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

Supervisor Dr. Holger Arndt

First Reviewer Dr. Holger Arndt

Second Reviewer Dr. Marcel Schweitzer

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

Entwickler müssen oft eine große Menge an Informationen in ihrem Kurzzeitgedächtnis behalten, um Code effektiv zu navigieren und in der Lage zu sein, präzise Code-Änderungen vorzunehmen, die sich gut in dem bereits vorhandenen Code einfügen. Softwareentwickler müssen jedoch häufig eine Vielzahl von Tools verwenden und in kurzen Abständen zwischen ihnen wechseln. Dies verursacht eine Unterbrechung, die sich nachteilig auf ihre Produktivität auswirken.

VSWorkbench ist ein Tool, das darauf abzielt, die Dauer solcher Unterbrechungen zu verkürzen, indem es zwei zentrale Softwareentwicklungstools zusammenführt, Visual Studio Code und GitLab, näher zusammenbringt, um die Produktivität der Softwareentwickler zu erhöhen.

Abstract

Developers often need to keep a big amount of information in their short time memory in order to navigate code effectively and be able to introduce precise code changes that integrate well with the already existing code. However; software developers' often need to use a diverse set of tools, switching between them at small intervals. This causes an interruption that is detrimental to their productivity.

VSWorkbench is a tool that aims to shorten the length of such interruptions by bringing two central software development tools, Visual Studio Code and GitLab, closer together as a means of increasing the throughput of software developers.

Contents

1 Introduction

Software development saw most of its early application in big governmentally funded projects such as the NASA Apollo 11 and the ARPANET. In these settings, the budget was huge[16], which translates to the financial ability to hiring train new software developers. As Software found its way into commercialization and was made generally available to the public through the Internet, the demand for software developers and software developers soared in a way that made it hard to keep up with demand. This great demand and meager supply of software developers made it a highly vital task to increase the efficiency of the very few software developers available. Hence developer tools.

Developer tools aim to assist developers with all various, enabling developers to cooperate on a higher scale (communication software, e.g. teams, zoom, email, collaboration tools for technical, e.g. Jira, Trello, Atlassian Confluence, collaboration tools for technical userss that touch on many different subjects, e.g. Git, GitLab, GitHub, time saving tools made to accelerate software development, e.g. CI/CD, GitLab CI, GitHub Actions, error reduction tools, e.g. automated testing frameworks like Cypress and Playwright).

These developer tools are essential in the work life of a software developer. Use them fluently and efficiently, and your productivity will increase [14].

2 Problem and Goal

2.1 Effects 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 terms, in order to get through their tasks efficiently.

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

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 [20].

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

VSWorkbench is an attempt to minimize these interruptions in the context of developers who utilize GitLab in their daily life. VSWorkbench strives to reduce time needed to reach a GitLab project, search for a GitLab issue or access GitLab CI/CD information by bringing the most used functionalities of GitLab closer to where the developer is usually to be found when they is in their 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 that is used 80% of the time and present them in a familiar manner that is not much distinguishable from the native web app GitLab experience, with the end goal being minimizing interruptions and hightening developer's productivity.

3 Fundamentals

To implement VSWorkbench, web content is represented in HTMl (3.1), styled in CSS (3.2) and relies on JS (3.3)/TS (3.4) to communicated to VSCode for display through the VSCode API (4.1). VSWorkbench relies on the GitLab server (3.10) (3.9), utilizing Rest (3.6) and GraphQL (3.6) endpoints for data exchange (3.5). In case authorization (3.8) is required, GitLab issued PATs are used to confirm identity and check priviliges. VSWorkbench is then bundled and packaged using webpack (3.11), a module bundler. In the phase of development, NPM (3.13) package manager was used to run workflows, such as packaging the extension using webpack. To promote open source contributions, Docusaurus (3.12) will be used to host code documentation and contribution tutorials.

3.1 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 domains, 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.

3.2 CSS

Cascading Style Sheets (CSS) is a style sheet mechanism that allows web page authors and readers to attach style [11]. Using CSS, a developer can, among other things, specify fonts, colors and spacing. The latest CSS standard is CSS3, put forward by the W3C [11]. It has also been standardized in 1996 when the W3C released its first standard. CSS's progress and future development is managed collaboratively a set of companies[2] under the umbrella of the W3C.

[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/.

[2] See (CSS Working Group Members)[https://www.w3.org/Style/CSS/members]

3.3 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 Document Object Model 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, and object relational mappers [5]. The Javascript variant used in this project is EcmaScript.

3.4 Typescript

Typecript is a statically typed subset of JS, developed and maintained by Microsoft. 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 [12][13]. Typescript currently has approximately 82.9k stars on GitHub, making it the 48th most starred repository on GitHub[6].

3.5 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 [19]. 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 about the business logic. 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. The client then takes care of building a user friendly interface to hold the information given back by the server.

