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**<br>RAINER-GRUENTER-STR 21<br>42119 WUPPERTAL<br>TELEFON (o2o2) 439 – 1 | FACHBEREICH E<br>ELEKTROTECHNIK / INFORMATIONSTECHNIK / MEDIENTECHNIK<br>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. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

## 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.

# Inhaltsverzeichnis

**1 Problem and Goal** **1**

1.1 Developer Productivity . . . 1

1.2 Affects of Context Switching on Developer Productivity . . . 1

**2 Fundamentals** **3**

2.0.1 Tech Stack . . . 3

2.1 The modern web architecture (web 2.0) . . . 4

2.1.1 Client-Server Architecture . . . 4

2.1.2 HTTP methods, REST, GraphQL . . . 4

2.1.3 iframes . . . 5

2.1.4 vscode and electron js . . . 6

2.1.5 Authentication, Authorization and Personal Access Tokens . . . 6

2.1.6 Version Control Software and Git . . . 7

2.1.7 Gitlab . . . 8

2.1.8 Module Bundlers and Webpack . . . 8

2.1.9 Docusaurus and GitLab CI. . . . 9

2.2 npm, npm scripts . . . 10

**3 Implementation** **12**

3.1 VSCode API . . . 12

3.1.1 Tree View . . . 12

3.1.2 Webview . . . 16

3.1.3 Webview View API . . . 18

3.1.4 Commands . . . 18

3.2 Gitlab REST and GraphQL apis . . . 18

3.3 publishing on marketplace.visualstudio.com , current progress . . . 19

3.4 telemetry? . . . 20

3.5 contexts . . . 20

3.6 Design Patterns Used . . . 20

3.7 write about making it look native, aka resemble gitlab very well (goal) . . . 21

3.8 write about extensibility, PR requests, Opensource, a bit about why github . . . 21

3.9 write about making it small, improving performance and time needed to start up, and how it holds up comapred to gitlab workflow (goal, plus comparison and results section) . . . . . . . . . . 21
3.10 goals moving forward . . . . . . . . . . 22

**Abbildungsverzeichnis** **23**

**Quellcodeverzeichnis** **23**

**Tabellenverzeichnis** **23**

**Symbole** **23**

**Abkürzungen** **23**

**Akronyme** **23**

**Glossar** **24**

**Literatur** **24**

**Weiterführende Literatur** **25**

# 1 Problem and Goal

## 1.1 Developer Productivity

Programming and software development has its deep roots in the scientific communities and largely resided, for the first couple of decades, in the (obscure|dark|somepoeticthing)? rooms of researchers. As programming found its way into commercialization not bound to any hardware product 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 the pipeline fed|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 [teams, zoom, email], collaboration tools for technical and nontechnical staff [jira, trello, atlassian wiki], collaboration tools for technicals that touch on many different subjects [git, gitlab, github], tools made to save time [CICD, gitlab ci, github action], tools made to increase precision and correctness of work output
made by programmers [automated testing frameworks => cypress, playwright], and debugging).

These developer tools are essential in the work life of a programmer. Use them fluently and efficiently, and your output will easily multiply.

## 1.2 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 [Csi00]. 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 [SAK+18].

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/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 is in his 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.

# 2 Fundamentals

### 2.0.1 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. [BL95] 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 is a style sheet mechanism that allows web page authors and readers to attach style [HWL96]. Using CSS, a developer can, among other things, specify fonts, colors and spacing. Latest CSS standard is CSS3, put forward by the W3C [HWL96]. 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 [Dan09]. The Javascript variant used in this project is EcmaScript.

[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/ . #### CSS

#### 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 [Inc21][Inc22]. Typescript currently has approximately 82.9k stars on GitHub, making it the 48th most starred repository on GitHub currently [Eva22].

