BERGISCHE
UNIVERSITÄT
WUPPERTAL

Fakultät für Mathematik und Naturwissenschaften

Bachelor-Thesis

VSWorkbench: An Extensible Visual Studio Code Plugin for Bridging the Gap between Key Developer Tools

Sufyan Dahalan

1836674

Informatik

Wuppertal, den 31. August 2022

Betreuer Dr. Holger Arndt

Erstgutachter Dr. Holger Arndt

Zweitgutachter Dr. Marcel Schweitzer

BERGISCHE UNIVERSITÄT WUPPERTAL
RAINER-GRUENTER-STR 21
42119 WUPPERTAL
TELEFON (o2o2) 439 – 1

FACHBEREICH E
ELEKTROTECHNIK / INFORMATIONSTECHNIK / MEDIENTECHNIK
Univ.-Prof. Dr.-Ing. Dietmar Tutsch

Bachelor Thesis

KANDIDAT	Max Mustermann
MATRIKELNUMMER	123456
STUDIENGANG	Informationstechnologie
STUDIENRICHTUNG	IS
BETREUER	Vorname Name

THEMA

Entwurf und Entwicklung eines Lorem-Ipsum-Generators

AUFGABENSTELLUNG

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer libero erat, tincidunt quis molestie nec, ultrices nec felis. Cras tincidunt tempor sapien ac cursus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nunc eu magna ut sem condimentum posuere. Nulla ullamcorper sapien et sem placerat in blandit libero tempor. Pellentesque non justo in arcu porta lacinia non eget massa. Integer vel lectus sed ipsum sagittis mollis. Cras congue, orci et suscipit tristique, enim metus congue ante, et adipiscing neque justo eget mi. Aliquam ut ligula tortor, eu commodo ante. Nam faucibus lorem ultricies metus suscipit cursus. Maecenas adipiscing convallis felis, mattis sollicitudin sapien aliquam eget. Vivamus cursus mattis massa id scelerisque. Quisque dolor tellus, bibendum in adipiscing in, imperdiet vel augue. Fusce posuere lacus vel neque molestie in congue leo ultrices.

Wuppertal, den

.....
(Unterschrift des Betreuers)

ERSTGUTACHTER : Prof. Dr.-Ing.
ZWEITGUTACHTER : Prof. Dr.-Ing.

Prüfungsamt Kennziffer :
Ausgabedatum :
Abgabedatum und Signum :

.....
(Unterschrift)

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Wuppertal, den 31. August 2022 _____

(Unterschrift)

Einverständniserklärung

Ich bin damit einverstanden, dass meine Abschlussarbeit wissenschaftlich interessierten Personen oder Institutionen zur Verfügung gestellt werden kann. Korrektur- oder Bewertungshinweise in meiner Arbeit dürfen nicht zitiert werden.

Wuppertal, den 31. August 2022 _____

(Unterschrift)

Kurzfassung

Der Text der Kurzfassung wird hier eingetragen. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Abstract

The english version. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

1 Introduction

Software has its deep roots in the scientific communities and largely most of its early application in research. As Software found its way into commercialization, free from most hardware limitations, and was made generally available to the public through the internet/web, the demand for programmers and software developers soared in a way that made it hard to keep up with demand. This great demand and meager supply of programmers made it a highly vital task to squeeze efficiency out of the very few programmers available. Hence developer tools.

Developer tools try to achieve a big subset of things, enabling developers to cooperate on a higher scale (communication software ,e.g. teams, zoom, email, collaboration tools for technical and nontechnical staff, e.g. jira, trello, atlassian wiki, collaboration tools for technicals that touch on many different subjects, e.g. git, GitLab, github, time saving tools made to accelerate software development, e.g. CICD, GitLab ci, github action, error reduction toos, e.g. automated testing frameworks like cypress, playwright, and debugging).

These developer tools are essential in the work life of a programmer. Use them fluently and efficiently, and your productivity will see an exponential growth.

2 Problem and Goal

2.1 Affects of Context Switching on Developer Productivity

Research shows that there is a special state of mind in which a person shows hightened focus.

Developer's heavily rely on getting into the 'zone', also called flow in scientific circles, in order to get through their tasks.

Flow is essentially characterized by the complete absorption in what one does, and a resulting transformation in one's sense of time and improving one's productivity [3]. todo paraphraze

Research shows that to get into the state of flow, a developer needs a an average of 10-15 minutes of clear thinking and uninterrupted work [14].

Interruptions, however, can lead to a reset of the flow state. The developer would then need to boot up his flow state again, where he or she will need another 10 minutes of bootup focus to get in his/her flow state again.

VSWorkbench is a trial to minimize these interruptions in the context of developers who utilize GitLab in their daily life. VSWorkbench should cut the time needed to look up a project, search for an issue or looking where that button is again by bringing the most used functionalities of GitLab closer to where the developer is usually to be found when he or she is in his or her state of flow: the code editor.

The goal of VSWorkbench within the framework of this bachelor's thesis is to bring the 20% of functionality of GitLab as a start that is used 80% of the time and present them in a familiar manner that is not much distinguishable from the native webapp GitLab experience, with the end goal being minimizing interruptions and hightening developer's productivity.

3 Fundamentals

3.0.1 Tech Stack

HTML

HTML (HyperText Markup Language) is the standard language used to used to create hypertext documents that are platform independent, usually rendered on a browser. HTML documents can be used with generic semantics to represent information from a wide range of domain, including but not limited to: mail, hypermedia, news, documentation, or simple structured documents with inlined graphics. [1] HTML in its current form traces its origins to a Request For Comment [1] by Tim Berners-Lee and has ever since been developed by a set of companies (for WHATWG: Apple, Google, Mozilla, Microsoft) under the umbrella of the W3C and the WHATWG. HTML is a living standard, meaning changes occur without maintaining or incrementing a version number. Informally , however, the HTML living standard is called HTML5.

CSS

CSS is a style sheet mechanism that allows web page authors and readers to attach style [9]. Using CSS, a developer can, among other things, specify fonts, colors and spacing. Latest CSS standard is CSS3, put forward by the W3C [9]. It has also been standardized in (TODO year) and progress and future development on it is managed by a joint committee under the umbrella of the W3C.

Javascript

Javascript (often abbreviated JS) is an event driven language developed to be used in browsers. It features JIT, or just-in-time-compilation, which means that the javascript files received by the browser will be compiled and run simultaneously. The two most common variants of Javascript are CommonJS and EcmaScript. While EcmaScript is developed solely for the browser, packaging DOM TODO explain more manipulation libraries with it, CommonJS is developed for server-side use, therefore it is shipped with modules that enable Javascript to interact with its hardware, including but not limited to IO, template engines, object relational mappers, and middleware [5].

[1] An RFC is a document that contains technical specifications and organizational notes for the Internet. For more information see https://www.ietf.org/standards/rfcs/.

