditorLabel(editor: IEditorInput): void;
~ abstract updateEditorLabels(): void;
~ abstract updateEditorDirty(editor: IEditorInput): void;
~ abstract updateOptions(oldOptions: IEditorPartOptions, newOptions: IEditorPartOptions): void
~ abstract updateStyles(): void;
Uses
Uses
Uses
«interface»
IEditorGroupView
~ setActive(isActive: boolean): void;
~ notifyIndexChanged(newIndex: number): void;
~ relayout(): void;
Implements
EditorGroupView
- active: boolean
- dimension: Dimension

Tabs for Integrated Terminal

Github issue #10546: Tabs for integrated terminal

In August 2016, a feature request to add tabs for Visual Studio Code's integrated terminal was raised. These tabs would replace a dropdown menu for selecting which terminal is actively being viewed by the user. The comments proceeding this request suggest that the feature requested was considered beforehand by the main contributors at Microsoft. However, it is still an open issue as those contributors decided that it would disregard the "light-weight" perceptions of the UI. This is a valid point as the users of Visual Studio Code range from beginners to advanced, and adding such a feature could make it feel harder to use for beginners.

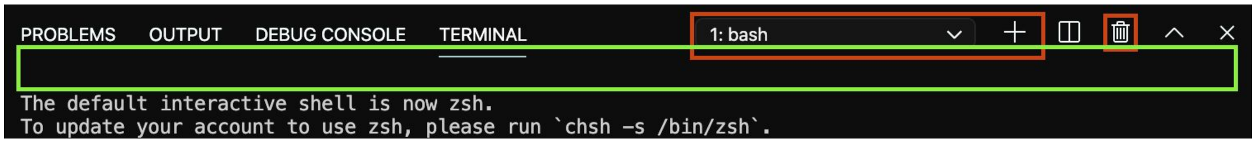
However, with that being said, the issue is still open and actively being discussed. With over 2000 reactions and 170 comments, it is one of the most longest-standing popular feature requests out there. It may also be interesting to implement as it allows us to play with the UI, which we have never directly touched before. It also allows us to build something impactful that a variety of developers hope to use, and may also increase their efficiency in the process. This issue, however, potentially could take over a few days to complete and may actually be too hard to implement in the timeframe we have (especially with COVID-19).

Modification Explanation

After taking a careful look into the architecture of how terminals are designed, it may be very difficult to modify this as it involves thousands of lines of the codebase. However, the explanation below is high level enough to explain the main parts that need to be considered when making modifications.

Currently, **`layout.ts`** combines all the UI parts of the VSCode workbench together, setting up defaults such as positioning. These parts include the title bar, status bar, sidebar and most importantly the panel. Note that **`workbench.ts`** actually creates all the UI parts. The panel part is by default positioned at the bottom and in itself contains "view panes" for Problems, Output, Debug Console and Terminal.

In **`terminalView.ts`** under **`src/vs/workbench/contrib/terminal/browser`**, there is a class called **`TerminalViewPane`** which is the class that defines how the body of the terminal view pane is rendered and which actions are available. As the intention is to modify it such that each terminal is accessed through tabs, we can remove the selection, addition and deletion terminal actions from the **`getActions()`** method in **`terminalView.ts`**. By doing this, the actions in the red boxes in the picture below would be removed, as desired.

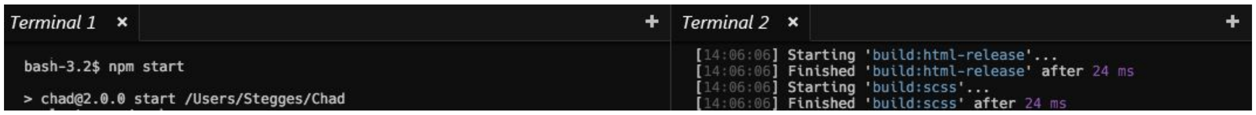


```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
```

These actions were removed such that the new UI can support tabbed selection of terminals, giving the ability to remove each terminal through its tab (similar to Chrome tabs or VSCode editor tabs).