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

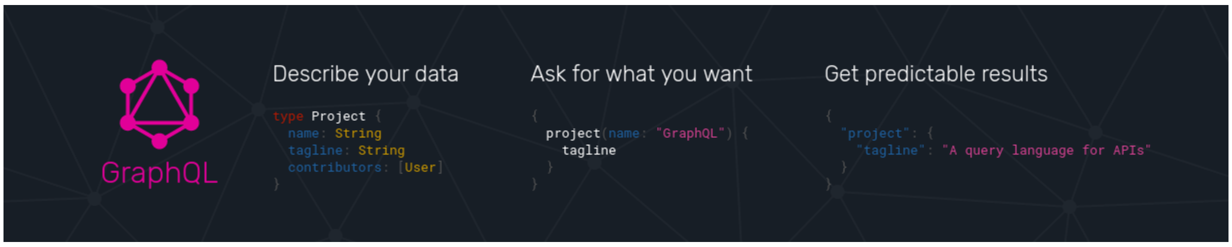
## 2.1 The modern web architecture (web 2.0)

### 2.1.1 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 it's roots in the early ARPANET days in the 1970s where the Stanford researchers worked toward creating interactive programms that function across computer networks [Rul69]. 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 knwo bla bla. 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.

### 2.1.2 HTTP methods, REST, GraphQL

HTTP [FIG+99] 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



```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

**Abbildung 2.1** GraphQL Query and Result

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{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 needs, nothing more and nothing less [Byr15]. 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 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[FIG+99]. It features a simple intuitive syntax

### 2.1.3 iframes

An `iframe`, or an inline frame, is an html tag todo{link somewhere to explain what a tag is?} 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.

### 2.1.4 vscode and electron js

ElectronJS was started in 2013 [Rob13] 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 view into desktop app development. It's wide adoption, inspite of it's lack of performance compared to traditional technologies, speaks for it's success.

Visual Studio Code (short: VSCode) is an open source cross platform language agnostic extendible code editor started in 2015 by microsoft [Gam15]. It was intended to be part of Microsoft's trend towards open source. It was based in it's core on open source/open standard technologies such as html, css, javascript and electronjs, with all developers as end users in mind. It's 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.

### 2.1.5 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?. 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 service, whether self-hosted or the SAAS version, The GitLab API 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 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}

## 2.1.6 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 proprietary and in part paid [3].

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

[2] see `https://docs.gitlab.com/ee/api/index.html`
[3] See BitKeeper, StarTeam, Rational Synergy, and Vault VCS
[4] see Git Flow

### 2.1.7 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/system that is effectively a fully featured toolbox: 1. 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 it's own maintenance and administration overhead. This is essentially the philisophy that inspired the development of VSWorkbench.

### 2.1.8 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 it's 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.

[5] e.g. nuget packages for dotnet, npm packages for node
[6] See Service Desk Documentation

### 2.1.9 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 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 (aka gitlab server), the `pages` job runs on the repository where it is defined. 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 following syntax? : [gitlab username].gitlab.ioa/[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.

## 2.2 npm, npm scripts

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. A package/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 it's 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.

# 3 Implementation

## 3.1 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.

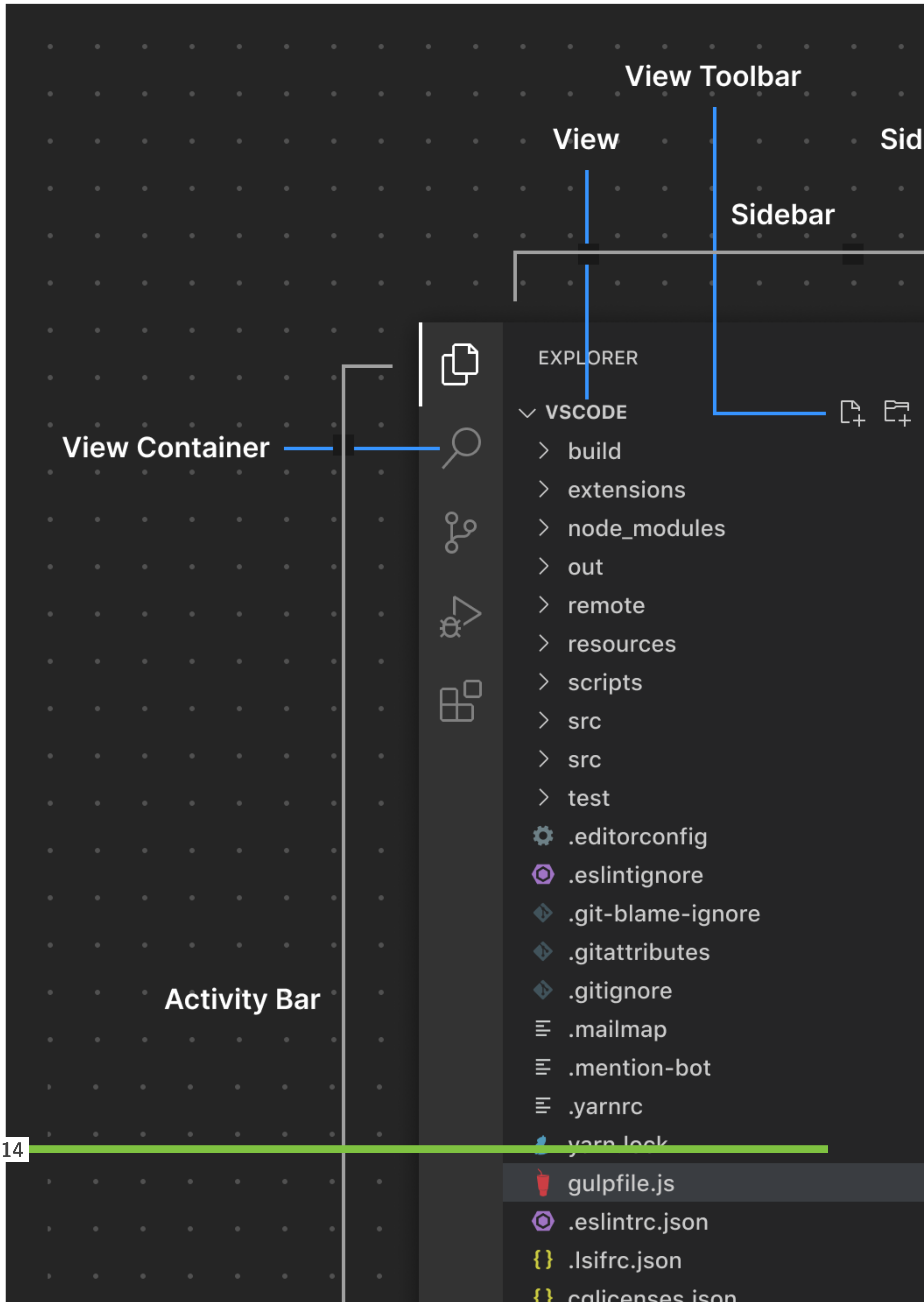
### 3.1.1 Tree View

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 it's 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 explains the anatomy and components of VSCode visually.


View Toolbar
View
Sid
Sidebar
View Container
Activity Bar
EXPLORER
VSCODE
build
extensions
node_modules
out
remote
resources
scripts
src
src
test
.editorconfig
.eslintignore
.git-blame-ignore
.gitattributes
.gitignore
.mailmap
.mention-bot
.yarnrc
yarn.lock
gulpfile.js
.eslintrc.json
.lsifrc.json
cglicenses.json

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.