The Javascript variant used in this project is EcmaScript.

Typescript

Typecript is a statically typed subset of js todo cite wikipedia or smth, developed and maintained by Micosoft. While there were multiple trials to create a statically typed language that transpiles to Javascript, Typescript is the langauge that saw the most adoption by the community, consistently scoring high on StackOverflow Developer Surveys [10][11]. Typescript currently has approximately 82.9k stars on GitHub, making it the 48th most starred repository on GitHub currently [6].

tsconfig, TS interfaces, eventemitter and their use in notifications across contexts

3.1 The modern web architecture (web 2.0)

3.1.1 Client-Server Architecture

The client-server architecture describes the relationship between the device of a website owner (server) and the devices of users (client). It has its roots in the early ARPANET days in the 1970s where the Stanford researchers worked toward creating interactive programms that function across computer networks [13]. The server receives HTTP requests across the network and returns data, usually in the form of HTML, CSS, JS for the rendering of a website or raw data, usually as JSON or raw text data.

The client-server architecture is used as an abstraction to simplify the workflow of clients. The client does not need to know todo - write what it abstract and whatnot. Conversely, the server must take care of business logic required to retrieve data requested by the client, and return an internet package, containing the response, in a way that follows common API specifications such as ReST or GraphQL todo cite smth? idk. The client then takes care of building a user friendly interface to hold the information given back by the server.

Figure 3.1 GraphQL Query and Result

3.1.2 HTTP methods, REST, GraphQL

HTTP [7] is an application-level protocol for distributed, collaborative information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext through extension of its request methods, error codes and headers. It defines the specification through which common internet communication between systems happens. It sets standards for interactions between different systems, most notabely the HTTP GET and HTTP POST methods. The HTTP GET method is used to retrieve data from an API. In case authentication is required, it is achieved through a special `Authentication` header, enabled by the generic nature of HTTP. An HTTP POST method is used to transfer data back to the API, e.g. to perform a set of operations based on the data transferred or to simply save it.

REST defines a way to todo

todo attach a screenshot of the api.ts file, showcasing REST api requests

GraphQL is a technology that defines a way to query data from the server, enabling the developer to get exactly the data he or she needs, nothing more and nothing less [2]. GraphQL is a project released by developed by Facebook and released to the opensource community in 2015. Since then, its further developement has been a collaboration between Facebook and volunteers from open source community.

GraphQL enables a frontend developer todo explain what a frontend developer does in a footnote to describe the data that he or she wants to receive. This way, the developer gets exactly the data he or she asks for, allowing the internet packets to be smaller when allowed, improving performance for the end user. GraphQL can only be used in combination with a POST request, because it strictly needs an `HTTP body`, a feature that GET requests lack according to the HTTP spec[7]. It features a simple intuitive syntax.

3.1.3 iframes

An `iframe`, or an inline frame, is an HTML element that is used to embed an HTML document into another. It has many uses, most notably embedding web resources of other companies, websites into others, e.g. youtube videos on wordpress websites or personal blogs. The way it works is as follows:

The `iframe` HTML element was first introduced by Microsoft Internet Explorer in 1997, they were a notable addition to HTML5 iframes are used as a way of guerrela marketing, e.g. iframes of youtube videos, or collaboration and arbitrary extension of a document, such as in vscode.

While iframes present a challenge for search engines and accessibility readers, they are a considerable solution for extending web applications.

3.1.4 vscode and electron js

ElectronJS was started in 2013 [12] to enable web developer to build cross platform desktop application in a familiar way to web development. Electron apps (such as microsoft teams, vscode and others) are bundled together with the nodejs runtime and chromium browser engine, causing a simple hello world programm to reach hundreds of Megabytes in size. Electron, however, allows developers to benefit from the already existing cross platform compatibility of browsers and lowers the bar for companies building desktop apps by allowing them to tap into the abundant web developer talent.
ElectronJS essentially proposed a new perspective into desktop app development. Its wide adoption, inspite of its lack of performance compared to traditional technologies, speaks for its success.

Visual Studio Code (short: VSCode) is an open source cross platform language agnostic extendible code editor started in 2015 by microsoft [8]. It was intended to be part of Microsoft's trend towards open source. It was based in its core on open source/open standard technologies such as HTML, css, javascript and electronjs, with all developers as end users in mind. Its extensibility makes it appealing to web developers and vim/emacs users.

Furthermore, major companies have endorsed VSCode by writing language/platform specific extensions for them and offering them free of charge. Examples of such are the Docker extension by Microsoft, java language support extensions by redhat and microsoft, and many other language support and cloud development extensions developed by major companies. Each and every extension of these are a step into bringing together the scattered bits and pieces of developers' workplace and tools into one workbench.

3.1.5 Authentication, Authorization and Personal Access Tokens

Authentication is the process of ensuring the identity of the user is correct. Authorization is the process of checking priviliges of a user during the processing of an action started by the user.

Personal Access Tokens (short: PATs) are user specific unique strings that can be generated by the service. All valid PATs can be assigned to one unique user. On the other hand, a user can create multiple PATs with varying privilige levels. PATs are usually issued with an expiration date, in the case of microsoft and visualstudio.marketplace.com, it can reach a max of 120 days todo is it really 120 or is it 180. However, in some cases, such as GitLab, PATs can be issued by the user to never expire.

PATs are used to gain access to APIs and services [2]. The GitLab API, in its self-hosted version as well as the SAAS version, checks if the user has enough priviliges before executing the action, and provides an HTTP response based on the result. If authentication information is not valid or is missing, GitLab API returns an error message with a status code of 401:

```
{
    "message": "401 Unauthorized"
}
```

todo add screenshot of user priviliges / access levels, and screenshot of exemplary reponses for a 200 and a 401

3.1.6 Version Control Software and Git

As projects grow in complexity and software teams grow in size, a coordination and communication overhead starts to take place. One of the developer tools Used to minimize this overhead is Source Code Management (short: SCM).

SCM tools enable multiple developers to work individually and simultaneously on the same files, incrementally introducing changes in order to implement different features without much friction. With the help of SCM, multiple versions of the same software can coexist, where they can be deployed to different environments for different purposes (testing, staging, and production).

The most widespread SCM tool currently is Git, a project started by the linux kernel maintainer Linus Torvalds in 2005 for use in the development of the Linux Kernel. It was developed with the intent to keep it open source, in contrary to previous SCM tools that were

[2] see https://docs.GitLab.com/ee/api/index.HTML

proprietary and in part paid [3].

Git was designed to be fast and efficient. As it follows the unix philosophy of programming [15], its functionality can also be extended [4].