3.6 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 standard for how web services should communicate, introducing constraints that would define behaviour and improve performance of web services [8].

```
1  //#region projects
2  archiveProject(project_id: number): Promise<AxiosResponse> {
3      return
       ↪ Api.Instance.api.post(`v4/projects/${project_id}/archive`);
4  }
5  unArchiveProject(project_id: number): Promise<AxiosResponse> {
6      return
       ↪ Api.Instance.api.post(`v4/projects/${project_id}/unarchive`);
7  }
8  getProjectIssueBoards(project_id: number): Promise<AxiosResponse>
   ↪ {
9      return
       ↪ Api.instance.api.get(`v4/projects/${project_id}/boards`);
10 }
11 getProjectIssues(project_id: number): Promise<AxiosResponse> {
12     return
       ↪ Api.instance.api.get(`v4/projects/${project_id}/issues`);
13 }
```

Source Code 3.1 Examples of API Requests

GraphQL is a technology that defines a way to query data from the server, enabling the developer to get exactly the data they needs, nothing more and nothing less [2]. GraphQL is a project released by developed by Facebook and released to the open source community in 2015.

Figure 3.1 GraphQL Query and Result

Since then, its further development has been a collaboration between Facebook and volunteers from the open source community.

GraphQL enables a frontend developer, who is mainly responsible for the visual elements of a website, to describe the data that they requires. This way, the developer gets exactly the data they 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 specification[7]. It features a simple intuitive syntax.

3.6.1 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 blog sites or advertisement served by an Advertising Network.

Iframes are used by Visual Studio Code (VSCode) to implement the VSCode Webview and Webview views APIs [15]. This enables the developer to contribute a web page or web app to VSCode through their extension.

3.7 VSCode and ElectronJS

ElectronJS was started in 2013 [18] to enable web developer to build cross platform desktop applications 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.

VSCode is an open source cross platform language agnostic extendible code editor started in 2015 by Microsoft [9]. It was intended to be part of Microsoft's trend towards open source. It is 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 and Emacs users.

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

3.8 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 a service requiring an authentication or authorization workflow. 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 Visual Studio Marketplace, it can reach a max of 180 days. 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 [3]. The GitLab API, in its self-hosted version as well as the Software As A Service 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:

```
1 {
2     "message": "401 Unauthorized"
```

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

```
3 }
```

3.9 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 various 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. An example of this is deploying a version that can be used internally for testing purposes, polishing this version and later publishing it public for use by the end users.

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 proprietary and in part paid [4].

Git was designed to be fast and efficient. As it follows the unix philosophy of programming [17], its functionality can also be extended. A good example of that is Git Flow [5].

3.10 GitLab

As Git started to gain traction, multiple efforts to introduce a graphical user interface where made. The most successful were web app 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 CI/CD 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 CI/CD tooling and support, starting in 2012[6]. Over the course of the years, GitLab developed and integrated more DevOps tools, enabling developers to publish their software packages through GitLab [7] and to speed the

[4] See BitKeeper, StarTeam, Rational Synergy, and Vault VCS
[5] see Git Flow
[6] See blog post on the matter
[7] e.g. nuget packages for dotnet, NPM packages for node

development lifecycle [8]. 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 [9].

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 philosophy that inspired the development of VSWorkbench.

GitLab exposes REST and GraphQL APIs[10] 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 self-hosted version of GitLab, or are available to premium accounts. Additionally, some endpoints are only invokable on a self-hosted instance, e.g. the createGroup API Endpoint [11].

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

Source Code 3.2 Example of the usage of the GitLab API of the University of Wuppertal

3.11 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 build per the config of the module bundler.

A famous module bundler is Webpack. Webpack was started in 2014 by Tobias Koppers in order to simplify bundling and compilation workflow of web projects based on HTML, CSS and JS. Webpack is used in VSWorkbench to seperately build the extension and the three vanilla Typescript apps in a way that facilitates them working as one unit.

[8] See https://about.gitlab.com/stages-devops-lifecycle/
[9] See Service Desk Documentation
[10] See GitLab Documentation
[11] See documentation reference

3.12 Docusaurus and GitLab CI.

VSWorkbench aims to be open source 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 [12] 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, eliminating 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 and GitLab CI.

The process is as follows:

1. The developer defines a GitLab CI job[13], named `pages`, in the repository's `.gitLab-ci.yml` file.

2. When a commit is pushed into the remote Git repository on the GitLab server, the `pages` job automatically 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 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 end user navigates to the url to which the artifact is mapped, the GitLab servers 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.