```
public async handleDrag(source: GroupNode[], treeDataTransfer: vscode.DataTransfer,

  _token: vscode.CancellationToken): Promise<void> {
    if (source[0].contextValue !== "user") {
        treeDataTransfer.set("application/vnd.code.tree.groupView",
         new vscode.DataTransferItem(source));
    }
}
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, it's 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.

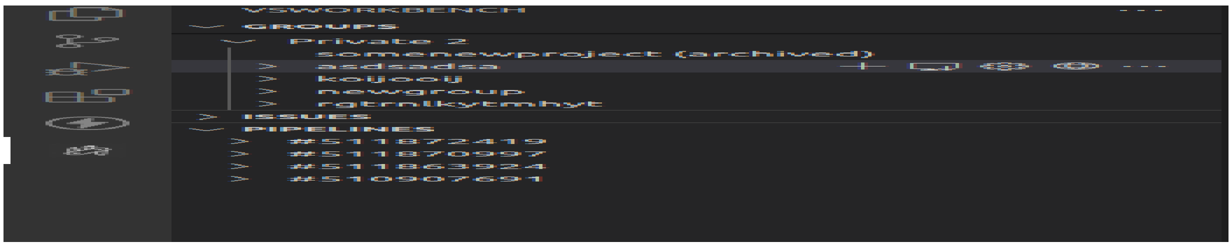


**Abbildung 3.1** The groupView Tree View in VSWorkbench

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);
}
transferProjectToGroup(group_id: number, project_id: number):