3.1.7 GitLab

As git started to gain traction, multiple trials to introduce a graphical user interface where made. The most successful were webapp based user interfaces, which implies the existence of a central repository (or a central source of truth). Most notable examples are GitHub, GitLab, and BitBucket. These services also began to extend their functionlality beyond being a simple git graphical interface, integrating state of the art tools like CICD and tools that adhere to new methodologies that make developing and deploying software to users easier, more frictionless and more efficient. GitLab therefore started integrating more and more CICD tooling and support, starting in 2012. Over the course of the years, GitLab developed and integrated more DevOps tools, enabling developers to publish their software packages through GitLab [5]. As a result, GitLab was able to build a platform that is effectively a fully featured toolbox; developers can develop, collaborate, communicate, build and release their libraries, packages and software incrementally and efficiently.

Furthermore, developers can incorporate user support and communication into GitLab using features like Service Desk [6].

GitLab is working towards an all encompassing platform that eliminates that need for multiple scattered services, each with its own maintenance and administration overhead. This is essentially the philisophy that inspired the development of VSWorkbench.

GitLab exposes REST and GraphQL APIs[7] for users to enable automation. These APIs are, unlike the GitHub APIs, publicly available.

While the APIs are publicly available, some API endpoints are only accessible by instance administrators, which applies to the selfhosted version of GitLab, or are available to premium accounts. Additionally, some endpoints are only invokable on a selfhosted instance, e.g. the createGroup API Endpoint [8].

```
1 curl "https://git.uni-wuppertal.de/api/v4/projects"
```

[3] See BitKeeper, StarTeam, Rational Synergy, and Vault VCS
[4] see Git Flow
[5] e.g. nuget packages for dotnet, NPM packages for node
[6] See Service Desk Documentation
[7] See GitLab Documentation
[8] See documentation reference

Sourcecode 3.1 Example of the usage of the GitLab API of the University of Wuppertal

3.1.8 Module Bundlers and Webpack

Module bundlers are tools that combine multiple JS files into one. They can additionally minimize the file (by e.g. ommiting extra spaces and using shorter names for variables and functions). In addition to that, the output file can have a concatenated hash in its name in order to support better caching. The end result will be automatic inclusion of new JS code from new files into the webpages/websites/webapps per the config of the module bundler.

A famous module bundler is wbepack. Webpack was started in 2014 by Tobias Koppers in order to simplify bundling/compilation workflow of JS(HTML? CSS?) based projects. Webpack is used in VSWorkbench to seperately build the extension and the three vanilla TS apps in a way that facilitates them working as one unit. Webpack also bundles and (optionally) inlines css styles.

3.1.9 Docusaurus and GitLab CI.

VSWorkbench aims to be opensource and will be extended through collaboration by the open source community. To enable this and make the contribution process as frictionless as possible, documentation has to be tackled.

Docusaurus [9] is a tool developed by Facebook that enables developers to write documentation in markdown and deploy it without needing to dive into the details of building and deploying a documentation website. It outputs a static website and does not create the need for server side services that will need maintenance and will incur costs. The static build can also be deployed on GitLab using GitLab pages with the help of cicd and GitLab ci.

The process is as follows:

1. The developer defines a GitLab CI job, named `pages`, in the repository's GitLab-ci.yml file.
2. When a commit is pushed into remote git repository on the GitLab server, the `pages` job runs on the repository. It builds the website according to the steps predefined in the GitLab ci job, and saves the build in an artifact
3. The GitLab ci artifact, which contains the website build, will then be accessible by the end users through a domain made availabe by GitLab. It usually follows the syntax [GitLab

[9] See Docusaurus

username].GitLab.io/[repository name]. Alternatively, the developer can bind the artifact to a domain name of his own choosing, e.g. VSWorkbench.Dahalan.De

4. When an enduser navigates to the url to which the artifact is mapped, the GitLab servers/CDN? send a reply with the artifact. The artifact will contain HTML, CSS, JS, and possibly assets (images, pdfs, etc.) that the end user's browser can parse and build the website from.

GitLab CI and docusaurus dramatically decrease the overhead of generating and hosting documentation, enabling the developer to focus on major tasks such as new features, refactoring or pull requests.

3.2 NPM

NPM (short for Node Package Manager) is one of the most popular pacakge managers for javascript (along with yarn and others) created by Isaac Z. Schlueter and maintaned by npm, Inc. Due to its popularity, it is the default package manager for the JavaScript runtime environment Node.js. NPM the CLI program is used to install packages, run tests available in a specific project and run scripts.

When using NPM, it will create a file in the project by the name of `package.json`. In the `package.json` file, packages along iwth their versions are defined, scripts that can be run using `npm run XXX` and meta data that can be used by platforms hosting projects to display information.

An example of this is defining the repository link in the `package.json` file, where it will be linked by visual studio marketplace to give the extension viewer quick access to the repository, if it is opensourced. An extension publisher can also define different tags that will help users in visual studio marketplace to search and find his or her package.

The `package.json` file furthermore allows its user to define custom reusable scripts in the sections block.

```
    "scripts": {
        "vscode:prepublish": "npm run package",
        "ts": "npm run esbuild-base -- --minify",
        "compile": "webpack",
        "watch": "webpack --watch",
        "package": "webpack --mode production --devtool
        ↪hidden-source-map",
```

```
        "compile-tests": "tsc -p . --outDir out",
        "watch-tests": "tsc -p . -w --outDir out",
        "pretest": "npm run compile-tests && npm run compile &&
        ↪ npm run lint",
        "lint": "eslint src --ext ts",
        "test": "node ./out/test/runTest.js",
        "deploy": "vsce publish --no-yarn",
        "postinstall": "cd docs && npm install && cd ..",
    },
```

The above code block showcases an excerpt from the `package.json` file used in developing VSWorkbench. It defines commands to test, compile and package, continiously build and test the extension. Furthermore, it also defines steps that are run before testing (pretest script) and after installing packages (postinstall script).

The `package.json` file is a tool that facilitates contribution and collaboration between developers by enabling them to declaratively define the dependencies needed to develop and deploy the project.

Futhermore, the `package-lock.json` file will be automatically created and updated after each package installation. It describes the exact dependency tree that represents the node_modules file in order to build the exact dependency tree again on different machines, ensuring the all developers will always have the same dependencies with the same versions installed. To build the dependency tree using the `package-lock.json` file, a developer has to run

```
npm ci;
```

in his or her command line interface.

4 Implementation

4.1 VSCode API

To facilitate the process of extending the functionality of VSCode, an API is exposed that can be used by the extension developer. The API provided by VSCode provides visual and functional components.