[12] See Docusaurus
[13] Jobs are the smallest CI units consisting of commands and operations. Multiple CI jobs make up a CI pipeline

3.13 Node Package Manager

The Node Package Manager (short: NPM) 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. Propelled popularity, it became the default package manager for the Javascript runtime environment Node.js. NPM 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 with 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 for meta data 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 in case it is open-sourced. An extension publisher can also define different tags that will help users in Visual Studio Marketplace to search and find their package.

```
1    "scripts": {
2        "vscode:prepublish": "npm run package",
3        "ts": "npm run esbuild-base -- --minify",
4        "compile": "webpack",
5        "watch": "webpack --watch",
6        "package": "webpack --mode production --devtool
         ↪hidden-source-map",
7        "compile-tests": "tsc -p . --outDir out",
8        "watch-tests": "tsc -p . -w --outDir out",
9        "pretest": "npm run compile-tests && npm run compile &&
         ↪npm run lint",
10       "lint": "eslint src --ext ts",
11       "test": "node ./out/test/runTest.js",
12       "deploy": "vsce publish --no-yarn",
13       "postinstall": "cd docs && npm install && cd ..",
14   },
```

The code snippet 3.13 showcases an excerpt from the `package.json` file used in developing VSWorkbench. It defines commands to test, compile and package 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, decoupling the project from a single developer's local environment.

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

```
1 npm ci;
```

in their command line interface.

4 Implementation

VSWorkbench focuses on being a faster way to do the limited subset of widely used functionalities of GitLab. Therefore, it is essential to keep in mind that not all GitLab web UI functionalities are implemented in GitLab. VSWorkbench otherwise offers quick access to the correct part of GitLab UI. This enables users to quickly view a project or a groups settings, an issue, a CI pipeline or a job.

- Preview and management of groups and projects: Users can clone, create, and navigate to the GitLab web UI page responsible for a namespace or a project. Users can also navigate to the settings of a namespace or a project.
- Quick access to issues: Users can browse issues, quickly navigate to the corresponding GitLab URL, add and delete comments.
- Quick access to CI: Users can access key GitLab CI information such as pipelines, stages, and runner logs. From there, they can quickly access GitLab UI for a more comprehensive view. Users can also retry a job or delete it.
- Quick access to Snippets and Wiki: Users can access to GitLab projects' wikis and snippets through VSWorkbench.

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. To utilized it, a class (as seen in code snippet 4.1) that inherits and implements the `vscode.TreeDataProvider` class and its functions has to be defined. An object of this newly created class has to be created. This object will be used to communicate to the VSCode UI and embed visual elements into it. The developer also has to define an entry for the implemented class in the `package.json` file(see 4.2).

```
export class GroupTreeDataProvider implements
↪ vscode.TreeDataProvider<GroupNode>,
↪ vscode.TreeDragAndDropController<GroupNode> {
    dropMimeTypes = ["application/vnd.code.tree.groupView"];
    dragMimeTypes = ["application/vnd.code.tree.groupView"];

    onDidChange?: vscode.Event<vscode.Uri>;
    private _onDidChangeTreeData: vscode.EventEmitter<GroupNode |
    ↪ undefined | void> = new vscode.EventEmitter<GroupNode |
    ↪ undefined | void>();
    readonly onDidChangeTreeData: vscode.Event<GroupNode |
    ↪ undefined | void> = this._onDidChangeTreeData.event;
    constructor(context: vscode.ExtensionContext) {
        newAuthentication.event(this.refresh, this);
        const groupModel = new GroupModel();

        const view = vscode.window.createTreeView("groupView", {
            treeDataProvider: this,
            canSelectMany: false,
            dragAndDropController: this,
            showCollapseAll: true,
        });
        ...
```

Source Code 4.1 The GroupTreeDataProvidor Class which inherits from the vscode.TreeDataProvidor class

```
{
...
"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"
            },
            ...
            ]
        ...
        }
}
}
```

Source Code 4.2 GroupTreeDataProvidor Declaration in ‘package.json‘ file under the Id "groupView"

The `contributes` entry in the `package.json` file declaratively defines the visual and logical functionalities that the extension adds to VSCode. 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, e.g. the `groupView` as seen in code snippet 4.2. 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.

The Drap and Drop API is used by implementing two functions for the drag and drop functionality in the `GroupTreeDataProvider` that are inhereted from the `vscode.TreeDragAndDropController` class.