Now looking into **`terminalService.ts`** (in **`workbench/contrib/terminal/browser`**), there is only one **`TerminalViewPane`** and multiple **`TerminalTab`** instances. A **`TerminalTab`** class in **`terminalTab.ts`**, holds the current view of the terminals. When selecting a terminal normally through the selection action, a **`TerminalTab`** becomes "active" in a sense where all other instances of **`TerminalTab`** are hidden (specifically **`display: none`** is CSS). Note that a **`TerminalTab`** can hold more than one terminal since two or more terminals can be in a **`TerminalTab`** through the split action in instances of **`SplitPane`** of a **`SplitContainer`** in a **`TerminalTab`**. This observation can be seen in **`terminalTab.ts#351`** with **`SplitPane`** and **`SplitContainer`** defined in that same file.

As we want the split action to allow us to split terminals with each terminal area having its own tabs, a large chunk of code needs to be modified and added to make this happen because each **`TerminalTab`** can't have a **`SplitContainer`** to hold different split panes. The picture below depicts what is being requested from the pull request.



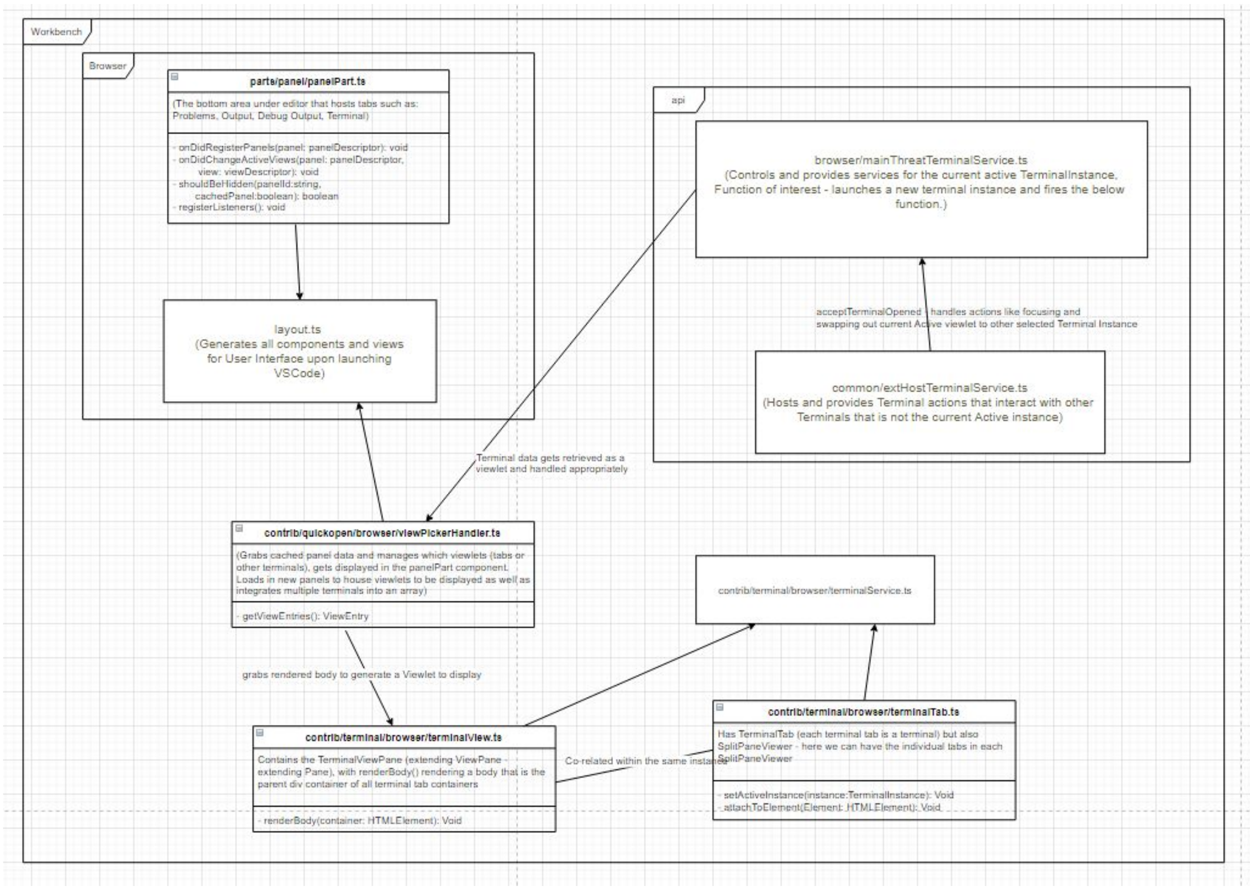
```
bash-3.2$ npm start

> chad@2.0.0 start /Users/Stegges/Chad
```

```
[14:06:06] Starting 'build:html-release'...
[14:06:06] Finished 'build:html-release' after 24 ms
[14:06:06] Starting 'build:scss'...
[14:06:06] Finished 'build:scss' after 24 ms
```

