UNIVERSITÄT
BERN

WebAssembly Security

What security-related questions do developers discuss about WebAssembly?

Bachelor Thesis

Pascal André
from
Tschingel ob Gunten BE, Switzerland

Faculty of Science, University of Bern

December 30, 2021

Prof. Dr. Oscar Nierstrasz
Dr. Mohammad Ghafari

Software Composition Group
Institute of Computer Science
University of Bern, Switzerland

Abstract

WebAssembly (WASM) is a new binary-format code that runs in browsers and serves as a compilation target for other programming languages. Although WASM is specified to run in a safe, sandboxed environment, there are still security concerns that we investigate in this thesis. We use Stack Overflow to gather a dataset of security-related WebAssembly questions. We investigate what types of security-related WebAssembly questions developers ask and what topics that they talk about in their posts. Additionally, we analyse the activity of the developers that are responsible for all the questions, answers and comments to find out if certain groups among them are particularly active.

Contents

1 Introduction

In this introductory chapter, we give a brief overview of WebAssembly and present the research questions that will guide us through the following chapters.

WebAssembly (or short WASM) is a binary-format code that can be run in browsers and exists since 2017. It serves as a compilation target for other programming languages. For example: If you have C++ code from a game or virtual/augmented reality app, it can be compiled to WebAssembly in order to run inside the browser. Its goals are to aim for a near-native performance across platforms by taking advantage of common hardware capabilities and its low-level assembly language can be viewed, written or debugged by hand.

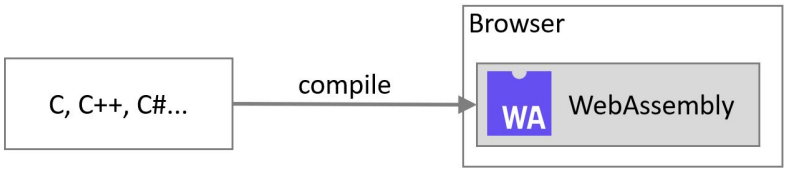


Figure 1.1: WebAssembly serves as a compilation target for applications written in C, C++, C# and many other programming languages which can then be run within the browser.

Although WebAssembly is specified to run in a safe, sandboxed environment, there are still developers who have security related concerns that we are investigating.

In this paper, we define the following three questions that will guide us through our research.

Research Question #1:
What are the intentions behind asking security-related WebAssembly questions?

In a first step, we want to find out what types of questions developers ask and why they are asking them. Are they asking a question to better learn a theoretical concept they don't fully understand? Do they need help implementing a particular feature that they have no idea how to start with? Do they need help fixing a bug in their code that they can't solve themselves? Or do they write various other security-related WebAssembly questions, for example to apply best practices to an implementation or to clarify setup or configuration issues?

Using the open card sorting approach, we identified five major question types. The two most popular categories among them are questions regarding bug fix support and instructions on how to implement a particular features, as both appear in roughly 50% of all questions. Issues around fixing bugs most commonly appear when they try to implement authentication that works locally in development mode but not after deployment, though the variety of problems is quite wide. When developers ask for instructions on implementing a particular feature, the question mostly revolves around implementing third party authentication systems among other related issues such as assigning user roles.

Research Question #2:
What are the security-related WebAssembly topics that developers are asking?

In a second step, we want to examine the concrete problems that developers encounter and, as a result, ask the community for help. Do the problems occur in the context of specific services, frameworks, tools, libraries, and so on? By answering this research question, we will then know the security-related WebAssembly issue areas that are causing problems for developers and which of these are particularly prevalent or frequently occur along with other concerns.

After using open card sorting again, we identified 19 topics. The two most popular topics by far were Blazor, which appears in 78% of all questions, and Authentication, which appears in 52% of all questions, followed by other topics such as Azure, Coding Questions, HTTP Requests, API, Storage, and a handful of other topics that are close together at a share of 8-12%. We also found that there is a high correlation between authentication and Blazor, as all questions asked about authentication were also related to Blazor.

Research Question #3:
Can we find groups of developers who ask or answer particularly many questions?

For the third research question, we focus on the developers who raise these security-related WebAssembly issues. We want to find out if there are certain groups of developers who are particularly active in the community and either ask or answer a variety of questions on the topic. In doing so, we will also try to incorporate a developer's knowledge and skill level into the assessment to find out how the experience levels of the developers, from novice to expert, are distributed within the community.

If we look at both the number of questions a developer asks and the number of accepted answers a developer writes, we find that in both groups the vast majority ask only one question or write only one answer. Among the developers who write accepted answers, we could identify a small group of 6.4% of all developers that is responsible for 28% of all accepted answers.

Are there any additional interesting conclusions that we can draw from the dataset?

During our statistical analysis we did at the beginning to find out more details about our dataset, it became apparent early on when we evaluated the tags that questions about Blazor are very popular, as eight of the eleven most popular tags have a strong if not even direct link to Blazor development. From a timeline perspective, we also see that the official release of Blazor in May 2020 led to a huge increase in security-related questions about WebAssembly, of which the popularity has remained consistently high.

2 Dataset

In this chapter, we present the steps we have taken to gather a dataset of security-related WebAssembly questions on which we will base our research. The source of these questions will be Stack Overflow, which, as one of the largest Q&A forums for developers, has captured many current relevant discussion topics and issues as well as those from previous years.

To find the questions that are relevant to us, we use a list of security-related keywords, each of which we will then combine with WebAssembly to create a search term like "WebAssembly [keyword]" to collect and download all resulting search results on that subject from Stack Overflow. In order to automate the process, we will use the StackAPI to avoid having to manually request the data. For easier processing, we then save the obtained JSON data in an Excel file after performing some data transformations.

After that, we make sure that only questions that are actually about security-related WebAssembly issues are present in our dataset and all other questions are omitted.

2.1 Keywords

We begin our research work by consulting security literature to create a list of 35 unique security-related keywords which should cover a large majority of the security-related questions that developers ask in regard to WebAssembly on Stack Overflow.

Original 35 Keywords

security, privacy, vulnerability, attack, exploit, risk, danger, threat, compromise, login, authentication, password, privilege, permission, encryption, leak, breach, injection, sanitize, overflow, bypass, malicious, defense, protect, sandbox, untrusted, trusted, trustworthy, secret, isolate, hazard, expose, signin, verification, certification.

2.2 Keyword Variations

We manually process the keywords to find all possible variations of them in order to match as many Stack Overflow questions as possible. To extend the list, we add for each keyword, if available, its noun (singular and plural), verb (incl. present and past participle), adverb, adjective (several forms by adding suffixes such as "-able", "-ed" or "-ing") and other similar word variations we could find. We also included different variations of how a word could be written, like for example "login", "log-in" or "log in". Some keywords such as "defend" even had multiple variations for the noun that we considered which included "defense" or "defender".

Following this process, the unique keyword "security" resulted in the following seven keyword variations: security (noun singular), securities (noun plural), secure (verb), securing (present participle), secured (past participle), securely (adverb) and securable (adjective).

This process eventually resulted in a final list of 266 keywords based on our initial 35 unique keywords that we extended by finding all their variations that share the same lemma which means that they have the same canonical form. The final keyword list can be found under the additional resources for this paper in section 7.1.

2.3 Stack Overflow

Stack Overflow, or short SO, is a community-based platform where developers post questions and answers to computer programming related topics. It was created in 2008 by the two American software developers Jeff Atwood and Joel Spolsky [5]. Today, SO is part of the Stack Exchange Network, a mostly technology-based Q&A platform, where it serves as their main flagship site.

Since SO became the go-to platform for many developers over the years as they seek advice on their programming related issues, it reflects well the problems that the developer community is facing today or was facing in previous years. It therefore seems to be the most extensive and reliable source for gathering a dataset that we can then base our research on.