4.1.1 Tree View

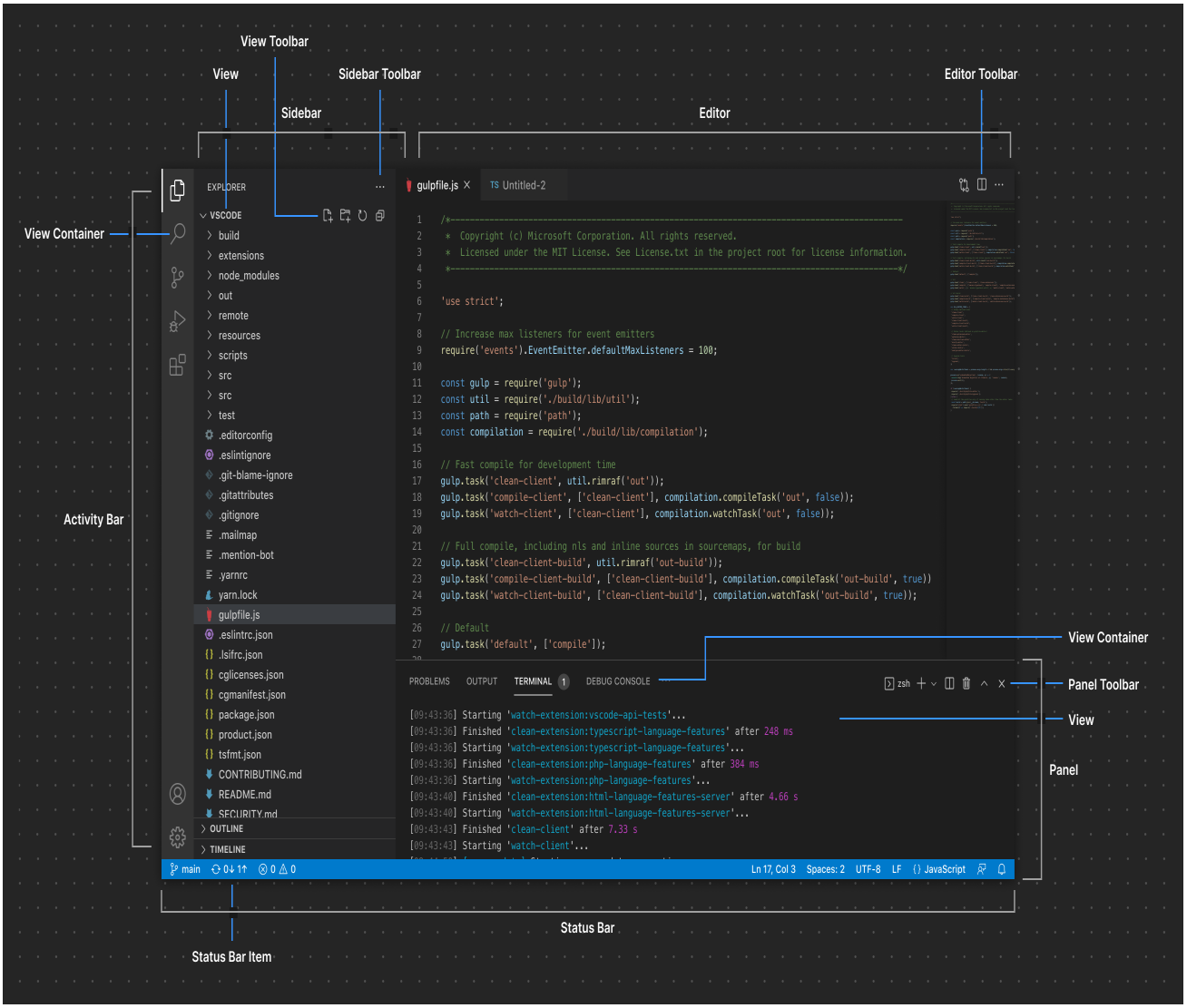
The Tree View API is a visual component exposed by VSCode that can be utilized by creating a class that inherits from the `vscode.TreeDataProvider` class and implementing its functions. The developer also has to define an entry for the implemented class in the `package.json` file.

```
{
...
"contributes": {
    ...
    "viewsContainers": {
        "activitybar": [
            {
                "id": "GitLabCode-activityBar",
                "title": "VSWorkbench",
                "icon": "src/assets/icon.png"
            }
        ],
        "panel": [
            {
                "id": "GitLabcode-panel",
                "title": "VSWorkbench",
                "icon": "src/assets/icon.png"
            }
        ]
    },
    "views": {
        "GitLabCode-activityBar": [
            {
```

```
                "id": "groupView",
                "name": "Groups",
                "contextualTitle": "VSWorkbench",
                "visibility": "visible",
                "when": "VSWorkbench.authenticated"
            },
            ...
            ]
        ...
        }
}
}
```

The `contributes` entry in the `package.json` file declaratively defines the visual and functional exntesions that the extension provides. The `views` entry refers to containers of visual componets, such as the container `GitLabCode-activityBar`. The `GitLabCode-activityBar` container then accepts an array of components, in the above example the `groupView` tree view. The `GitLabCode-activityBar` component container will then be embedded, also declaratively, into the VSCode activity bar.

The following picture visualizes the anatomy and components of VSCode visually.



todo add source. URL: https://code.visualstudio.com/assets/api/ux-guidelines/examples/architecture-sections.png The Drap and Drop API can is used by implementing two functions for the drag and drop functionality respectively in the `GroupTreeDataProvider` that are inhereted from the `vscode.TreeDragAndDropController` class.

```
1  public async handleDrag(source: GroupNode[],
   ↪ treeDataTransfer: vscode.DataTransfer,
2
3    _token: vscode.CancellationToken): Promise<void> {
4      if (source[0].contextValue !== "user") {
5          treeDataTransfer.set("application/vnd.code.tree.groupView",
6            new vscode.DataTransferItem(source));
7      }
```

```
    }
    public async handleDrop(target: GroupNode | undefined,
    ↪ sources: vscode.DataTransfer,

     _token: vscode.CancellationToken): Promise<void> {
        const transferItem =
        ↪ sources.get("application/vnd.code.tree.groupView");
        ...
```

The `handleDrag` function takes two GroupNode as a parameter in addition to a cancellation token paramter todo explain more. The source, the GroupNode that is currently being dragged and a `vscode.DataTransfer` object. The function checks if the source object is the user namespace todo exaplin more about namepsaces n stuff, which is not a group and cannot be moved or deleted. If the node is not a user namepsace, i.e. if it is a group or a project node, its status will be set to be dragged.

This prevents user namespaces from entering the `handleDrop` function, which would potentially result in illegal actions that will be later rejected by the GitLab API.

The `handleDrop` function accepts three parameters, the source node, or the node being dragged, and a target node, where the source node was dropped. The target node is allowed to be `null`, which means that the source node was dropped into an empty area where no node exists. If the target node is not `null`, it is of type `GroupNode`. A GroupNode can be a user namespace, a Group or a subgroup, or a project. The function will check if the action is legal, and then call the correct function to conclude the operation. The user can move a project between groups, subgroups and personal namespaces. However; the user cannot move a project into another project or otherwise move a group, subgroup or user namepsaces into projects. Furthermore the user can convert groups into subgroups by moving them into other groups or subgroups, but cannot move them into user namespaces.