Now, for this to work, we must create a new class such as **`TerminalTabGroup`** that holds instances of **`TerminalTab`**. Each **`TerminalTabGroup`** would contain some of the same functionality as **`TerminalTab:`** to enable a **`SplitContainer`** to house instances of **`TerminalTab`** instead of **`SplitPane`**. Meaning that the split terminal action must be replaced such that it does not split tabs in a **`TerminalTab`** but rather split tabs between instances of **`TerminalTab`**, disabling splitting inside instances of **`TerminalTab`** themselves. Different actions to also create new tabs and select tabs must also be added in addition to the existing actions, but for this configuration specifically.

To ensure this feature is optional, we can create a new flag in VSCode settings named **`terminal.integrated.tabbedTerminals.`** In order to do this, we must modify **`terminal.contribution.ts`** and **`ITerminalConfiguration`** with the new setting and check for this flag when deciding whether or not to enable the configuration in the terminal service (**`terminalService.ts`**).

Below is a depiction of how the aforementioned component's architecture is structured


Workbench
Browser
parts/panel/panelPart.ts
(The bottom area under editor that hosts tabs such as: Problems, Output, Debug Output, Terminal)
- onDidRegisterPanels(panel: panelDescriptor): void
- onDidChangeActiveViews(panel: panelDescriptor, view: viewDescriptor): void
- shouldBeHidden(panelId:string, cachedPanel:boolean): boolean
- registerListeners(): void
layout.ts (Generates all components and views for User Interface upon launching VSCode)
api
browser/mainThreatTerminalService.ts (Controls and provides services for the current active TerminalInstance, Function of interest - launches a new terminal instance and fires the below function.)
acceptTerminalOpened - handles actions like focusing and swapping out current Active viewlet to other selected Terminal Instance
common/extHostTerminalService.ts (Hosts and provides Terminal actions that interact with other Terminals that is not the current Active instance)
Terminal data gets retrieved as a viewlet and handled appropriately
contrib/quickopen/browser/viewPickerHandler.ts
(Grabs cached panel data and manages which viewlets (tabs or other terminals), gets displayed in the panelPart component. Loads in new panels to house viewlets to be displayed as well as integrates multiple terminals into an array)
- getViewEntries(): ViewEntry
contrib/terminal/browser/terminalService.ts
grabs rendered body to generate a Viewlet to display
contrib/terminal/browser/terminalView.ts
Contains the TerminalViewPane (extending ViewPane - extending Pane), with renderBody() rendering a body that is the parent div container of all terminal tab containers
- renderBody(container: HTMLElement): Void
Co-related within the same instance
contrib/terminal/browser/terminalTab.ts
Has TerminalTab (each terminal tab is a terminal) but also SplitPaneViewer - here we can have the individual tabs in each SplitPaneViewer
- setActiveInstance(instance:TerminalInstance): Void
- attachToElement(Element: HTMLElement): Void

Selected Issue

We selected this issue because we thought it offered enough of a challenge, as we have to investigate how the tab dragging behavior is being completed, as well as enabling the actions that can be done on a single tab to be done for multiple tabs. The testing will be a suite consisting of unit and acceptance tests.