2.4 StackAPI

In order to automate the process of collecting all SO questions that match the search term "WebAssembly [keyword]" as well as its answers, comments, developers etc. we are going to use StackAPI [4] which is a Python wrapper for the official Stack Exchange API [1]. Using this library, we can write simple Python scripts to access various endpoints of the Stack Exchange API. To gather the SO questions, we are mostly interested in the `/questions` endpoints but for further analysis of the dataset, we are also going to collect full details for each answer, comment and developer related to the SO questions via their separate endpoints.

2.5 Data Transformation

Now that we have saved all the necessary data in JSON format, we want to transform the data so that we can view, compare and process it more easily later. For this purpose, we save all relevant properties of the questions (incl. title, tags, score, creation date, questioner etc.) in a separate Excel file, with all data for a single question in one row. This makes it easier for us to view the data later, perform statistical procedures and add our results to additional criteria as new columns. We can also use the comment function in Excel to add our notes and reasons for decisions to individual cells.

2.6 Cleaning the Dataset

Because the questions were collected automatically using our list of security-related keywords, it does not guarantee that all of them actually feature security-related WebAssembly topics. To make sure that false positives (questions which are not related to WebAssembly security) are excluded, we apply a rating system with a scale from 0-3 as we process them to assess how strongly the question is linked to WebAssembly security.

Level 0: Irrelevant in an obvious way

The question is in an obvious way neither related to security nor WebAssembly because the user has an issue that is explicitly NOT related to WebAssembly or security. Consequently, this question is a false positive and is not relevant for our research work and we exclude it during our later analysis of the dataset.

Examples for Level 0

Question ID: 61684831

This question was matched during a search request with the term "WebAssembly protect". In its body, the owner writes "*. . . finding guidance on using Blazor Server (not WebAssembly). . .* ". The developer describes an issue that is explicitly not related to WebAssembly, so we ignore it.

Question ID: 62114179

The second example features a question that was matched with the search term "WebAssembly security". The developer asks "*Other than portability and security reasons, why would someone want to run their existing Go/Rust/C++ applications in web browsers via WebAssembly?*". Unlike the first question, this one is actually related to WebAssembly but is clearly not related to security, so we also exclude it.

Level 1: No relation to WebAssembly Security

Compared to level 0, where it explicitly is NOT related to WebAssembly or security, level 1 does not have such remarks. However, it also does not have any clear safety-related topics and if we could identify any, they are far-fetched and not related to the original topic. We therefore do not consider the question at a later stage during our evaluation.

Examples for Level 1

Question ID: 63593076

In this first example, we have an issue from a developer who's working on a Blazor WebAssembly app where the retrieved models have all their properties set to `NULL` because he forgot to add `get` and `set` to his model properties. Since there are no obvious security-related implications and the issue is more a general bug, we assign it *Level 1*.

Question ID: 66923063

The owner from this question asks how to exclude unused `.dll` files when publishing a WebAssembly project. This problem is related to configuration settings in the project and does not have any obvious security-related issues which is why we assign it a *Level 1* and do not consider it later.

Question ID: 62107035

The last example is about an issue where docker does not reflect the changes in the HTML file from a WebAssembly app due to missing port forwarding. Since this is a general setup issue without any obvious security-related implications, we assign a *Level 1*.

Level 2: Weak link to WebAssembly Security

Level 2 is assigned to those questions which have a slight reference to any, also in a more distant sense, safety-relevant topic, but which are still related to the given question. This includes questions where the main question does not have a direct security implication, but some of the additional questions do, which the developer asks along the way in the same post. Later on, we will consider these questions for our evaluation.

Examples for Level 2

Question ID: 66733106
The developer's main question is how JavaScript and WASM virtual machines work. This is more of a general question that does not have too obvious security implications. Nevertheless, we assign a *Level 2* to the question since in one of the additional questions the user also asks if the virtual machines provide isolation.

Question ID: 61821267
In this example, the user asks if there exists a consortium that is curating a content delivery network (CDN) for WebAssembly libraries that would offer improved loading times due to caching or, in regards to encryption and security, centralized standards for their behavior. Although it was a more general question, there are safety-related aspects, as just mentioned, which justify *Level 2*.

Question ID: 61052684
The developer asks how to load a WebAssembly module locally, which is the main point of the question. *Level 2* is assigned because of the security-related implications that come into play because he ran into CORS issues that blocked direct access to the local module since he did not want to use HTTP to load it.

Level 3: Strong link to WebAssembly Security

A question of level 3 has a strong link to security, meaning its topic has a direct impact on protecting a system from threats that could exploit vulnerabilities, and securing it by giving access to its resources by only authorized or trusted parties.

Examples for Level 3

Question ID: 66873168
The first example features a developer who's asking for help on how to implement a user management module according to best practices. Consequently, this question has a strong link to security and gets assigned a *Level 3*, as it is about giving only authorized users access to the system.

Question ID: 61956919
In the second question, its owner asks how to setup and run the client side of a Blazor WebAssembly app using HTTPS on a custom domain. We see a strong link to security in this example since HTTPS supports a secured communication.

Question ID: 57993453
The third and last example is about a developer who tries to access the memory of a WebAssembly module that is loaded by webpack. We assign this question a strong link to security because it is about exposing memory of a WASM module.

To decide which questions are relevant or irrelevant to our work, we are going to refer to this scale from level 0-3 to rate how strongly a question is related to WebAssembly security. With level 0 and 1 we describe questions that have no relation to WebAssembly security and will therefore be omitted from the final dataset for this reason. In the initial 467 questions, level 0 or 1 is identified for 108 of them. From this we conclude that 23% of the questions from the original dataset are false positives.

The vast majority of the keywords that we identified within the excluded questions were “log in” followed by “private” and “sign in”. The keyword “log in” caused the most false positives because its first half “log” was matched in many questions where it was used in the context to log data to the console, so it was found in code snippets and problem descriptions where developers explained how they log data to the console, but the post itself was not a security-related WebAssembly topic. Similar issues and consequences occurred with the other two keywords “private” and “sign in” whereas “private” was matched as the access modifier in code snippets and “sign” was identified in problems related to signed or unsigned integers.

We can additionally determine from this result that 359 questions (77%) are rated with level 2 or 3, since they are security-related WebAssembly questions. These 359 relevant Stack Overflow questions are the ones that create the final dataset that we are going to use in our work from here onward.

Note: Any later mention of the *dataset* now refers to these 359 questions. Should we need to address the initial 467 posts at some point, we are going to denote them as the *original dataset*.

3
Methodology

In this chapter, we present our methodology for our research approach. We describe the two processes with which we want to work out a list of developer intentions and question topics that are featured within our dataset of 359 Stack Overflow questions. We mainly focus on the process of how these developer intentions and question topics were gathered to show how we assessed the questions and what rules we based our decisions on. The results (developer intentions and question topics) will be presented in the subsequent fourth chapter along with detailed definitions and extensive examples.

The developer intentions focus on the question types why a particular question was asked whereas the question topic focuses on the problem area in which its corresponding developer was asking for help. At this stage, all the important properties from the Stack Overflow questions were saved in an Excel spreadsheet. The remaining data was stored in separate files in their original unedited JSON format, exactly how they were requested via the StackAPI.

3.1 Checking Question Properties

In this section, we process and review basic properties of all 359 questions in our dataset to better understand them and lay a foundation for later steps. For each question, we record the following details:

- Brief summary describing the fundamental problem in the question
- Up to about five tags that cover the relevant issues in the question
- Security related keywords from our list that were matched within the question

As we will discuss in more detail shortly, we also record the following properties for each question:

- Was the question resolved by the question owner himself or a different developer?
- Does the answer status correctly reflect the state of the question?
- Which observations can we draw why the question was likely answered or not?

3.1.1 Was the question resolved by the owner himself or not?

We want to keep track of whether a question was resolved by the owner himself or a different developer. Because there is no attribute that records this property, we manually check it and save it as a boolean value named `resolved-by-owner`. If the owner responds to his question by either writing a comment or posting an answer where he explains how he solved the issue, we mark the corresponding boolean `TRUE` or `FALSE` otherwise. This property can later help us answer questions such as:

- Which question types are answered by the question owner himself most often?
- Do question owners respond more quickly to their posts compared to other devs?