The functions that will be called are defined in the `api.ts` file and have the following declarations:

```

    transferGroup(id: number, group_id?: number):
    ↪ Promise<AxiosResponse> {
        return Api.instance.api.post(`v4/groups/${id}/transfer`,

        group_id ? { group_id } : null);
    }
```

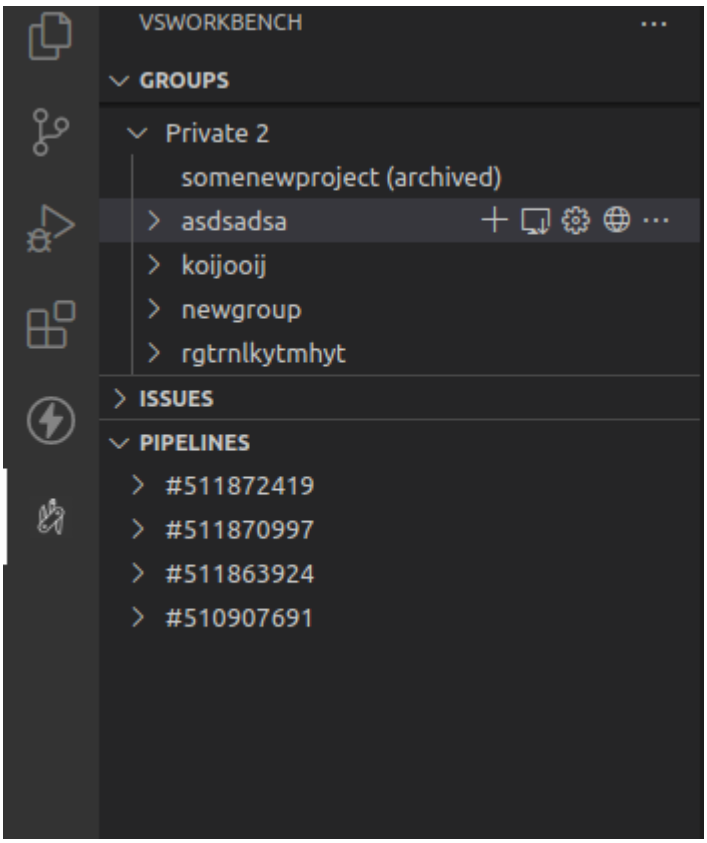


Figure 4.1 The groupView Tree View in VSWorkbench

```
7     transferProjectToGroup(group_id: number, project_id: number):
8
9     Promise<AxiosResponse> {
10        return Api.instance.api.put
11
12        (`v4/projects/${project_id}/transfer?namespace=${group_id}`);
13    }
```

The transferGroup function accepts two arguments, the id of the group (id), and the id of the namespace (group_id) where it should be moved. The reason behind making the 'group_id' parameter optional is to enable to make groups top level groups, i.e. not nested in other groups, depending on this parameter.

4.1.2 Webview

The Webview component is a visual component that is used to utilize the editor space todo reference the reader to the VSCode anatomy by adding an editor tab. It is used by VSWorkbench to display snippets and wiki pages of a specific project.

The webviews, or editor tabs, are in their essence dynamic HTML wrapped in a container. When the user changes the selected tab, javascript is used to instantly delete the previous content of the container, evaluate the new content and append it to the container again.

To utilize the Webview API exposed by VSCode, an instance of the class vscode.WebviewPanel by utilizing the {vscode.window.createWebviewPanel} function.

```
let panel = vscode.window.createWebviewPanel(this.viewType,
↪ `${name} | ${ViewEvents[type]}`, vscode.ViewColumn.One, {
            enableScripts: true,
            retainContextWhenHidden: true,
        });
```

The function is called with three parameters, namely the view type of the webview, its title, the show options for the webview, which specify the location of the webview, and finally a set of optional parameters.

The optional parameters are used to enable the use of javascript in the webview context and to prevent the discardment of the webview context in case it is hidden, or when the user switches to another tab.

Moreover, the webview panel is instantiated without any HTML assigned. Therefore, the developer has to assign it HTML.

The HTML is assigned to the webview using the follow function.

```
private getHTML(webview: vscode.Webview): string {
    const scriptUri = webview.asWebviewUri(
        vscode.Uri.joinPath(this._extensionUri!, "dist",
        ↪ "editor", "main.js"));
    return `<!DOCTYPE HTML>
        <HTML lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width,
            ↪ initial-scale=1.0">
            <meta http-equiv="Content-Security-Policy">
```

```
10              <title>GitLab CI | Editor</title>
11          </head>
12          <body>
13          <div id="app"></div>
14              <script src="${scriptUri}"></script>
15          </body>
16          <script >window.vscode = acquireVsCodeApi();</script>
17          </HTML>‘;
18  }
```

The function `getHTML` first locates the extension in order to locate the Javascript file used for dynamic HTML and css content and assigns the value to the variable `scriptUri`. Secondly, the function embeds the `scriptUri` variable in an HTML script tag so that the HTML document embedded in the can access the Javascript file. Lastly, the function assigns the VSCode API to a variable named `window.vscode` that will later be used for communication between the Webview embedded app/HTML document and the extension context.

To prevent any confusion, the VSCode API refered to in the above snippets refers to an API that can merely send messages back and forth between the extension context and the contexts of the apps/HTML documents embedded and managed by the extension, not the API exposed by vscode to enable the extension to communicate with VSCode and make use of its components.

The embedded app will first import the API class used to communicate with GitLab and instantiate an object. It will stand by until it receives a message from the parent context informing it of the API token for authentication with GitLab and additional information, such as whether the context to be loaded belongs to a wiki or a snippet, and the ID of the object to be loaded.

After receiving the aforementioned information, the embedded app will start querying GitLab for the information needed to model the HTML document around. After the HTML document is ready, the result will be injected via Javascript into the uppermost parent container, the `div` element with the id `app`.

4.1.3 Webview View API

The Webview view API provided by vscode enables the developer to create a context in the panel area of VSCode, where the iconic terminal usually resides.

The webview view API is used by VSWorkbench to render two child contexts that are responsible for integration and poritng of GitLab CI and GitLab Issues into VSCode.

To create a webview view, a class inhereting `vscode.WebviewViewProvider` must be imple-

mented. The class's constructor takes the extension context as a paramater in order to register the new child context. Registering the webview view requires a unique id for the webview view, which also has to be configured in the `package.json` file under the contributions entry. Additionally, the constructor will register the new wbeview view context as a subscriber/listener to the event emitter that is configured by the tree view, in order to recieve updates about the currently selected/focused group node. Moreover, a function inherited from `vscode.WebviewViewProvider` will be called when the webview view is brought into focus, which ensures the creation of the document to be shown and appends it in its rightful location, the VSCode panel.

```
constructor(context: vscode.ExtensionContext) {
    changeValidEmitter.event(this.eventCallback, this);

    vscode.window.registerWebviewViewProvider(this.viewType, this,
    { webviewOptions: { retainContextWhenHidden: true } });
    this._extensionUri = context.extensionUri;
}
```

Similiar to the case of the WebView used in VSWorkbench, this function returns an empty HTML document that includes a link to the script file responsible for the communication with GitLab and the insertion of HTML nodes as needed.