This feature would help improve the productivity of VS Code users because it helps with better organizing tabs into specific groups and saves time for when they want to move multiple tabs to another view. Plus, similar features of selecting and moving multiple tabs have been implemented in web browsers such as Firefox and Chrome.

Acceptance Tests Suite

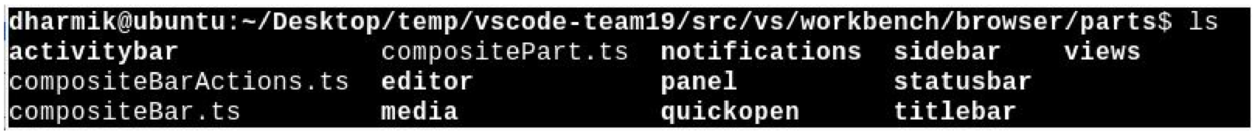
- When the control key is held down, they can select multiple tabs
- After an user select multiple tabs, they can drag multiple tabs
- When an user drag multiple tabs, they are all moved
- When an user drag multiple tabs to another group, the tab that was currently opened in the selection will be opened in the new group
- After an user select multiple tabs, they can right click to bring up the context menu
- When an user selects an option on the context menu, the action is applied to all selected tabs

Re-visit: Architecture Overview

To recap, the architecture of VS Code consists of five layers:

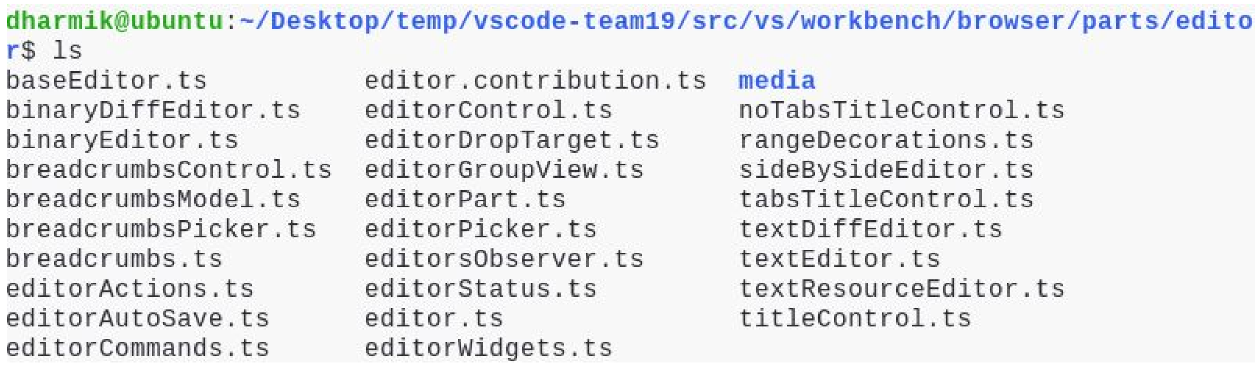
- **Base**: provides utilities and basic building blocks required to construct user interfaces, such as defining the behaviour of the UI elements using CSS.
- **Platform**: provides the basic services of VS Code, mostly consisting of definitions and exported interfaces that interact with each other using dependency injections.
- **Editor**: contains the independent Monaco Editor, which serves as the text editor portion of VS Code, along with implementations of editor-specific commands.
- **Workbench**: implements the elements surrounding the Monaco Editor.
- **Code**: where all of the components of the previous four layers are integrated together using the Electron framework.

After better understanding the code structure, we realized that most of our edits will need to occur in the **workbench** layer. The main file we will need to be modifying is **`/browser/parts/editor/tabsTitleControl.ts`**, which is inside that layer. This takes care of all things tabs, which includes initializing them, closing them, moving them around, and their on click listeners. We will need to extend the listener which listens for a mouse click on the tabs, and create a new feature in which we can select more than one. Then we will need to modify the move listener, to take into account multiple tabs if there are any.



```
dharmik@ubuntu:~/Desktop/temp/vscode-team19/src/vs/workbench/browser/parts$ ls
activitybar               compositePart.ts  notifications  sidebar    views
compositeBarActions.ts    editor            panel          statusbar
compositeBar.ts           media             quickopen      titlebar
```