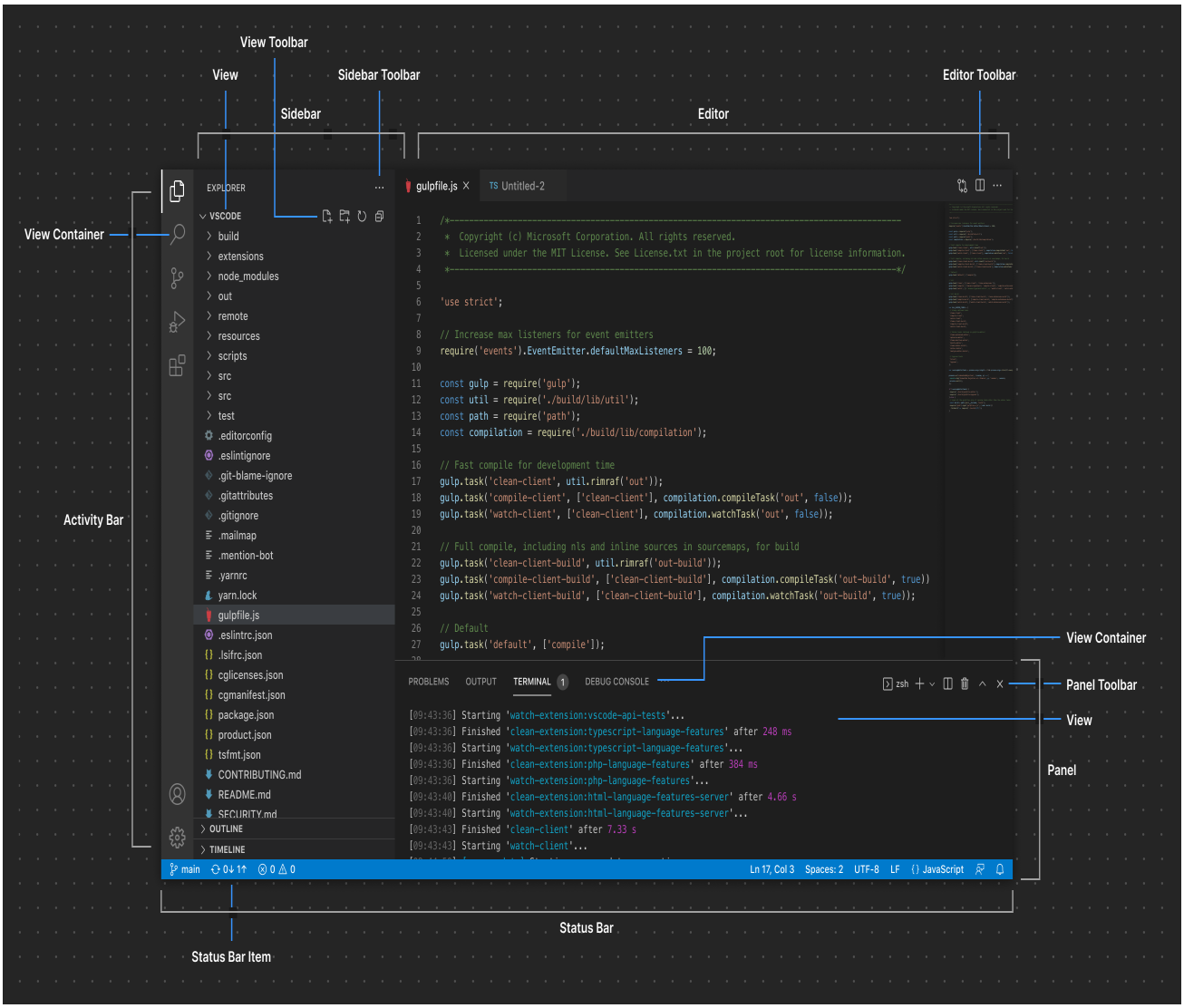

Figure 4.1 VSCode Extension Anatomy

```
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        }
8    }
9    public async handleDrop(target: GroupNode | undefined,
     ↪ sources: vscode.DataTransfer,
10
11    _token: vscode.CancellationToken): Promise<void> {
12       const transferItem =
         ↪ sources.get("application/vnd.code.tree.groupView");
13       ...
```

The `handleDrag` function takes two GroupNode objects as a parameter in addition to a cancellation token parameter, required for cancelling the action cooperatively between the multiple threads enabling the functionality. 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. User namespaces are usually denoted by https://GitLab.com/[username], and they cannot be moved or deleted. If the node is not a user namepsace, i.e. if it is a group[1] or a project node[2], 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 towards the bottom of the view. 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 namespace into projects. Moreover,

[1]See GitLab's (main group)[https://gitlab.com/gitlab-org]
[2]See the (GitLab project repository)[https://gitlab.com/gitlab-org/gitlab]

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 depending on the source and target nodes are defined in the `api.ts` file and have the following declarations:

```
1
2  transferGroup(id: number, group_id?: number):
   ↪ Promise<AxiosResponse> {
3      return Api.instance.api.post(`v4/groups/${id}/transfer`,
4
5      group_id ? { group_id } : null);
6  }
7  transferProjectToGroup(group_id: number, project_id: number):
8
9  Promise<AxiosResponse> {
10     return Api.instance.api.put
11
12     (`v4/projects/${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 converting subgroups to 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, demonstrated in figure 4.1, 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 must be instantiated by utilizing the {vscode.window.createWebviewPanel} function.