The script file included is written in Typescript. It features its stand alone instance of the api, due to it possessing its own context.

The first steps undertaken by the linked script is to receive the user's API token. Secondly, it will wait for an update regarding the currently selected group node. Once a group node is chosen, the script will fetch the information regarding the group node, which can be either comments related to a group or a project, or pipeline and jobs data regarding a project. Next, the execution of the script will depend on the action of the user.

todo add some swt diagramm/ablauf smth

4.1.4 ExtensionContext.globalState

VSCode also exposes an API to allow the user to make use of the storage managed by VSCode. This allows to store the GitLab instance URL and the authentication token of the user in the

```
function initStorage(context: vscode.ExtensionContext) {
    context.globalState.setKeysForSync([AUTH_TOKEN_KEY]);
    context.globalState.setKeysForSync([GITLAB_INSTANCE_KEY]);
}
```

```
context.globalState.update(AUTH_TOKEN_KEY, gitlabAuthToken); //
↪ call to update an entry
context.globalState.get(AUTH_TOKEN_KEY) // call to get an entry
↪ of a specific key
```

The above snippet shows two callable function which are effectively the getters and setters of the `globalState` object.

The use of globalState further simplifies the task of VSCode extension development and lowers overhead?.

4.1.5 Commands

VSCode Commands facilitate passing commands that interact with VSCode's internals.

The commands API enables the developer to interact with programs installed on the system as well as divers extensions installed in VSCode.

An example of this is the `vscode.git` extension, a default extension that ships with VSCode integrating basic git functionality into the heart of VSCode.

The `vscode.git` extension is used by VSWorkbench to enable the user to clone projects or groups available to him or her.

```
    async cloneNameSpace(): Promise<any> {
        if (this.contextValue === "project") {
            return vscode.window.showErrorMessage("Entity Chosen
            ↪ is not a Namespace!");
        }

        let res = await api.getProjects(this.contextValue ===
        ↪ "group", this.node_id);
        let path: vscode.Uri[] | undefined = await
        ↪ vscode.window.showOpenDialog({
            canSelectFiles: false,
            canSelectFolders: true,
            canSelectMany: false,
        });
        if (path === undefined || path[0] === undefined) {
            return vscode.window.showErrorMessage
```

```
14              ("Please choose a folder to clone into");
15          }
16          res.data.forEach(async (project: any) => {
17              if (!project.archived) {
18                  await cloneFromGitLab(project.http_url_to_repo,
                    ↪ path![0].path);
19              }
20          });
21          return true;
22      }
```

The above code snippet demonstrates the function used by VSWorkbench to clone groups. VSWorkbench first checks that the node, on which the function was called / the button was clicked is indeed a group node. Thereafter VSWorkbench either returns an error message via the VSCode UI API, or proceeds to fetch the information of the child projects of the group chosen by the user. Once the information is fetched, VSCode UI API is once again used to open a dialog todo explain or dedicate a section to these UI APIs? idk, through which the user can choose a directory on his/her PC, onto which the projects will be cloned. Once a folder is chosen, VSWorkbench proceeds to clone the projects that are not archived todo maybe undo it an dstart cloning everyhtnig? idk i just dont wanna get asked about it. one by one.

```
1 export async function cloneFromGitLab(url: string, path: string):
  ↪ Promise<any> {
2     return await vscode.commands.executeCommand('git.clone', url,
      ↪ path)
3 }
```

The above snippet of code shows how to execute commands through the commands api implemented by vscode. in this short snippet the git extension is called with the clone command. The url of the repository and the path onto which the project will be cloned are also specified.

Otherwise, the developer can define commands. Common uses for this is for calls through the `package.json` todo add snippet code file, enabling the users to call the commands directly through VSCode UI todo add screenshot, or to allow other extensions to call VSWorkbench's commands.

To prepare for declaring commands, the developer has to create a declaration in the `package.json` file. Secondly, the function will be implemented in code, and lastly registered with VSCode via the API. todo add snippets

```
...
"contributes":
    "commands": [
                {
                    "command":
                    ↪"VSWorkbench.updatePersonalAccessToken",
                    "title": "Update Personal Token",
                    "category": "VSWorkbench Personal"
                },
                ...
```

```
vscode.commands.registerCommand("VSWorkbench.updatePersonalAccessToken",
async () => {
    await GlobalFunctions.settings(context.globalState);
})
```

```

export async function settings(globalState:
↪vscode.ExtensionContext["globalState"]) {
    // get personal auth token and gitlab instance
    let gitlabInstance = "",
        gitlabAuthToken = "";
    const inputPersonalAuthToken = vscode.window.createInputBox();
    inputPersonalAuthToken.placeholder =
    "Please Enter Your Personal Authentication Token";
    inputPersonalAuthToken.onDidChangeValue((tokenInput) => {
        gitlabAuthToken = tokenInput;
    });
    inputPersonalAuthToken.onDidAccept(async () => {
        inputPersonalAuthToken.hide();
        if (gitlabInstance.endsWith("/") &&
        ↪!gitlabInstance.endsWith("/api/")
        && gitlabInstance !== GitLab_SaaS_Base_URL) {
            gitlabInstance += "api/";
        } else if (gitlabInstance !== GitLab_SaaS_Base_URL) {
            gitlabInstance += "/api/";
        }
```

```
20          let res = await
            ↪ checkGitlabInstanceAndAuthToken(gitlabAuthToken,
21          gitlabInstance);
22
23          if (res) {
24              globalState.update(GITLAB_INSTANCE_KEY,
                ↪ gitlabInstance);
25              globalState.update(AUTH_TOKEN_KEY, gitlabAuthToken);
26              newAuthentication.fire();
27          } else if (!res) {
28              Api.updateAuthToken(globalState.get(AUTH_TOKEN_KEY)
                ↪ as string);
29              Api.updateBaseURL(globalState.get(GITLAB_INSTANCE_KEY)
                ↪ as string);
30              // optionally show some error message depending on
                ↪ whats wrong
31              inputGitlabInstance.show();
32          }
33      });
34
35      const inputGitlabInstance = vscode.window.createInputBox();
36      inputGitlabInstance.placeholder =
37      "Please Enter Your Gitlab Instance [e.g. https://gitlab.com]";
38      inputGitlabInstance.onDidChangeValue((tokenInput) => {
39          gitlabInstance = tokenInput;
40      });
41      inputGitlabInstance.onDidAccept(() => {
42          gitlabInstance = gitlabInstance.length ? gitlabInstance :
            ↪ GitLab_SaaS_Base_URL;
43          inputGitlabInstance.hide();
44          inputPersonalAuthToken.show();
45      });
46      inputGitlabInstance.show();
47  }
```

4.2 publishing on marketplace.visualstudio.com , current progress

```yaml
name: Publish Extension

on:
  push:
    tags:
      - 'v*'
jobs:
  publish:
    name: VSCode Marketplace Publishing
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: 16.x
          cache: npm

      - name: Install dependencies
        run: npm ci

      - name: publish
        run: npm run deploy
        env:
          VSCE_PAT: ${{ secrets.VSCE_PAT }}
```

Sourcecode 4.1 CI Workflow used to publish VSWorkbench to Visual Studio Marketplace.