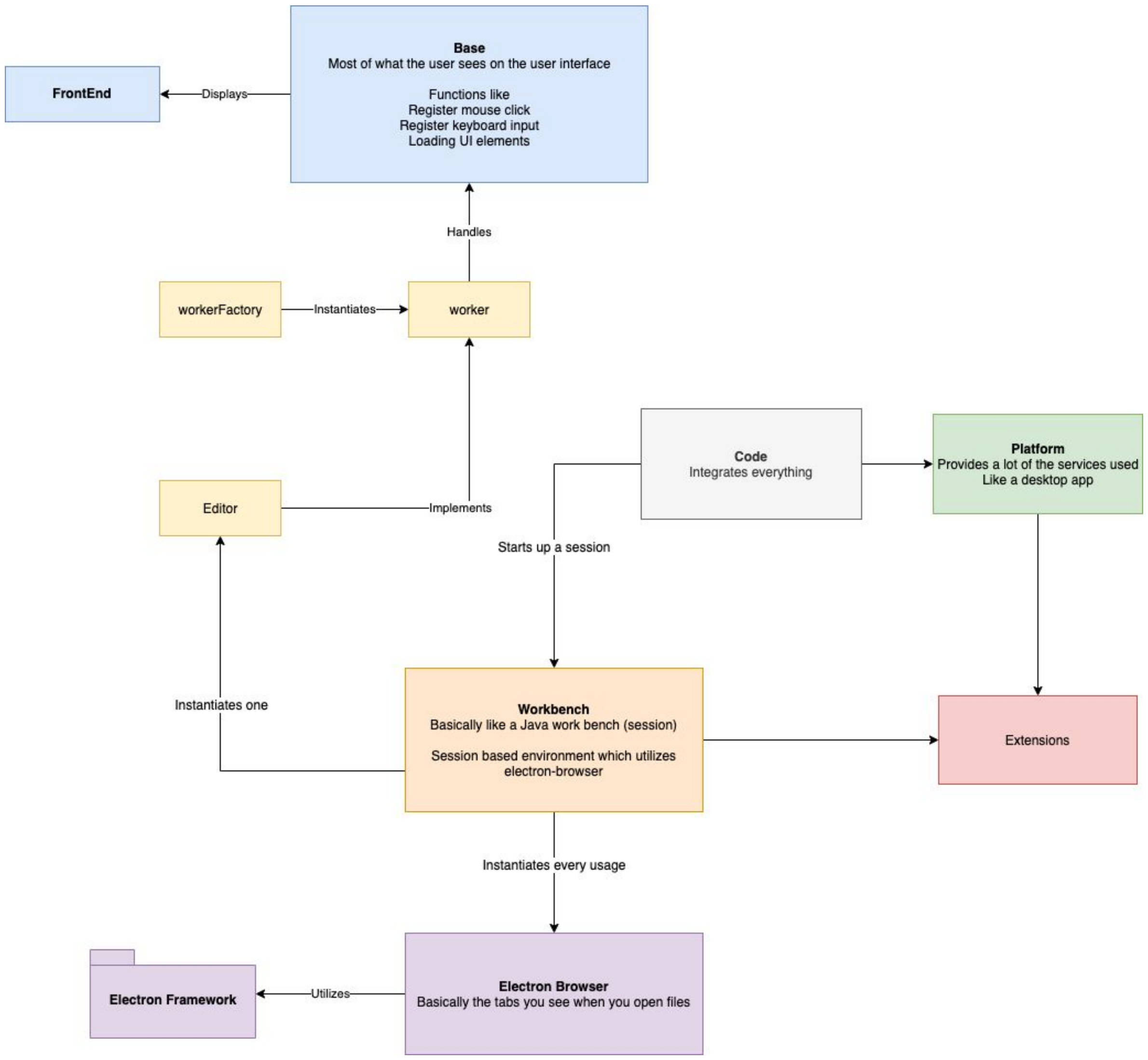
Inside the parts folder, it separates the logic into specific sections of the browser. For example, this includes the editor section, the side panel sections, the title bar, the sidebar, etc. Our changes are mostly inside the editor, which comprises files you will see in the next image.