```
1  let panel = vscode.window.createWebviewPanel(this.viewType,
   ↪ `${name} | ${ViewEvents[type]}`, vscode.ViewColumn.One, {
```

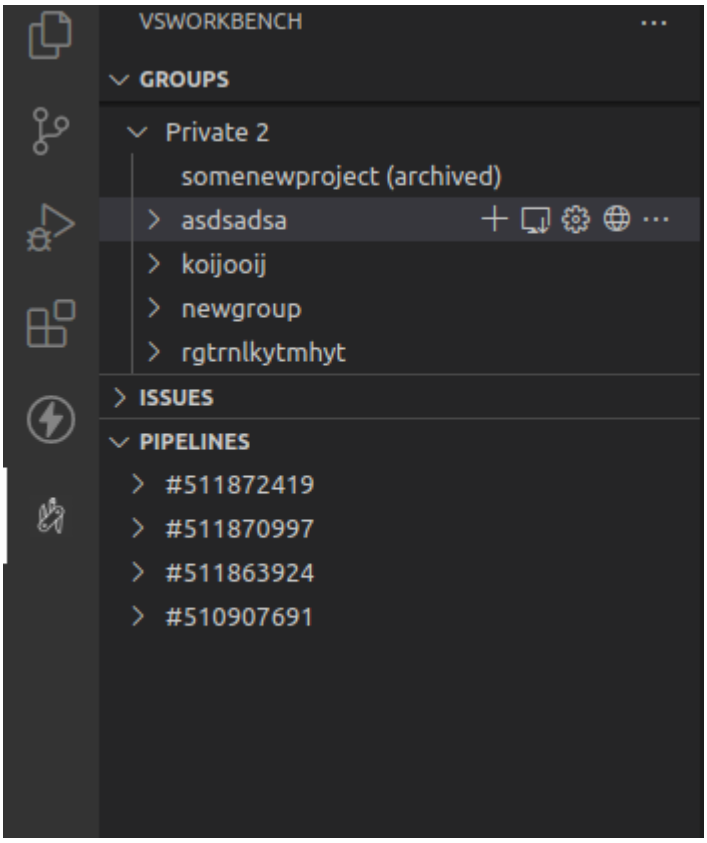

Figure 4.2 The groupView Tree View in VSWorkbench

```
        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 following 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">
            <title>GitLab CI | Editor</title>
        </head>
        <body>
        <div id="app"></div>
            <script src="${scriptUri}"></script>
        </body>
        <script >window.vscode = acquireVsCodeApi();</script>
        </HTML>`;
}
```

```
import * as vscode from "vscode";
```

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

that is hardcoded in the extension can access the
Javascript file designated to it. Lastly, the function assigns the VSCode API to a variable named `window.vscode` that will later be used for communication between the Webview embedded HTML document and the extension context.

To prevent any confusion, the VSCode API that is assigned to `window.vscode` in snippet 4.1.2 refers to an API that can merely send messages back and forth between the extension context and the contexts of the webviews and webview views, not the API exposed by vscode to enable the extension to communicate with VSCode and make use of its components, represented by `vscode` in snippet 4.1.2.

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 implemented. 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.

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

```
3
4       vscode.window.registerWebviewViewProvider(this.viewType, this,
5       { webviewOptions: { retainContextWhenHidden: true } });
6       this._extensionUri = context.extensionUri;
7   }
```

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.

4.1.4 ExtensionContext.globalState

VSCode also exposes an API to allow the user to make use of the storage managed by VSCode. Using this API VSWorkbench hands over the responsibility of storing the GitLab instance URL and the authentication token of the user to VSCode.

```
1   function initStorage(context: vscode.ExtensionContext) {
2       context.globalState.setKeysForSync([AUTH_TOKEN_KEY]);
3       context.globalState.setKeysForSync([GITLAB_INSTANCE_KEY]);
4   }
```

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

The code snippet shown in reference 4.1.4 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 relieves the developer from reimplementing the implemented.

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 diverse 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 them.

```
1   async cloneNameSpace(): Promise<any> {
2       if (this.contextValue === "project") {
3           return vscode.window.showErrorMessage("Entity Chosen
            ↪ is not a Namespace!");
4       }
5       let res = await api.getProjects(this.contextValue ===
        ↪ "group", this.node_id);
6       let path: vscode.Uri[] | undefined = await
        ↪ vscode.window.showOpenDialog({
7           canSelectFiles: false,
8           canSelectFolders: true,
9           canSelectMany: false,
10      });
11      if (path === undefined || path[0] === undefined) {
12          return vscode.window.showErrorMessage
13          ("Please choose a folder to clone into");
14      }
15      res.data.forEach(async (project: any) => {
16          if (!project.archived) {
17              await cloneFromGitLab(project.http_url_to_repo,
                ↪ path![0].path);
18          }
19      });
20      return true;
21  }
```

The code snippet seen in 4.1.5 demonstrates the function used by VSWorkbench to clone groups. VSWorkbench first checks that the node, on which the function was called / the button

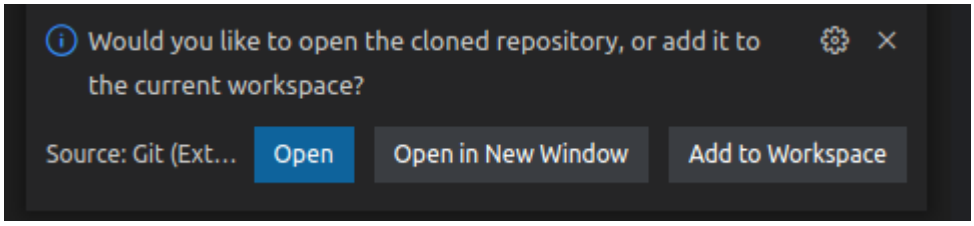

Figure 4.3 VSCode Git Success Message

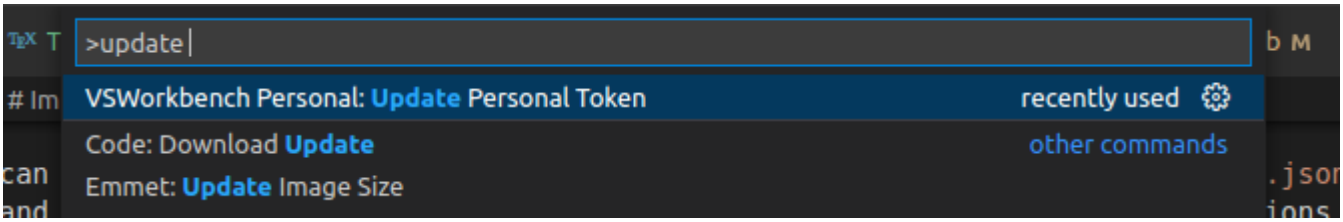

Figure 4.4 A VSWorkbench Command Contributed to VSCode

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 an ElectronJS dialog asking the user to choose a directory on their PC, onto which the projects will be cloned. Once a folder is chosen, VSWorkbench proceeds to clone the active, i.e., non archived projects on which development is ongoing, one by one. After cloning the projects successfully, the VSCode Git extension will display a success message to the user.

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

The snippet of code 4.1.5 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.

Alternative to executing VSCode commands in code, the developer can also define their own commands. Common uses for this is for calls through the `package.json` file, enabling users to call the commands directly through the VSCode command palette as seen in figure 4.4, 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.

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

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

Source Code 4.3 Registering a funtion or method as an extension command.