Promise<AxiosResponse> {
    return Api.instance.api.put

    (`v4/projects/${project_id}/transfer?namespace=${group_id}`);
}
```

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.

### 3.1.2 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
            enableScripts: true,
            retainContextWhenHidden: true,
        });
```

The function is called with three parameters, namely the view type of the webview, it's 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">
            <title>GitLab CI | Editor</title>
        </head>
        <body>
        <div id="app"></div>
            <script src="${scriptUri}"></script>
        </body>
        <script >window.vscode = acquireVsCodeApi();</script>
        </html>`;
}
```

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 it's 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`.

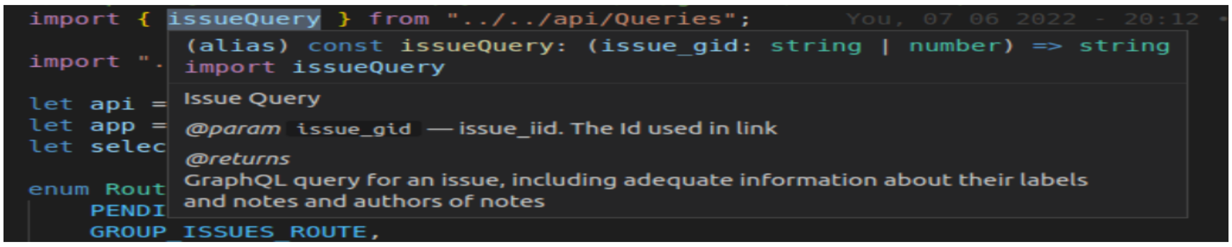
### 3.1.3 Webview View API

### 3.1.4 Commands

tree classes , tree models, drag n drop, webview and webviewview

## 3.2 Gitlab REST and GraphQL apis

GitLab exposes APIs for users



```
import { issueQuery } from "../../api/Queries";        You, 07 06 2022 - 20:12
         (alias) const issueQuery: (issue_gid: string | number) => string
import ". import issueQuery

let api =  Issue Query
let app =  @param issue_gid — issue_iid. The Id used in link
let selec
           @returns
enum Rout  GraphQL query for an issue, including adequate information about their labels
    PENDI  and notes and authors of notes
    GROUP_ISSUES_ROUTE,
```

**Abbildung 3.2** JSDOC Output

## 3.3 publishing on marketplace.visualstudio.com , current progress

vsce publish, how to compiles and packages and link the three apps ot the one parent app ## Documentation, jsdoc, api file

JSDOC is an API documentation generator for JavaScript, that utilizes comments to generate websites, output to JSON format, and add hover support 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 will highlight 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
 * @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, I utilized the npm package `jsdoc-to-markdown` wrapped in an npm script that extracts jsdoc documentation from the specified files into a markdown file. This markdown file will then be fed into the docusaurus build pipeline, where it will be rendered in it's 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.

## 3.4 telemetry?

analyze how much specific functions/services/features are used and say smth about it.

## 3.5 contexts

write something about having 4 contexts running at the same time. (3 vanilla apps, and the extension, therefore requiring 4 api singletons)

## 3.6 Design Patterns Used

Due 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;
    }
```