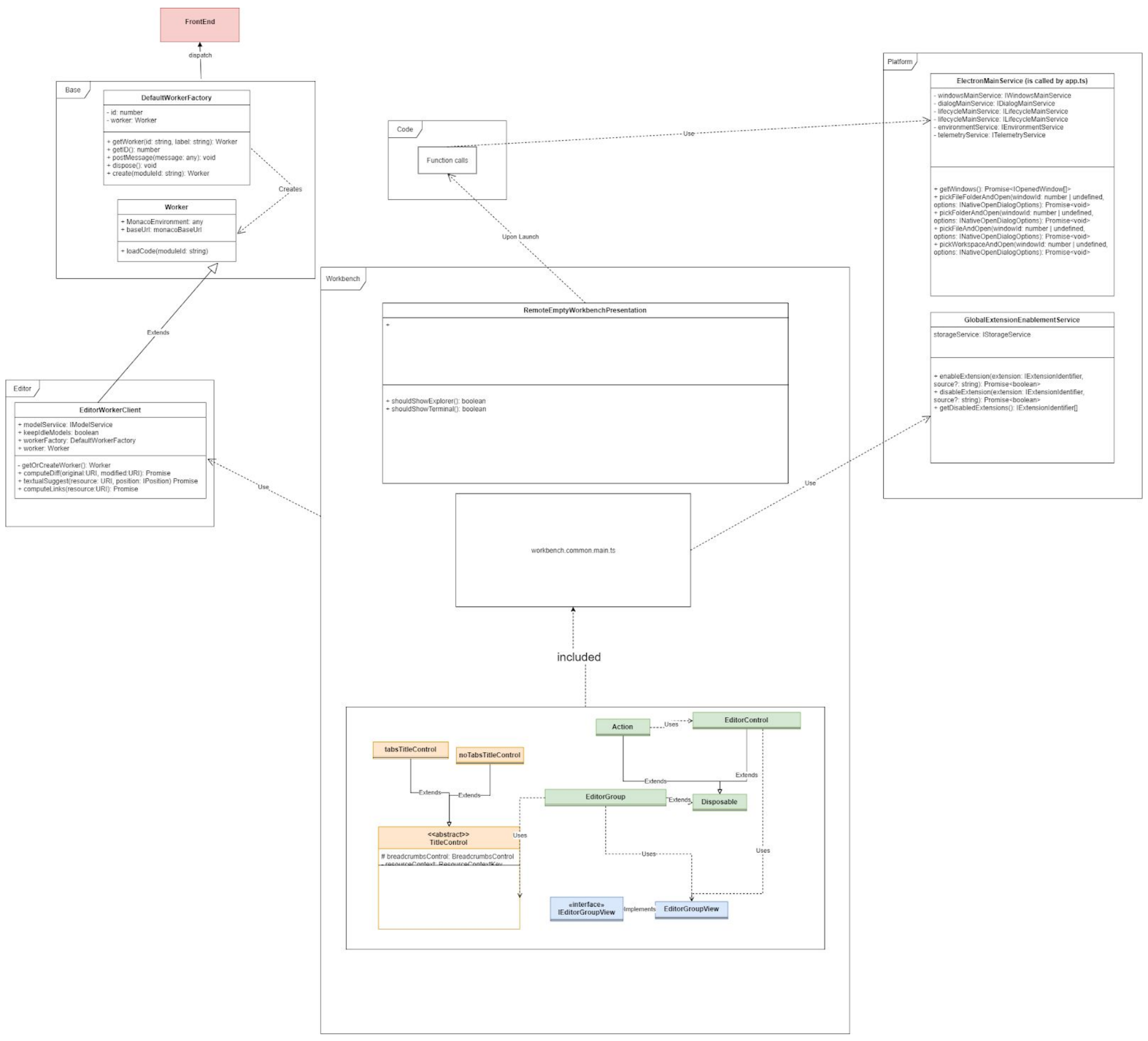
```
dharmik@ubuntu:~/Desktop/temp/vscode-team19/src/vs/workbench/browser/parts/edito
r$ ls
baseEditor.ts              editor.contribution.ts  media
binaryDiffEditor.ts        editorControl.ts        noTabsTitleControl.ts
binaryEditor.ts            editorDropTarget.ts     rangeDecorations.ts
breadcrumbsControl.ts      editorGroupView.ts      sideBySideEditor.ts
breadcrumbsModel.ts        editorPart.ts           tabsTitleControl.ts
breadcrumbsPicker.ts       editorPicker.ts         textDiffEditor.ts
breadcrumbs.ts             editorsObserver.ts      textEditor.ts
editorActions.ts           editorStatus.ts         textResourceEditor.ts
editorAutoSave.ts          editor.ts               titleControl.ts
editorCommands.ts          editorWidgets.ts
```