```

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) {
```

```
18              gitlabInstance += "/api/";
19          }
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  }
```

Source Code 4.4 GlobalFunctions.settings Function Registered.

4.2 Publishing on Visual Studio Marketplace

```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 }}
```

Source Code 4.5 CI Workflow used to publish VSWorkbench to Visual Studio Marketplace.

The code snippet seen in 4.5 illustrates publishing extension automatically to Visual Studio Marketplace. The workflow described in 4.5 runs 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 set up, 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. Afterwards, `vsce`, a Command Line Interface (CLI) CLI program developed by Microsft to act as the VSCode extension manager tool [3], will use the PAT injected by the CI workflow

[3] See vsce on GitHub

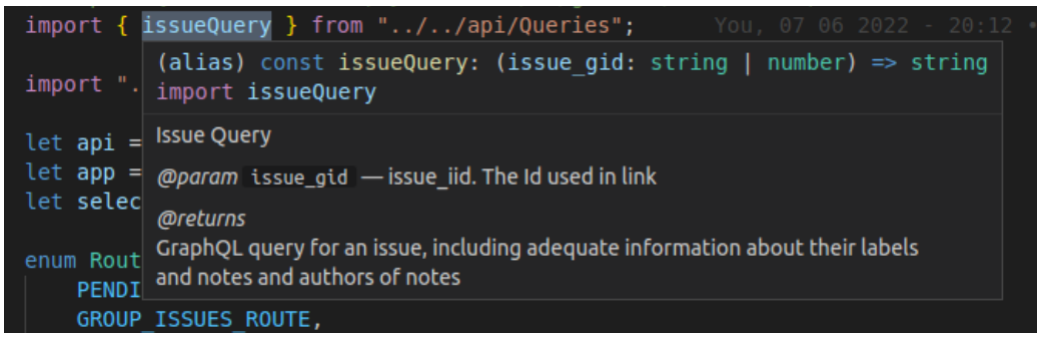

Figure 4.5 Documentation on Hover

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",
...
```

Source Code 4.6 'npm run deploy' script

4.3 Online Documentation and JSDOC Code Documentation

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.

```
/**
 * Issues Query
```

```
3   * @param isGroup describes whether the query targets a group or
    ↪ a project
4   * @param fullpath full path of group or project, i.e.
5
6   * 'GitLab-org' or  'GitLab-org/GitLab-foss'
7   * @returns Issues of the specified project with adequate
    ↪ information about them
8   */
9 export const issuesQuery = (isGroup: boolean, fullpath: string):
  ↪ string =>
10      ...
```

The code snippet 4.3 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 Performance and Size Optimizations

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.

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;
    }
```

Source Code 4.7 API Singleton Object Instantiatabion.

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;
```

Source Code 4.8 API Singleton Usage Example

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.

See Table 4.4 for a functionality comparison between GitLab Workflow and VSWorkbench: [4]

Feature	VSWorkbench	GitLab Workflow
Manage GitLab namespaces	X	-
Manage GitLab projects	X	-
View GitLab issues	X	X
Create and review MR	-	X
Validate GitLab CI/CD configuration	-	X
Manage pipelines	X	X
Manage snippets	X	X
Browse a GitLab repository directly	-	X
Auto-complete GitLab CI/CD variables	-	X
Clone a repository	X	-

GitLab Workflow uses Vue, a frontend framwork, to render its webviews and webview views. This results in more, sometimes unneccesary, code to be packaged.