The above code snippet illustrates publishing extension automatically to Visual Studio Marketplace. The above workflow run only on tags that start with the character `v`, which stands for version. The CI job runs on Ubuntu Linux, with NodeJS installed. Once the operating system is setup, the repository is checked out and the NPM packages are installed. Thereafter, the NPM script `deploy` is run to package and publish the extension. The script will run the `vscode:prepublish` NPM script in the background, which will in turn run the `package` script.

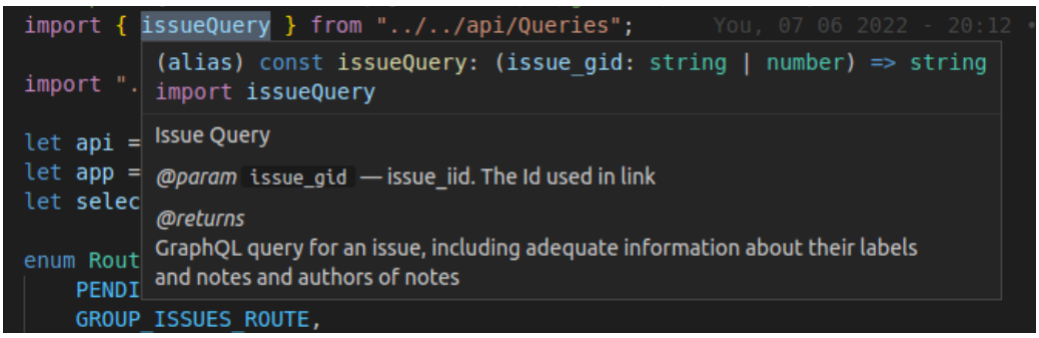


Figure 4.2 Documentation on Hover

Afterwards, `vsce`, a cli program developed by Microsft to act as the VSCode extension manager tool [1], will use the PAT injected by the CI workflow to publish the extension in Visual Studio Marketplace. In approximately 10 minutes, the new version of the extension will be available to users in the Marketplace.

```
...
"scripts": {
    "vscode:prepublish": "npm run package",
    "package": "webpack --mode production --devtool
    ↪hidden-source-map",
    "deploy": "vsce publish --no-yarn",
...
```

: `npm run deploy` script

4.3 Documentation, jsdoc, api file

JSDOC is an API documentation generator for JavaScript, that utilizes comments to generate websites, output documentation to JSON format, and add documentation on hover on code editors and IDEs like VSCode.

JSDOC's syntax allows the developer to quickly add documentation in the form of comments with specific keywords that emphasize important aspects of code, such as return value, parameters and their qualities, or references to other documented snippets of code that help other developers understand the function of the referencing code snippet.

JSDOC facilitates the steps of maintaining, refactoring and extending software.

```
/**
```

[1] See vsce on GitHub

```
 * Issues Query
 * @param isGroup describes whether the query targets a group or
 ↪a project
 * @param fullpath full path of group or project, i.e.

 * 'GitLab-org' or  'GitLab-org/GitLab-foss'
 * @returns Issues of the specified project with adequate
 ↪information about them
 */
export const issuesQuery = (isGroup: boolean, fullpath: string):
↪string =>
    ...
```

The above code snippet showcases a function along with its JSDOC comments.

It will be visualized in VSCode to developers at hover will be represented in a similiar fashion on the documentation website that is wrapped with Docusaurus.

The NPM package `jsdoc-to-markdown` is a package that enables the developer to extract jsdoc comments into markdown files. It takes a minimum of one paramater, specifying the files from which the documentation is to be extracted.
To automate documentation and build of the documentation website, the NPM package `jsdoc-to-markdown` was utilized, wrapped in an NPM script that extracts jsdoc documentation from the specified files into a markdown file.

This markdown file will then be streamed into the docusaurus build pipeline, where it will be rendered in its own tab. Docusaurus, built on react, will build HTML files out of the markdown documents available and create a client side rendered web app.

The architecture of Docusaurus, primarely relying on client side rendering enabled by react and CI infrastructure enabled by GitLab CI lowers overhead that accompanies the task of documenting software while simultaneously building it.

4.4 Design Patterns Used todo rethink title

As VSCode is built with web technologies, and JS being a slower language, VSCode has notoriously slower start up times compared to other code editors. Additionally, VSCode has trouble rendering big files, as opposed to C/C++ based code editors like sublime text.

Therefore, it is vital to make smart decisions when possible in order to not slow down the user's VSCode instance, or worse the user's operating system.

Figure 4.3 GitLab Workflow Startup Overhead

To lower memory expense, the API was implemented using the singleton design pattern.

```
public static get Instance() {
        if (!this.instance) {
            this.instance = new Api();
        }
        return Api.instance;
    }
```

The singleton will be instantiated once for each running context, namely the extension (parent) context, and the three child contexts representing the webviews and the webview views.

This contributes to minimizing resources consumed in total by an already resource heavy application that is VSCode.

```
import { Api } from "../../api";
let api = Api.Instance;
```

Using vanilla Typescript has also contributed to a smaller bundle size for the extension, reducing the lines of code that will be regularly run.

A good contrast is the GitLab Workflow extension by GitLab. GitLab Workflow uses Vue, a frontend framwork, to render its webviews and webview views. This results in more, sometimes unneccesary, code to be packaged.

The NPM package for Vue, version 2.6.14, is 2.97 MB unpackaged, which is probably the biggest suspect for the relatively big extension size. VSWorkbench on the other hand is around ~90 kB, which is substantially more compact, resulting in a smoother experience for the end user.

Ultimately, another optimization done was opting for web native, unicode emoji instead of images in either png, svg or jpeg formats. The main benefit that unicode emoji bring is that they are universally available, provided by the operating system, thus they do not have to be packaged, as what's included is merely a reference, while the responsibility falls on the operating system to provide them.

The two figures below show the big difference in resources overhead the two extensions GitLab Workflow and VSWorkbench bring. While GitLab Workflow needs 87ms for activation, which inevitably translate in an 87ms slower VSCode startup time, VSWorkbench needs only 10ms, which makes it as fast as some of the native VSCode extensions.