Additionally, it helps us to exclude the questions where the user was able to fix the issue himself and therefore did not necessarily require any interaction with another developer which is useful to know for some of our later analyses of the dataset.

Examples

Question ID: 63000069
Question owner writes solution as comment:
In this first example, the question owner "Ronak SHAH" describes a problem where his requests using QNetwork are getting blocked due to CORS errors. He later writes in a comment to the original question that configuring the server to accept requests from different origin (i.e. through WASM) has solved his issue. Thus, we mark the boolean `resolved-by-owner` as `TRUE`.

Question ID: 66878555
Question owner writes solution as answer:
A similar problem can be found in the second example where the developer "bedrock" has `PUT` and `DELETE` requests being blocked by CORS. The solution, which was written by "bedrock" as an answer, was to add `AllowAnyMethod` in CORS config settings of the web service app. Consequently, we also mark this question's `resolved-by-owner` boolean as `TRUE`

3.1.2 Is the question's answer status property set correctly?

Every question has an `is_answered` boolean property that keeps track of whether a question has been answered or not. Due to the fact that this boolean is changed by the owner of the question, this value might not always reflect the current status of the question, meaning another developer might have given an accepted answer or the owner was able to resolve the issue himself but, in neither case, marked the question as answered so the property does not accurately reflect the question's answer state.

Verifying the `is_answered` boolean gives us more accurate results when answering questions where the answer status is included in its analysis such as:

- Which types of questions are answered most often?
- Overall, what percentage of questions has been answered?

As we process the questions, we check if the `is_answered` property matches the state of what we have seen or read in the question, answer or comment. If for instance a question with `is_answered: FALSE` has either an accepted answer or an answer from another developer which is not marked as accepted but has a comment from the question owner that it resolved the issue or finally the owner himself writes a separate answer or a comment to the original question where he explains how the issue was resolved, then we flag the question meaning that its `is_answered` property should be `TRUE`.

Examples

Question ID: 66949040
Question has an accepted answer:
The first example features the case where the owner has marked one of the answers as the accepted one but the `is_answered` property of the question is still `FALSE`, which is why we flag this question.

Question ID: 63000069
Comment from question owner resolves issue:
This question shows an example where the `is_answered` property is set to `FALSE` even though the owner responds to a comment explaining how the issue was resolved. Consequently, we flag this question because the answer state is not correct.

Question ID: 64291881
Answer from question owner resolves issue:
The last example of a question with `is_answered` set to `FALSE` is the case where the owner himself writes a separate answer where he explains how the issue was resolved, which is why we flag this question as well due to incorrect answer state.

3.1.3 Why was the question likely answered or not?

Another feature we want to capture only for unanswered questions are reasons why a question has not yet been marked as answered. With this data, we could then later evaluate what were the most common reasons why certain questions were not answered. The reasons recorded could be, for example:

- The problem was in relation with another tool or library and therefore requires additional knowledge and experience in order to be solved.
- The question owner gave very few details about the problem which makes it difficult for other developers to provide assistance because they can only assume certain details or make an educated guess.
- The question owner did not respond to any comment or answer from another developer which would tell whether the issue was resolved or not.

Examples

Question ID: 65950937
No response from question owner:
In this example we have a question from developer "user82395214" with two comments and one answer. Unfortunately, he did not respond to any of them, so we can't tell if they would have resolved the issue, so the status remains unanswered because the question was abandoned by its owner.

Question ID: 65078337
Question owner gave very few details:
The second question features a problem from a developer who created a Blazor WebAssembly app that he debugged on IIS (Internet Information Service) where the app was opened in a new Chrome window where he was not signed in. It seems that he gave very few details because he did not mention what system he was on or which browser version he's running which, amongst other additional details, would presumably help in solving the presented issue.

Question ID: 57993453
Problem is related to another tool or library:
The third example covers the case with a specific question in relation to another tool or library. In the given scenario, the developer asks how to access the memory of a WASM module that is loaded by webpack. In order to answer this question, it needs additional knowledge of webpack in order to know how one can access the module's memory, which is a likely reason why the question was not answered.

3.2 Developer Intentions

In the next step of our methodology, we focus on the question types which will help us in finding the intentions why developers are asking these security related WebAssembly questions. This can later help us to answer the following questions:

- Which question types are asked or answered most often?
- Which question types get the quickest response?
- With which intentions do developers ask questions?

Because we don't have an initial list of categories to which we could assign the questions, we have to come up with a process that will help us to create said list. We decided that the *open card sorting* approach would be most suitable for this process.

Applying the open card sorting approach means that at the beginning we study the first question and record its intention as "Category A". Then we go to the second question and identify its intention. If the identified intention corresponds to the one of "Category A", we add it to this category, otherwise we create the new "Category B" and assign it to this group. We repeat this process for all questions, so that in the end we have a list of intentions where each category is represented in one question type.

For each intention AKA category in our list, we add a title, a short description and definition of the intention as well as a sample question from the dataset which is assigned to this category. If the number of intentions gets too large and exceeds a total count of 15, we might have defined them too specifically. At that point it would make sense to always try to find the two closest intentions and merge them together, which makes them a bit less specific and reduces the total number of categories.

After the first 50 questions have been categorized, it has become apparent for many questions that it would be better if we could assign a question to more than one category, since in most cases several intentions can be identified in a question. In order to improve the categorization process and to get more accurate results later in the evaluation, we decided to revise the original process. In the new process, any number of categories can now be assigned to a question, depending on which ones can all be identified. In addition, we use a scale from 0-3 to describe how strongly a category was identified in the question.

The revised procedure allows for questions to be assigned more accurately to all appropriate categories. Later, improved results can be obtained when evaluating the data.

Similarly, to "How strong is the question related to WASM security?", we also apply a scale from 0-3 to reflect how strongly a category is identified within a question:

Level 0 *(or just NULL by adding no value at all for simplicity) means that this category has not been identified within the question and we exclude this question during later analysis for that particular category.*

Level 1 *means that the category was identified within the question but it has a weak link to it and is not relevant enough to the main topic so that we would consider this question during later analysis for that particular category.*

Level 2 *is assigned if the category was in the possible selection for the chosen category in the old procedure, because it was identified in the question and was relevant to it. Level 2 is the advantageous level that allows us to select other secondary categories relevant to the question, which will later be included in the evaluation of that category and should provide better results, as we can record the relevant information more precisely.*

Level 3 *is selected for the category that reflects the primary category of the question and corresponds to the category that would have been selected in the old procedure, in which only one category could be selected. The question will be considered during later analysis for that particular category.*

Categorizing based on Developer Intentions

Question ID: 66567628
In the first example the developer asks how to authenticate users so that they can only fetch their own personal images with a `GET`-request. After reading the question and studying its problem we conclude that he mainly asks for specific instructions on how to implement the feature regarding the user authentication when fetching personal images during `GET`-requests, which is why we assign the category "How-To Instructions" with a level 3 to it. In the question, the developer also explains a possible solution that he could attempt, where he converts the image into base64, but this bypasses the browser cache and seems needlessly complicated which is why he is looking for a better solution and why we also assign "Best Practice Guidance" as a secondary category with level 2 to it. Of the remaining categories none have been identified. In summary, this question will be considered during the evaluation of the categories "How-To Instructions" and "Best Practice Guidance" due to their individual score of a level 2 or 3.

3.3 Question Topics

In the previous section, we examined which types of questions developers asked (such as clarifications, how-to instructions, bug fix support etc.) using the open card sorting approach. Now we want to make a second similar pass over the dataset but this time we focus on the topics of the developer's problems that they ask help for. What are the actual issues that they are facing which we can identify using the open card sorting approach?