As you can see here, the files correspond to certain sections of the editor. This includes breadcrumbs (the displays you see when you hover over text), and also tabsTitleControl (anything related to tabs). This is what we will modify.

High-level diagram (deliverable 1):


Base
Most of what the user sees on the user interface
Functions like
Register mouse click
Register keyboard input
Loading UI elements
FrontEnd
Displays
Handles
workerFactory
Instantiates
worker
Code
Integrates everything
Platform
Provides a lot of the services used
Like a desktop app
Editor
Implements
Starts up a session
Instantiates one
Workbench
Basically like a Java work bench (session)
Session based environment which utilizes electron-browser
Extensions
Instantiates every usage
Electron Framework
Utilizes
Electron Browser
Basically the tabs you see when you open files

Expanded high-level diagram


FrontEnd
dispatch
Base
DefaultWorkerFactory
- id: number
- worker: Worker
+ getWorker(id: string, label: string): Worker
+ getID(): number
+ postMessage(message: any): void
+ dispose(): void
+ create(moduleId: string): Worker
Creates
Worker
+ MonacoEnvironment: any
+ baseUrl: monacoBaseUrl
+ loadCode(moduleId: string)
Extends
Editor
EditorWorkerClient
+ modelServiice: IModelService
+ keepIdleModels: boolean
+ workerFactory: DefaultWorkerFactory
+ worker: Worker
- getOrCreateWorker(): Worker
+ computeDiff(original:URI, modified:URI): Promise
+ textualSuggest(resource: URI, position: IPosition) Promise
+ computeLinks(resource:URI): Promise
Use
Code
Function calls
Use
Upon Launch
Workbench
RemoteEmptyWorkbenchPresentation
+
+ shouldShowExplorer(): boolean
+ shouldShowTerminal(): boolean
workbench.common.main.ts
Use
included
tabsTitleControl
noTabsTitleControl
Extends
Extends
<<abstract>>
TitleControl
# breadcrumbsControl: BreadcrumbsControl
- resourceContext: ResourceContextKey
Uses
Action
Uses
EditorControl
Extends
Extends
EditorGroup
Extends
Disposable
Uses
Uses
«interface»
IEditorGroupView
Implements
EditorGroupView
Platform
ElectronMainService (is called by app.ts)
- windowsMainService: IWindowsMainService
- dialogMainService: IDialogMainService
- lifecycleMainService: ILifecycleMainService
- lifecycleMainService: ILifecycleMainService
- environmentService: IEnvironmentService
- telemetryService: ITelemetryService
+ getWindows(): Promise<IOpenedWindow[]>
+ pickFileFolderAndOpen(windowId: number | undefined, options: INativeOpenDialogOptions): Promise<void>
+ pickFolderAndOpen(windowId: number | undefined, options: INativeOpenDialogOptions): Promise<void>
+ pickFileAndOpen(windowId: number | undefined, options: INativeOpenDialogOptions): Promise<void>
+ pickWorkspaceAndOpen(windowId: number | undefined, options: INativeOpenDialogOptions): Promise<void>
GlobalExtensionEnablementService
storageService: IStorageService
+ enableExtension(extension: IExtensionIdentifier, source?: string): Promise<boolean>
+ disableExtension(extension: IExtensionIdentifier, source?: string): Promise<boolean>
+ getDisabledExtensions(): IExtensionIdentifier[]