[4] See the GitLab repository for more information

Figure 4.6 GitLab Workflow Startup Overhead

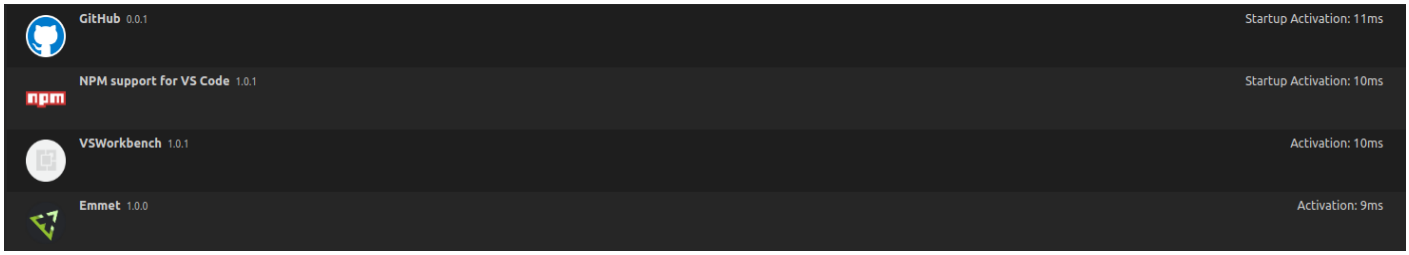

Figure 4.7 VSWorkbench Startup Performance

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 4.7 and 4.6 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.

The performance data mentioned in this section was generated on a machine with the following technical specifications

- OS: linux(5.15.0-46-generic)
- CPUs: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz(12 x 3500)
- Memory(System): 31.02 GB

5 Summary

The goal of this this bachelor's thesis came to light due to the scarcity of developers and the to increase the productivity of already existing developers, as stated in chapter 11.

The implementation of VSWorkbench is heavily influenced by the extensible nature of software surrounding the development of VSWorkbench, VSCode and the internet for instance.

VSWorkbench was built upon the already internet technologies, ranging from the HTML (3.1), CSS (3.2) and Javascript (3.3) standards to CI tools (3.1), Git (3.9) and VSCode (3.7).

Furthermore, due to its nature being an extension, it was build on top of the APIs provided by VSCode, as explained in chapter 44.

Despite being in an early stage of its active development with alot of work planned ahead, as described in chapter 6, VSWorkbench was installed 53 times as of the date of submitting this bachelor's thesis (3.1), according to data provided by Visual Studio Marketplace in figure 5.1.

Ultimately, the initial baby steps taken in this bachelor's thesis and the date provided by Visual Studio Marketplace shown in figure 5.1 underlines that future development, refinement and extension on VSWorkbench is worth undertaking.

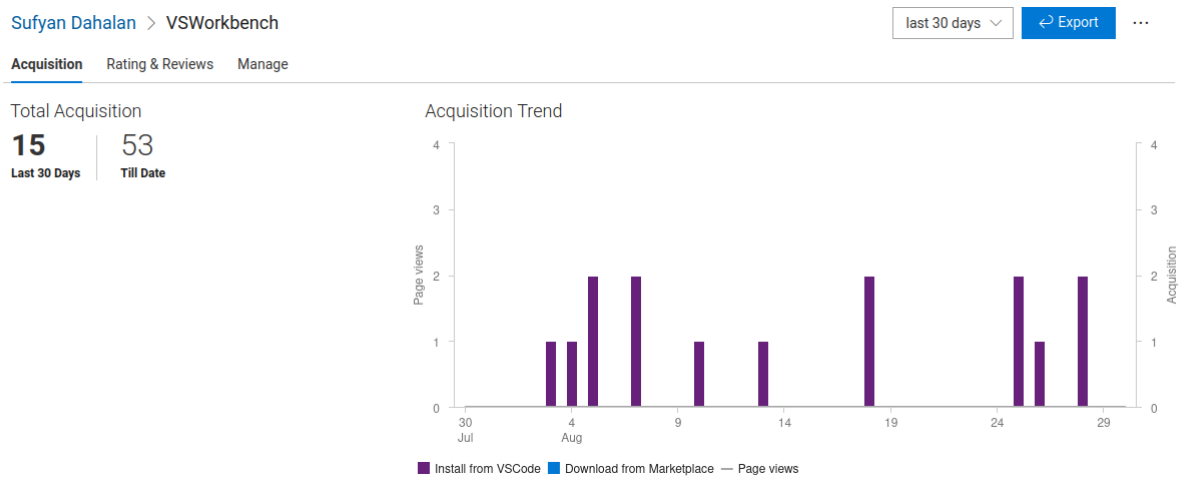

Figure 5.1 User Acquisition

6 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.