For more accurate results, we still allow multiple topics to be assigned to a question but because we expect a higher number of topics in comparison to the categories or question types, we are no longer going to use a scale from 0-3 to reflect how strongly a topic was identified within a question. This will help us to simplify the evaluation process due to the higher complexity caused by the increased number of topics. Instead, we simply use boolean values, meaning if a topic was identified within a question, we assign a 1 and if the topic was not identified or significant to the problem, we assign a 0.

Using the open card sorting approach, we could identify 19 different topics which we will present in more detail, including definitions and examples, in the subsequent chapter.

4 Results

In this chapter, we are going to present first our general findings followed by the identified developer intentions and question topics, by giving a definition and question examples, that we identified using the open card sorting approach that we explained in chapter 3. After that, we analyze the dataset based on these new results to gather the information that will then help us answer our research questions from chapter 1.

For the results that we will present in the following, it should be noted that they are subject to various threats that could potentially jeopardize their validity which we have explained in chapter 5. The quantity shares, which we will often report as percentages, have been rounded to the nearest integer.

4.1 General Findings

In this section, we are going to evaluate our dataset based on the available properties that we can gather from the Stack Overflow questions as well as our own collected data that we described in the previous chapter to find general findings that help us to better understand our dataset. We want to deduce from this what the most popular tags are, how popularity on the subject changed over time by looking at the number of questions that were asked in each quarter and how many questions received responses from developers and are now in an answered state.

4.1.1 What are the most popular tags?

When we analyse the tags, we need to consider that they are added manually by developers which means some of them may have been added incorrectly or that not all possible tags that would suit the question have been added. The final results are therefore affected by which tags the developer has chosen or which tags were suggested to him depending on the content of the post.

Table 4.1: Most popular question tags

Tag	Count	Percentage
blazor	198	55%
blazor-webassembly	135	38%
webassembly	104	29%
asp.net-core	80	22%
c#	80	22%
blazor-client-side	65	18%
authentication	44	12%
identityserver4	41	11%
javascript	31	9%
asp.net	19	5%
asp.net-identity	18	5%

In total 234 unique tags were assigned to the 359 questions a total of 1'348 times, which is an average of 3.79 tags per question. 147 of these different tags (63%) were used only once on a question. On the other hand, eleven tags (5%) were added to more than 5% of the questions, which is 18 or more questions per tag. Out of the 1'348 times a tag was assigned, the eleven most popular ones, as seen in table 4.1, claim 815 (60%) of these assignments. We can conclude that 5% of the unique tags cover 60% of all assigned tags.

Upon closer inspection of the eleven most popular tags, we notice that eight of them (*blazor, blazor-webassembly, asp.net-core, c#, blazor-client-side, identityserver4, asp.net and asp.net-identity*) have a strong or even direct link to Blazor development.

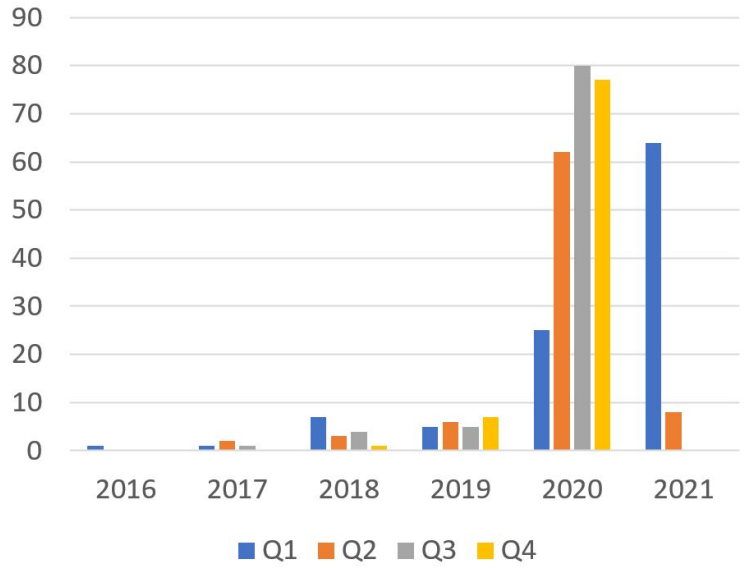
Table 4.2: Most popular programming languages featured in question tags

Tag	Count	Percentage
c#	80	22%
javascript	31	9%
c++	13	4%
rust	6	2%
typescript	4	1%
c	3	1%
html	2	1%
assembly	1	0%
go	1	0%

If we extract all programming or markup language related tags from the list and inspect them separately, as seen in table 4.2, we notice that C# is the most popular because it appears in one out of five questions. A clear second is JavaScript with almost one out of eleven occurrences. The remaining languages are tagged in less than 5% of all questions each.

4.1.2 When were the questions asked?

To find out how the popularity of the subject of security-related WebAssembly questions has changed over time, we look at the quarterly number of asked questions within our dataset.



	2016	2017	2018	2019	2020	2021
Q1	1	1	7	5	25	64
Q2	0	2	3	6	62	8
Q3	0	1	4	5	80	
Q4	0	0	1	7	77	

Figure 4.1: Number of questions asked per quarter since 2016

We notice a massive increase in popularity in 2020. Overall, there were more than ten times as many questions asked in 2020 than there was a year before in 2019. If we consider that the majority of question tags are related to Blazor development and Blazor WebAssembly was officially released in May 2020 (Q2 2020), we can comprehend why the number of questions suddenly spiked in early 2020, especially Q2 to be more specific.

Since Q2 2020 the number of questions has also remained consistently high. Due to the fact that the dataset was collected in early April 2021, the question count for Q2 2021 is not fully representative.

4.1.3 How many questions have answers and are flagged as answered?

Every Stack Overflow question has an `answer_count` property that holds the number of answers that the question received and an `is_answered` boolean property that allows the questioner to set his post as answered or not depending on whether the issue was resolved or still needs a response from the community that answers the question.

While the `answer_count` simply describes the number of answers that the question received and therefore does not leave any room for inaccuracies, the `is_answered` property will hold an incorrect value if the question owner does not set its value accordingly. This is why in section 3.1 we check that the `is_answered` property is set accurately and fix it if necessary to make sure that the following results closer reflect the reality of the answer state.

	is_answered == TRUE	is_answered == FALSE	
answers > 0	219 61%	34 9%	**253** 70%
answers = 0	10 3%	96 27%	**106** 30%
	229 64%	**130** 36%	**359**

Figure 4.2: Distribution of the 359 questions according to their answer state and whether they received answers or not.

Overall, 229 questions (64%) are marked as answered which means their issues have been resolved, whereas the problems of 130 questions (36%) still need to be addressed. From the above table we can also gather that ten questions (3%) were answered without a single answer (for example through just comments or the owner figuring it out on his own) and 34 questions (9%) received insufficient answers, meaning the answers that have been given did not resolve the problem.

From the 359 questions, 24 of them (7%) did not have the correct `is_answered` status set which we fixed as we analysed the questions according to section 3.1.

4.2 RQ #1: Developer Intentions

4.2.1 Identified Developer Intentions

As we explained in section 3.2, we used the open card sorting approach to find a list of question types reflecting the intentions why developers are asking security-related WebAssembly questions. After categorizing all 359 questions, we identified the following eight categories for each of which we will give a definition describing its type of question as well as a concrete example with a question from our dataset.

Category #1: Best Practice Guidance
Developer asks for best practice guidance on how to solve an issue or implement a certain feature.

Example for Category #1: Best Practice Guidance

Question ID: 66873168
The user develops a Blazor WebAssembly app where he is trying to create a basic user management module (have admin that can create and handle users including different roles) by following a guide. He asks for guidance on applying best practices according to today's standards.

Category #2: Bug Fix Support
Developer asks for help on resolving a problem with a bug he cannot fix.

Example for Category #2: Bug Fix Support

Question ID: 59696171
The developer asks for help regarding a bug that he has encountered in his Blazor WebAssembly app where after a successful login procedure (using the Identity Framework) the StateProvider in WebAssembly (that stores the details about the currently authenticated user) is always false.

Category #3: Clarifications
Developer asks for conceptual clarifications on a given topic that they do not fully understand yet and need help with.