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;
```

## 3.7 write about making it look native, aka resemble gitlab very well (goal)

## 3.8 write about extensibility, PR requests, Opensource, a bit about why github

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

todo{write about PR template.md or something and contributing.md and so on blz. trunk based development and so on}

## 3.9 write about making it small, improving performance and time needed to start up, and how it holds up comapred to gitlab workflow (goal, plus comparison and results section)

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(13.82GB free)

| Extension | Eager | Load Code | Call Activate | Finish Activate | Event | By |
|---|---|---|---|---|---|---|
| vscode.debug-auto-launch | true | 2 | 0 | 2 | * | vscode.debug-auto-launch |
| vscode.git | true | 22 | 13 | 27 | * | vscode.git |
| vscode.git-base | true | 2 | 0 | 2 | * | vscode.git |
| vscode.github | true | 3 | 0 | 0 | * | vscode.github |

SufyanDahalan.vsworkbench | false | 2 | 3 | 3 | onStartupFinished | SufyanDahalan.vsworkbench |
Vue.volar | false | 51 | 3 | 0 | onLanguage:markdown | Vue.volar |

: Demo or smth

## 3.10 goals moving forward

todo{write about using html/utf8 icons and so on.}

# Abbildungsverzeichnis

2.1 GraphQL Query and Result . . . . . . . . . . 5

3.1 The groupView Tree View in VSWorkbench . . . . . . . . . . 16
3.2 JSDOC Output . . . . . . . . . . 19

# Quellcodeverzeichnis

# Tabellenverzeichnis

# Symbole

# Abkürzungen

# Akronyme

**CLK** Clock *siehe* SCL & SCK

**SCK** Serial Clock *siehe* SCL & CLK

**SCL** Serial Clock Line *siehe* SCK & CLK

# Glossar

**Rekursion**

*siehe* Rekursion

# Literatur

[BL95] Berners-Lee, Tim. *Hypertext Markup Language - 2.0.* https://datatracker.ietf.org/doc/html/rfc18
IETF RFC 1866. Nov. 1995.

[Byr15] Byron, Lee. *GraphQL: A data query language.* Sep. 2015. URL: `https://engineering.fb.com/2015/09/14/core-data/graphql-a-data-query-language/`.

[Csi00] Csikszentmihalyi, M. *Beyond boredom and anxiety.* Jossey-Bass, 2000.

[Dan09] Dangoor, Kevin. *What Server Side JavaScript needs.* CommonJS Announcement. Jan. 2009. URL: `https://www.blueskyonmars.com/2009/01/29/what-server-side-javascript-needs/`.

[Eva22] EvanLi. *Top 100 Most Starred Repositories on GitHub.* Aug. 2022. URL: `https://github.com/EvanLi/Github-Ranking/blob/47ec47b28f64da66f2dcfcab9b3f791c19af6a7a`
`Top100/Top-100-stars.md`.

[FIG+99] Fielding, R.; Irvine, UC; Gettys, J.; Compaq/W3C; Mogul, J.; Compaq; Frystyk, H.; W3C/MIT; Masinter, L.; Xerox; Leach, P.; Microsoft; Berners-Lee, T. und W3C/MIT. *Hypertext Transfer Protocol – HTTP/1.1.* Juni 1999. URL: `https://datatracker.ietf.org/doc/html/rfc2616`.

[Gam15] Gamma, Erich. *Hello Code.* Nov. 2015. URL: `https://github.com/microsoft/vscode/tree/8f35cc4768393b25468416829e980d7550619fb1`.

[HWL96] Hakon Wium Lie, Bert Bos. *Cascading Style Sheets, level 1.* https://www.w3.org/TR/REC-CSS1-961217. Dez. 1996.

[Inc21] 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`.

[Inc22] Inc., Stack Exchange. *2022 Developer Survey*. TypeScript appearing in charts as the 4rth most loved technology by developers in 2022. Juni 2022. URL: `https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted`.

[Rob13] Robin, Adam. *initial commit*. März 2013. URL: `https://github.com/electron/electron/tree/e451d9212179197b88abeb752602de3859bb1765`.

[Rul69] Rulifson, Jeff. *DEL*. Juni 1969. URL: `https://datatracker.ietf.org/doc/html/rfc5`.

[SAK+18] Shakeri, Zahra; Abad, Hossein; Karras, Oliver; Schneider, Kurt; Barker, Ken und Bauer, Mike. *Task Interruption in Software Development Projects*. Mai 2018. URL: `https://arxiv.org/pdf/1805.05508.pdf`.

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

# Weiterführende Literatur

[DHM12] D. Hardt, Ed. und Microsoft. *The OAuth 2.0 Authorization Framework*. Okt. 2012. URL: `https://datatracker.ietf.org/doc/html/rfc6749`.