6.1 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 plugin 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
```

Source Code 6.1 Pull Request Markdown Template

A further step made to offer guidance for contributors is the `CONTRIBUTION.md` file.

Additionally, the contributor can opt for a standalone plugin that will has to be installed seperately. To facilitate this process, an API will be exposed that will register the new functionalities

[1] See Git trunk based development guide

through VSWorkbench.

```
export function registerPlugin(plugin: {TreeViews:
↪ vscode.TreeView<any>[], 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);
}
```

Source Code 6.2 registerPlugin API exposed by VSWorkbench

6.2 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.

6.3 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 development and automate deployment and publishment of new contributions to users.

[2] See VSWorkbench Docs

List of Figures

List of Source Codes

List of Tables

Symbols

Abbreviations

CLI	Command Line Interface	27
CSS	Cascading Style Sheets	3, 7, 9, 10, 20
VSCode	Visual Studio Code	6, 7, 13, 15, 18, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31

Acronyms

Glossary

Document Object Model

A Programming API for HTML through which HTML can be manipulated. 4

MR

A Merge Request, short MR, is the way a developer checks source code changes into a branch. Benefits of MRs are visualization, collaboration and enhances discussion of incoming code changes. 30

Software As A Service

SAAS is business model in which the users pay for access to a centrally hosted application. An example of this is the GitLab instance available at https://gitlab.com/. 7

WHATWG

The Web Hypertext Application Technology Working Group (short: WHATWG) is a collaborative nonprofit foundation that develops and maintains the HTML living standard. 3

Bibliography

[1] Berners-Lee, Tim. *Hypertext Markup Language - 2.0*. Tech. rep. IETF RFC 1866. Nov. 1995. (Visited on 08/24/2022).

[2] Byron, Lee. *GraphQL: A data query language*. Tech. rep. Sept. 2015. URL: `https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/` (visited on 08/24/2022).

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

[5] Dangoor, Kevin. *What Server Side JavaScript needs*. Tech. rep. CommonJS Announcement. Jan. 2009. URL: `https://www.blueskyonmars.com/2009/01/29/what-server-side-javascript-needs/` (visited on 08/24/2022).

[6] EvanLi. *Top 100 Most Starred Repositories on GitHub*. Tech. rep. Aug. 2022. URL: `https://github.com/EvanLi/Github-Ranking/blob/47ec47b28f64da66f2dcfcab9b3f791c19af6a7a/Top100/Top-100-stars.md` (visited on 08/24/2022).

[7] Fielding, R. et al. *Hypertext Transfer Protocol – HTTP/1.1*. Tech. rep. June 1999. URL: `https://datatracker.ietf.org/doc/html/rfc2616` (visited on 08/24/2022).

[8] Fielding, Roy Thomas. "Architectural styles and the design of network-based software architectures". Publication. University of California, Irvine, 2000. URL: `https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm`.

[9] Gamma, Erich. *Hello Code*. Tech. rep. Nov. 2015. URL: https://github.com/microsoft/vscode/tree/8f35cc4768393b25468416829e980d7550619fb1 (visited on 08/24/2022).

[11] Hakon Wium Lie, Bert Bos. *Cascading Style Sheets, level 1*. Tech. rep. Dec. 1996. (Visited on 08/24/2022).

[12] Inc., Stack Exchange. *2021 Developer Survey*. Tech. rep. 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 (visited on 08/24/2022).

[13] Inc., Stack Exchange. *2022 Developer Survey*. Tech. rep. 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 (visited on 08/24/2022).

[14] Lavazza, Luigi; Morasca, Sandro, and Tosi, Davide. "An Empirical Study on the Factors Affecting Software Development Productivity". In: *E-Informatica Software Engineering Journal* 12 (Feb. 2018), pp. 27–49. DOI: 10.5277/e-Inf180102.

[15] Microsoft. URL: https://code.visualstudio.com/api/get-started/extension-anatomy (visited on 08/24/2022).

[16] Microsoft. URL: https://www.planetary.org/space-policy/cost-of-apollo#project-apollo-costs-summary (visited on 08/29/2022).

[17] Raymond, Eric S. *The Art of UNIX Programming*. Pearson Education, 2003. ISBN: 0131429019.

[18] Robin, Adam. *initial commit*. Tech. rep. Mar. 2013. URL: https://github.com/electron/electron/tree/e451d9212179197b88abeb752602de3859bb1765 (visited on 08/24/2022).

[19] Rulifson, Jeff. *DEL*. Tech. rep. June 1969. URL: https://datatracker.ietf.org/doc/html/rfc5 (visited on 08/24/2022).

[20] Shakeri, Zahra et al. *Task Interruption in Software Development Projects*. Tech. rep. May 2018. URL: https://arxiv.org/pdf/1805.05508.pdf (visited on 08/24/2022).

Further Reading

[4] D. Hardt, Ed. and Microsoft. *The OAuth 2.0 Authorization Framework*. Tech. rep. Oct. 2012. URL: https://datatracker.ietf.org/doc/html/rfc6749 (visited on 08/24/2022).

[10] Gamma, Erich et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0201633612.