Example for Category #3: Clarification

Question ID: 42186728
In this example, the user asks for clarification if WebAssembly can be used to enforce digital rights management (DRM). He wants to know if it is possible for developers to detect if browser games are used by other people and lock down their access with a WebAssembly script that is deeply tied into the game mechanics.

Category #4: How-To Instructions

Developer asks for specific instructions on how to implement a certain feature. Simple "How to fix this bug?" questions are not considered in this category and are assigned to the "Bug Fix Support" category.

Example for Category #4: How-To Instructions

Question ID: 65735822
In the given sample question, the developer asks for specific instructions regarding how to configure a Blazor WebAssembly app to enforce strict-transport-security in its response headers.

Category #5: Not Supported

Developer asks for help to implement a feature that cannot be done in such a way or tries to do something with a tool that is not possible because it does not support that functionality.

Example for Category #5: Not Supported

Question ID: 61190809
The question owner tries to run WebAssembly on the new V8 Google Apps Script runtime, but it seems that async functions are terminated after they return a promise. Another developer writes in his answer that async functionalities are not fully supported in V8 yet and refers to an open issue tracker regarding this issue.

Category #6: Setup or Configuration Issue

Developer asks for help regarding an error that is related to an incorrect setup or configuration.

Example for Category #6: Setup or Configuration Issue

Question ID: 66878555
This example features a question from a developer who has issues with his Blazor WebAssembly app because the HTTP `PUT` and `DELETE` requests are being blocked. Updating the CORS configuration settings by adding "AllowAnyMethod" in the setup of the web service app solved the problem.

Category #7: Third-Party Bug

Developer asks for help regarding an issue that is caused by a bug in a third-party software, meaning he did everything right and his project should run fine if it was not for the issue in the third-party software that he is not responsible for.

Example for Category #7: Third-Party Bug

Question ID: 40625530
The question owner tries to follow a WebAssembly starter guide on his Mac but gets stuck due to an issue with missing files. It turns out that he did everything right, but the problems were encountered due to a bug in Emscripten that can cause errors when trying to build binaryen natively on some OS X setups. The problems were not caused by the developer who asked the question on SO, but by a third-party, in this case the software Emscripten.

Category #8: Unexpected Results

Developer asks for help on an issue with unexpected behavior or error message that confuse him and he needs support to understand them.

Example for Category #8: Unexpected Results

Question ID: 65536335
The user has an issue where the build is different when using Docker Compose in Visual Studio vs. using Docker in CMD despite the fact that both builds should be the same.

4.2.2 Evaluating Developer Intentions

In section 3.2 we described the analysis process with which we wanted to record the intentions why developers ask questions. For example: to better understand a concept (clarification), to find out how to implement a feature (how-to instructions), to find errors during setup (setup or configuration issue) or during development in general (bug fix support), as well as, among others, questions about the implementation of common standards (best practice guidance). We will evaluate this collected data in the following.

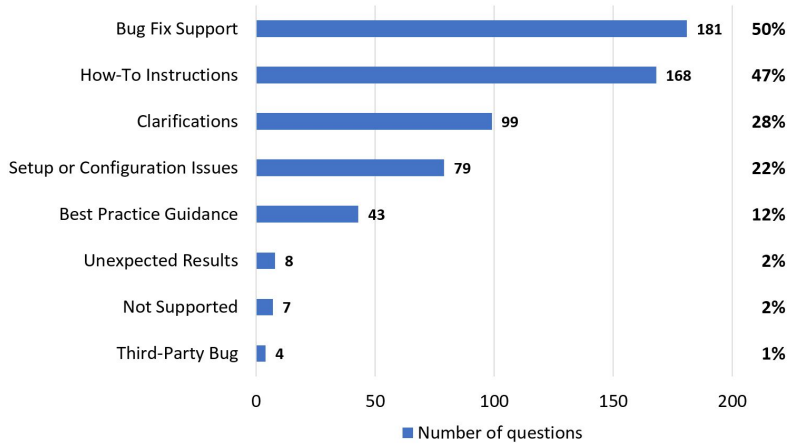


Figure 4.3: Number of questions per category

First, we evaluate the distribution of the different categories. Based on the bar chart in Figure 4.3, we see that with a share of 50%, every second question dealt with a bug. Close behind with 47% are questions about getting support on how to implement certain features. Much further down the list, but still to be found in every fourth question (28%), is assistance in clarifying certain topics. Setup and configuration problems occur in 22% and best practice guidance in 12% of all questions.

At the bottom we find the three niche categories “Unexpected Results”, “Not Supported” and “Third-Party Bugs”, each occurring in less than 3% of all questions. Due to their low relevance, we will focus on the top five categories in the following.

Next, we will analyze the questions according to their data on the number of views, upvotes, answers and the duration until a first reaction according to common statistical evaluation methods by calculating the average, median, range, standard deviation and the interquartile range.

For the question data, we consider the number of views (how often a question was viewed), number of upvotes (how often a question received an upvote from a developer), number of answers (how often the question received an answer from a developer), and the duration until the first reaction (duration between the time the question was published until the time the first comment or answer to the question was published).

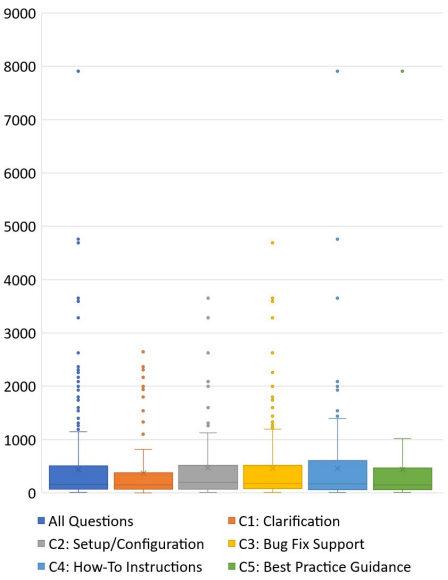


Figure 4.4: Box plot of the number of views that the questions within a category received.

In Figure 4.4 we see the box plots of the number of views the questions received. On the far left we see the box plot across all 359 questions in our dataset, the other five are then the further box plots for our five categories, which only contain their relevant questions. As we can see, there are no significant differences between the categories. Those questions that would have made a bigger difference are considered outliers from a statistical point of view.

We have made the same observations for the other criteria on the number of responses, question scores and response times, which is why we will not discuss and display their box plots further. However, we record the statistical values for the various properties such as views, scores, responses and reaction times across our dataset in the table in Figure 4.5 for the sake of completeness, but as mentioned, we do not go into detail about the individual categories.

	Views	Scores	No. of Answers	Reaction Time
Minimum	7	-1	0	82
Q1	68	0	0	1,762
Median	162	0	1	7,128
Q3	510	1	1	54,161
Maximum	7,905	15	6	23,255,892
Mean	441	1.15	0.98	471,159
Range	7,898	16	6	23,255,810
IQR	442	1	1	52,400
IQR Low	-374	-1	-1	-50,638
IQR High	952	2	2	106,561

Figure 4.5: General stats over the entire dataset of question properties including views, scores, answers and reaction time.

In section 3.1 we have recorded which questions were answered by the questioner himself or by another developer. We now want to evaluate our results from Table 4.3.

Category	Question Count	Percentage
All Questions	73	20%
Setup or Configuration Issues	34	43%
Bug Fix Support	55	30%
Best Practice Guidance	7	16%
How-To Instructions	26	15%
Clarification	11	11%

Table 4.3: Number of questions per category that were answered by the question owner

Over the entire dataset, 73 questions (20%) were answered by the questioner himself. If we now include the different question types, we notice considerable differences between the categories. At the top are installation and configuration problems, which with 43% were most frequently solved by the questioner. Bug fix support questions are also above the general average with 30%. Best practice guidance and how-to instructions are clearly behind. Questions to clarify certain topics are answered least often by the questioner with 11%.