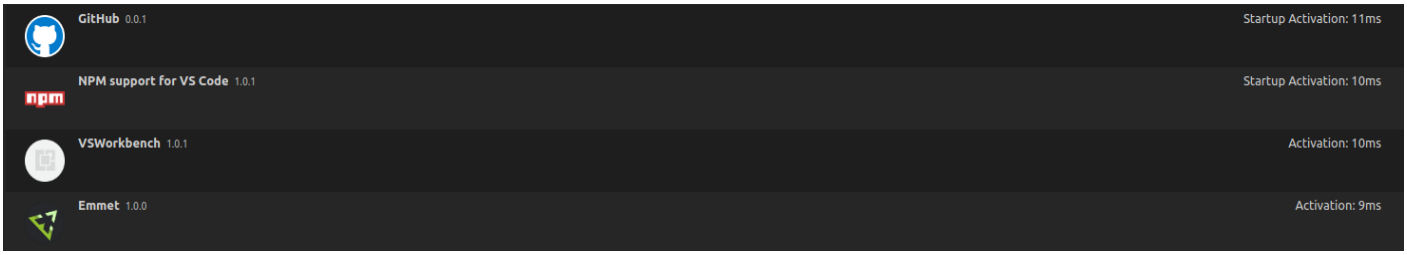


Figure 4.4 VSWorkbench Startup Performace

The performance data mentioned in this section was generated on a machine with the following technical specificationsOS: linux(5.15.0-46-generic) CPUs: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz(12 x 3500) Memory(System): 31.02 GB(13.82GB free)

5 Future Goals

As a famous saying goes, software is never finished, it is only released.

Therefore, further development to integrate more platforms into developers' primary workbench and to improve the already existing tools offered in this toolbox will continue.

5.1 Development Targeting Platforms

VSWorkbench is not intended to target only GitLab. Lots of other software tools used by developers make great candidates for further development. Therefore it is essential to make VSWorkbench extensible in order to support more platforms and tools in the future.

A contributor who intends to extend VSWorkbench can either extend it directly, or through a standalone module that has to be installed seperately. In case the contributor wants to integrate his plugin into VSWorkbench, trunk based development[1] will be used to decouple the development process and smoothen an eventual merge between the core VSWorkbench and the plugin to be integrated. Once the plugin is ready to be integrated, a pull request will be initiated to merge the two pieces of software together. To standardize this process, a simple pull request template was created, inspired by templates already used in development. The template will evolve in case the need emerges.

```
1 Describe here what is this PR about and what we are achieving
  ↪ merging this.
2
3 Thank you for your contribution!
4 Before submitting this PR, please make sure:
5
6 - [ ] Your code builds clean without any errors or warnings
7 - [ ] You have made the needed changes to the docs
8 - [ ] You have written a description of what is the purpose of
  ↪ this pull request above
```

Sourcecode 5.1 Pull Request Markdown Template

Additionally, the contributor can opt for a standalone plugin that will. To facilitate this process, an API will be exposed that will register the new functionalities through VSWorkbench. \begin{lstlisting}[caption={registerPlugin API exposed by VSWorkbench}]

[1] See guide

export function registerPlugin(plugin: {TreeViews: vscode.TreeView[], Commands: {description: string, callback: (...args: any[]) => any, thisArg?: any}[]}){ for(const treeView of plugin.TreeViews) Context.subscriptions.push(treeView); for(const command of plugin.Commands) vscode.commands.registerCommand(command.description, command.callback); } \end{lstlisting

5.2 Documentation

To assist contributions, documentation has to be improved. The goal is to provide plugin examples and tutorials on the current documentation website[2] to encourage platform targeted contributions and complete code documentation in the form of JSDOC comments.

5.3 Testing

In order to reduce errors, tests have to be written and integrated into the already existing CI pipelines. This will be essential to speed developement and automate deployment and publishment of new contributions to users.

[2]See VSWorkbench Docs

List of Figures

List of Sourcecodes

List of Tables

Symbols

Abbreviations

Acronyms

CLK Clock *see* SCL & SCK

SCK Serial Clock *see* SCL & CLK

SCL Serial Clock Line *see* SCK & CLK

Glossary

Rekursion

see Rekursion

Bibliography

[1] Berners-Lee, Tim. *Hypertext Markup Language - 2.0*. https://datatracker.ietf.org/doc/html/rfc1866. IETF RFC 1866. Nov. 1995.

[2] Byron, Lee. *GraphQL: A data query language*. Sept. 2015. URL: `https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/`.

[3] Csikszentmihalyi, M. *Beyond boredom and anxiety*. Jossey-Bass, 2000.

[5] Dangoor, Kevin. *What Server Side JavaScript needs*. CommonJS Announcement. Jan. 2009. URL: https://www.blueskyonmars.com/2009/01/29/what-server-side-javascript-needs/.

[6] EvanLi. *Top 100 Most Starred Repositories on GitHub*. Aug. 2022. URL: https://github.com/EvanLi/Github-Ranking/blob/47ec47b28f64da66f2dcfcab9b3f791c19af6a7a/Top100/Top-100-stars.md.

[7] Fielding, R. et al. *Hypertext Transfer Protocol – HTTP/1.1*. June 1999. URL: https://datatracker.ietf.org/doc/html/rfc2616.

[8] Gamma, Erich. *Hello Code*. Nov. 2015. URL: https://github.com/microsoft/vscode/tree/8f35cc4768393b25468416829e980d7550619fb1.

[9] Hakon Wium Lie, Bert Bos. *Cascading Style Sheets, level 1*. https://www.w3.org/TR/REC-CSS1-961217. Dec. 1996.

[10] Inc., Stack Exchange. *2021 Developer Survey*. TypeScript appearing in charts as the 3rd most loved technology by developers in 2021. Aug. 2021. URL: https://insights.stackoverflow.com/survey/2021#technology-most-loved-dreaded-and-wanted.

[11] Inc., Stack Exchange. *2022 Developer Survey*. TypeScript appearing in charts as the 4rth most loved technology by developers in 2022. June 2022. URL: https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted.

[12] Robin, Adam. *initial commit*. Mar. 2013. URL: https://github.com/electron/electron/tree/e451d9212179197b88abeb752602de3859bb1765.

[13] Rulifson, Jeff. *DEL*. June 1969. URL: https://datatracker.ietf.org/doc/html/rfc5.

[14] Shakeri, Zahra et al. *Task Interruption in Software Development Projects*. May 2018. URL: https://arxiv.org/pdf/1805.05508.pdf.

[15] Thompson, Ken. *The Art of Unix Programming*. URL: https://www.linuxtopia.org/online_books/programming_books/art_of_unix_programming/ch01s06.html.

Further Reading

[4] D. Hardt, Ed. and Microsoft. *The OAuth 2.0 Authorization Framework*. Oct. 2012. URL: https://datatracker.ietf.org/doc/html/rfc6749.