We find that questions about installation, configuration, and bug problems are relatively often solved by the questioner, as one can clarify such questions through one's own efforts with debugging, research as well as trial and error. However, this is more difficult for the other three categories, since the necessary experience including know-how cannot be acquired so easily to answer questions of this type.

Category	Question Count	Percentage
All Questions	130	36%
Best Practice Guidance	17	40%
How-To Instructions	63	38%
Bug Fix Support	64	35%
Clarification	31	31%
Setup or Configuration Issues	16	20%

Table 4.4: Number of unanswered questions by category

Since we checked the "answer state" for each question in section 3.1, we next want to find out how many questions are unanswered in each category, and then calculate relatively within a category what the proportion of unanswered questions is. The results

in Table 4.4 give us an indication which types of questions tend to be more difficult to answer.

Across the entire dataset, we already know that 130 questions (36%) have yet to receive a sufficient answer. The highest percentage of unanswered questions is in the best practices category, where two out of five questions (40%) are unresolved and thus tend to be more difficult to answer. Also above average are how-to instructions at 38%. In contrast, questions on installation and configuration problems are the most likely to be answered, with only one out of five questions (20%) still unanswered.

4.2.3 Developer Intention Results

Motivation: By answering the first research question, we can tell what types of questions developers are asking and for what reasons they asked the question to seek developer help from the community.

Approach: In section 3.2, we described the process of the open card sorting approach, which we used to extract a list of categories AKA question types. When manually processing the questions, we then assigned the appropriate category to each Stack Overflow question on a scale from 0-3. In subsection 4.2.2 we then analyzed all the data and for each category we considered those questions that scored at least a 2 on this scale. We then compared the number of questions per category to find the most popular categories among them.

Results: After applying open card sorting, we obtained a list of eight categories or question types: Clarification, How-To Instructions, Bug Fix Support, Best Practice Guidance, Setup or Configuration Issues, and the smaller niche categories of Unexpected Results, Unsupported, and Third-Party Bugs. Due to the low popularity of the last three niche groups, we did not explore them further. The most popular categories included Bug Fix Support occurring in 50% of all questions, with How-To Instructions close behind at 47%. The remaining three categories scored 28% (Clarifications), 22% (Setup or Configuration Issues), and 12% (Best Practice Guidance).

Summary: In terms of security-related WebAssembly questions, developers on Stack Overflow mainly ask five different types of questions: Clarification, How-To Instruction, Best Practice Guidance, Setup or Configuration Issues, and Bug Fix Support. The most popular among them are Bug Fix Support and How-To Instructions, which appear in about half of the questions.

4.3 RQ #2: Question Topics

4.3.1 Identified Question Topics

Using the open card sorting approach according to section 3.3, we could identify 19 topics which we will now list below along with a definition and several sample questions. To keep the list as concise as possible, we previously removed any topic with fewer than five occurrences over the questions within our dataset. This decision caused the topics SignalR, WebPack and QT to be removed as they were too unpopular. Since every question can have several topics assigned, the given sample questions could often be used as an example for multiple topics.

Topic #1: Azure
Questions with issues related to Azure services including Azure Active Directory (AAD), Blob Storage, B2C etc.

Examples for Topic #1: Azure

Question ID: 65808332
Questioner asks how to upload a file from Blazor WASM app to Azure Blob Storage.

Question ID: 62812702
Questioner asks how to fix an issue where app roles do not work in a Blazor WASM app using Azure Active Directory.

Question ID: 61973708
Questioner asks how to properly log out a user in a Blazor WASM app using AAD B2C.

Topic #2: API
Questions related to application programming interfaces (API) to perform CRUD operations, changing configuration options, managing endpoints and other API-related activities.

Examples for Topic #2: API

Question ID: 63782180
Questioner asks how to restrict a user from consuming an API endpoint for one location but not for the other.

Question ID: 60754751
Questioner asks how to handle multiple tokens for different scopes with protected API's using Azure Active Directory in a Blazor WASM app.

Question ID: 63849288
Questioner asks where to store and how to use the Auth0 keys that are required to interact with his API in his Blazor WebAssembly app.

Topic #3: CORS

Questions related to issues around CORS (Cross-Origin Resource Sharing) including how to enable, configure or disable CORS and fixing the errors it causes.

Examples for Topic #3: CORS

Question ID: 60727676
Questioner is struggling to enable CORS on the server side of his Blazor WASM app even tough he is following Microsoft's official documentation.

Question ID: 66884642
Questioner tries to fix a CORS error when trying to access his endpoint `/connect/token`.

Topic #4: Coding Questions

Questions related to the code such as syntax errors, compilation issues or how to write the code to implement a certain feature in any programming language.

Examples for Topic #4: Coding Questions

Question ID: 58908855
Questioner asks which JavaScript operations, if there are any, are guaranteed to not cause a StackOverflow RangeError.

Question ID: 49776226
Questioner asks for help to rewrite a code snippet from JavaScript to C.

Topic #5: Interaction between WASM and Programming Language

Questions related to interactions between WebAssembly and other programming languages (PL), browsers or tools for instance to pass data from a WebAssembly module to another PL or calling a WebAssembly module from another PL to describe the opposite direction.

Examples for Topic #5: Interaction between WASM and PL

Question ID: 50149603
Questioner asks if WASM could be used to check the integrity of a JavaScript function.

Question ID: 61696080
Questioner asks if it is possible to directly access the current URL in WebAssembly.

Question ID: 49538827
Questioner asks how private his code in WebAssembly is and whether it would be possible to transform the compiled WebAssembly back to C++ if it was sent to the client side.

Question ID: 47529643
Questioner asks how to return a string (or similar datatype) from Rust in WASM.

Topic #6: HTTP Requests
Questions related to HTTP requests such as how to create or send them, perform CRUD operations, set headers, send tokens or fixing any request-related errors.

Examples for Topic #6: HTTP Requests

Question ID: 66567628
Questioner asks how to authenticate users on image requests in a Blazor WASM app.

Question ID: 63831943
Questioner asks how to fix an issue where the HttpClient does not add cookies to the requests in Blazor WASM.

Topic #7: Browser
Questions with specific issues related to the browser like developer tools, features that only work in specific browsers or errors that differ depending on the browser that was used.

Examples for Topic #7: Browser

Question ID: 60164866
Questioner asks if there is a way to determine if a browser will be able to handle the Blazor WASM app correctly or not.

Question ID: 59790919
Questioner asks how to detect information about the currently used browser from within a WebAssembly module.

Question ID: 60986461
Questioner asks if there is a way to increase the size of the local storage without having to use a Google Chrome extension.

Topic #8: Navigation and Redirect
Questions related to navigation within an app and redirecting a user to a different location.

Examples for Topic #8: Navigation and Redirect

Question ID: 64947553
Questioner asks how to redirect a user after successful login using Azure Active Directory B2C in a Blazor WASM app.

Question ID: 63443588
Questioner has an issue where Blazor WASM using IdentityServer4 hangs on navigating the user to the login and register pages.

Topic #9: Blazor
Questions related to applications that are developed using Microsoft's Blazor and subsequently use WebAssembly to run the client-side C# code in the browser.

Examples for Topic #9: Blazor

Question ID: 63611732
Questioner asks for suggestions on how to handle 403 errors in a Blazor WebAssembly app.

Question ID: 60593985
Questioner asks how to authorize a Blazor WebAssembly SPA using Identity Server.

Question ID: 62467195
Questioner asks how to use two-way data binding in a Blazor Component Library project.

Topic #10: Hosting and Server
Questions related to hosting an application on a server and other general server-related issues regarding setup and configuration settings.

Examples for Topic #10: Hosting and Server

Question ID: 61120227
Questioner asks how to host the Blazor WASM client app on a different port to the server API.

Question ID: 63371014
Questioner asks for help on how to fix an issue where a default Blazor PWA project cannot be hosted into Internet Information Services (IIS).

Topic #11: User Interface
Questions related to the user interface of an application such as adding new elements to it, adjusting its behaviour or modifying pre-existing template components.

Examples for Topic #11: User Interface

Question ID: 65979821
Questioner asks how to switch between layouts for different types of users in a Blazor WASM app.

Question ID: 62834687
Questioner wants to know if it is possible to configure a Blazor WASM app that uses OIDC by adding a custom login component or page.

Question ID: 65238256
Questioner asks how to use the same login page in multiple projects of a Blazor WASM app.

Topic #12: Deployment of app

Questions related to building, publishing and deploying a new app or an already published or deployed app.

Examples for Topic #12: Deployment of app

Question ID: 60932702
Questioner asks how to fix an issue where login and registration work in development mode but not as soon as the project is published since it returns a 404 error.

Question ID: 64543545
Questioner asks for help regarding a problem where in a Blazor WASM app tokens are not used when it is deployed on IIS (Internet Information Services).

Topic #13: Performance

Questions related to performance issues such as finding out why a process performs the way it does or further improve the performance of an existing process.

Examples for Topic #13: Performance

Question ID: 46331830
Questioner asks for help in figuring out why his WebAssembly function runs slower than the JavaScript equivalent.

Question ID: 65021896
Questioner asks how to make his ASP.NET Core hosted website using Blazor WebAssembly load faster.

Question ID: 65011727
Questioner asks if it is okay and optimal to call `stateHasChanged()` once every second in his Blazor WebAssembly app or would it impact performance in the long run.

Topic #14: Emscripten

Questions related to the LLVM/Clang-based compiler Emscripten that can compile source code written in C and C++ to WebAssembly.

Examples for Topic #14: Emscripten

Question ID: 54646505
Questioner asks if you can exploit Emscripten-compiled WASM to run arbitrary JavaScript code.

Question ID: 63952910
Questioner asks how to access the File System API of Emscripten when compiled with MODULARIZE option.

Question ID: 56925313
Questioner asks how to return a byte array from JavaScript to Emscripten/Unity WebAssembly.

Topic #15: Authentication

Questions related to authentication procedures in order to identify and verify if someone or something is in fact who they say they are.

Examples for Topic #15: Authentication

Question ID: 66873168
Questioner asks what best practices are when creating a user management module in his Blazor WebAssembly app using IS4.

Question ID: 59696171
Questioner asks how to fix issue where the sign in is successful but the user is not logged in.

Question ID: 66934600
Questioner asks what the first steps are in creating a website which will communicate with an API using JWT authentication.

Question ID: 64889232
Questioner asks how to restrict page access on user properties.

Topic #16: How WASM works

Questions related to how WASM works in general or one of its specific features.

Examples for Topic #16: How WASM works

Question ID: 56506468
Questioner asks if WASM is safe to store client-side secrets.

Question ID: 44286798
Questioner asks if WASM programs can leak memory.

Topic #17: Database

Questions related to databases covering issues such as performing CRUD operations on structured data which is stored on a server using relational database management systems like SQLite.

Examples for Topic #17: Database

Question ID: 56500709
Questioner asks how to connect to a local MongoDB instance from a WASM module.

Question ID: 66846249
Questioner asks if the database file from SQLite gets downloaded to the user's device or how else would he access it.

Question ID: 66960982
Questioner asks how to bypass CORS to make a call to the Firebase Firestore database from a Blazor WebAssembly app.

Topic #18: User Token

Questions related to create, send and manage tokens used for procedures regarding authentication.

Examples for Topic #18: User Token

Question ID: 61591696
Questioner asks what the best way is to get a Bearer token and pass it on afterwards.

Question ID: 61830255
Questioner asks how to get the raw OAuth token in his Blazor pages.

Question ID: 62306681
Questioner asks how to add roles in JSON Web Tokens with the new version of Blazor WebAssembly.

Question ID: 65219885
Questioner asks how to get the Azure Active Directory access token from the currently logged in user in the Blazor WASM app.

Topic #19: Storage

Questions related to CRUD operations in file systems and local storage in browser as well as memory and cache related issues.

Examples for Topic #19: Storage

Question ID: 45535301
Questioner asks how to read files from the disk using WebAssembly.

Question ID: 62787148
Questioner asks how to protect and encrypt data stored within both the session and local storage in a Blazor WebAssembly app.

Question ID: 57993453
Questioner asks how to access the memory of a WebAssembly module that was loaded by webpack.

4.3.2 Evaluating Question Topics

In section 3.3, we described the process of using the open card sorting approach to find a list of 19 topics in which developers ask security-related WebAssembly questions (e.g. authentication, deployment, tokens, CORS, database, user interface, etc.). We will evaluate the results of this in the following.

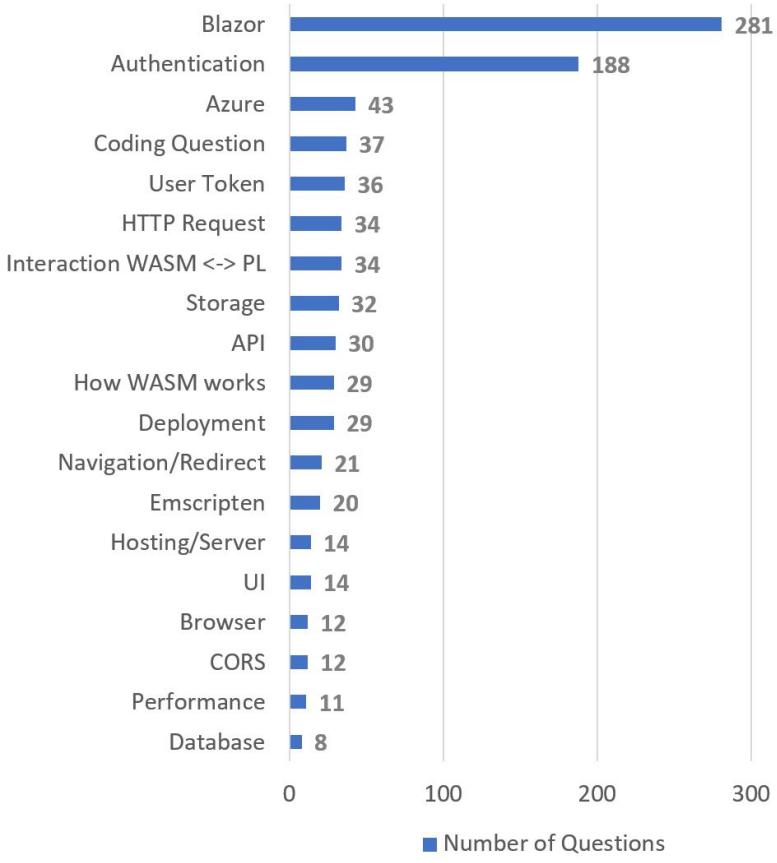


Figure 4.6: Number of questions per topic

First, we look at how many questions were identified for each topic. Figure 4.6 provides an overview. As we expected from the tags, Blazor WebAssembly is the main topic that developers ask about in our dataset, as 78% of all questions are about this topic. Also very popular and far ahead of all others are authentication questions, encountered in 52% of all questions.

In about 10% of all questions, one encounters topics about Microsoft's Azure, specific code implementations, tokens, HTTP requests, and interactions between WASM and other programming languages. The least popular, with a 2-4% share each, are questions about hosting and server, user interfaces, browsers, CORS, performance, and databases.

Since we have more than one topic assigned to a question, in Figure 4.7 we look at how often a pair of any two topics occur in a question to see which topics frequently occur together and are interrelated.

Topics	Blazor	Authentication	Azure	Coding Question	User Token	HTTP Request	Interaction with WASM <-> PL	Storage	API	How WASM Works	Deployment	Navigation/Redirect	Emscripten	Hosting/Server	UI	Browser	CORS	Performance	Database
Blazor	-	188	43	20	36	34	8	15	27	4	27	21		14	14	6	8	7	6
Authentication	188	-	26	7	32	16	1	5	16	1	16	17		5	11	1		5	1
Azure	43	26	-	3	6	4		2	7		7	3					3		1
Coding Question	20	7	3	-			5	4	1		1		3			1	1	1	
User Token	36	32	6		-	3		3	6		1	1					2		
HTTP Request	34	16	4		3	-	1		6		4	1		1		2	2	1	
Interaction with WASM <-> PL	8	1		5		1	-	7	1	11			7		1	1	1	1	1
Storage	15	5	2	4	3		7	-		6			4		1	3	1		2
API	27	16	7	1	6	6	1		-		3		1	1			1		
How WASM works	4	1					11	6		-			6			1			
Deployment	27	16	7	1	1	4			3		-	3		5	1	1		1	
Navigation/Redirect	21	17	3		1	1					3	-			1			2	
Emscripten				3			7	4	1	6			-			2	1		
Hosting/Server	14	5				1			1		5			-			1		
UI	14	11					1	1			1	1			-				1
Browser	6	1		1		2	1	3		1	1		2			-	1		
CORS	8		3	1	2	2	1	1	1				1	1		1	-		1
Performance	7	5		1		1	1				1	2						-	
Database	6	1	1				1	2							1		1		-

Figure 4.7: Number of questions that include a topic in relations with another topic.

Looking at the topics in pairs, we see that for authentication and Azure, among others, there is a high correlation with Blazor, as all their questions were asked in the context of Blazor WebAssembly.

For most topics, we see that the majority of questions were asked in the context of Blazor and authentication. Among the exceptions are for instance “How WebAssembly works” which has a stronger link to “Interaction between WebAssembly and other programming languages”, storage and Emscripten which themselves are also more strongly linked among these topics instead of Blazor and authentication.

In order to find the most frequently asked specific questions, we now examine those pairs that have a particularly high number of questions, which in this case is the pair Blazor and authentication. We want to find out the questions that have been asked the most. We will then do the same for the other most popular pairs, so that we end up with a list of the most frequently asked security-related questions about WebAssembly.

There was a total of 188 questions about Blazor and authentication. Many of them were unique and asked only once. Among those that were asked more than once, the following problems (P1-P6) appeared most frequently:

- P1: Implement third-party authentication
- P2: Authentication works locally in development mode but not after deployment
- P3: User navigation not working after login
- P4: How to customize the pre-existing UI elements of the login screen
- P5: How to assign user roles
- P6: How to secure API endpoints to only give access to a group of selected or authenticated users

Since these problems P1-P6 are not limited to the two topics Blazor and authentication, they are the same problems that can be found when further investigating the popular pairs (Blazor, Azure), (Blazor, User Token), (Blazor, HTTP Request) etc. (see Figure 4.7), as P2 appears frequently in the context of deployment, P3 in navigation and redirect, P4 in user interface, P5 in Azure, and P6 in APIs.

Thus, of the most popular topics, P1-P6 are the most frequently asked questions.

4.3.3 Question Topic Results

Motivation: Now that we know the most popular developer intentions, we would like to go a step further and use this research question to find out what specific topics appear in developers' questions. This will allow us to answer where exactly the issues occur and whether they occur in relation to specific services, tools, libraries, etc.

Approach: In section 3.3, we described how, similar to the developer intentions, we used the open card sorting approach to develop a list of topics describing the problems encountered in the developers' questions. We then assigned each of these topics to the questions in which they occurred. For each specific problem area, we counted the number of questions that were assigned to that topic. We have described the results of this process in subsection 4.3.2. The most popular problems are those topics that have the most questions. Then we continue analyzing the questions within the most popular pairs of topics to see which of them were unique or had been asked several times which would make them the most popular questions we are looking for. Following this process, we find the most frequently and specifically asked security-related WebAssembly questions.

Results: Using the open card sorting approach, we extracted 19 topics that cover the problems addressed in the Stack Overflow questions. In subsection 4.3.2, we then analyzed all our gathered results and found that Blazor was the most popular topic, appearing in 78% of all questions, followed by authentication questions with 52%. Subsequent topics such as Azure, coding questions, tokens, requests, and how WebAssembly interacts with other programming languages were found in about 10% of all questions. When working out the most popular questions, we started with the most popular topics and found six questions that were asked frequently. They include implementing third-party authentication, assigning user roles, how to modify predefined UI elements at login, how to make API endpoints available only to certain users, why the app runs fine locally in developer mode but not after deployment, and why user navigation does not work after login.

Summary: In terms of security-related WebAssembly topics that developers discuss, we found 19 different topics, of which Blazor and authentication were by far the most popular. Among the specific questions asked by developers, we singled out six questions that were most frequently asked in this manner, such as how to add user roles, implement third-party authentication, or how to grant access to API endpoints to only certain users.

4.4 RQ #3: Developers

4.4.1 Evaluating Developers

To find out more about the people behind the questions, let us investigate the developers. Is there a certain group of individuals that asks or answers particularly many questions? If we take the experience of a developer into account by looking at his reputation score, what can we way about the relation between the owner of a question and its writer of the accepted answer if there was one?

The final dataset contains a total of 631 developers that asked 359 questions to which they wrote 357 answers and 1'097 comments (635 to questions and 462 to answers).

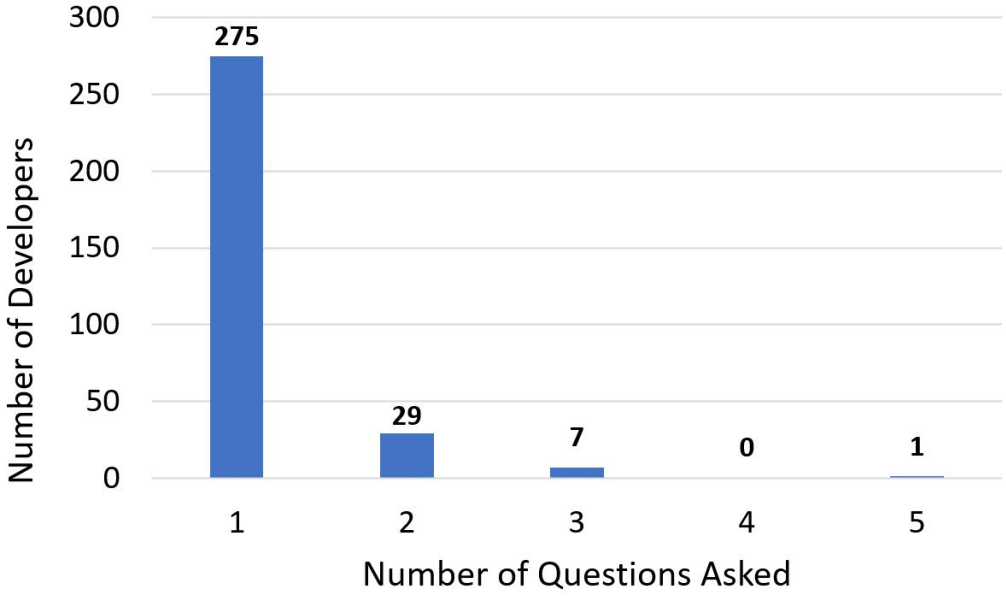


Figure 4.8: Number of developers that asked a given number of questions.

What can we say about the developers asking the questions? The 359 questions have been asked by 312 developers. 275 of them (88%), which is the vast majority, only asked a single question. At the other end of the chart only eight developers asked at least three questions or in other words: 3% of the developers asked 7% of the questions. Looking at their distribution in the above chart in Figure 4.8, we therefore cannot make out a significant group of developers that asked particularly many questions.

What about the developers who provided an accepted answer to a question? In our dataset, 109 developers wrote a total of 157 accepted answers to questions. Looking at the distribution in the chart in Figure 4.9, we still notice that a vast majority of 94 developers (86%) have only written one accepted answer. However, if we look at the other end of the bar chart, we see that the group of developers which wrote three or more accepted answers was quite active. These 6% of the developers are responsible for writing 28% of all accepted answers. As a result, we were able to identify a small group of developers who made a major contribution in regard to providing accepted answers.

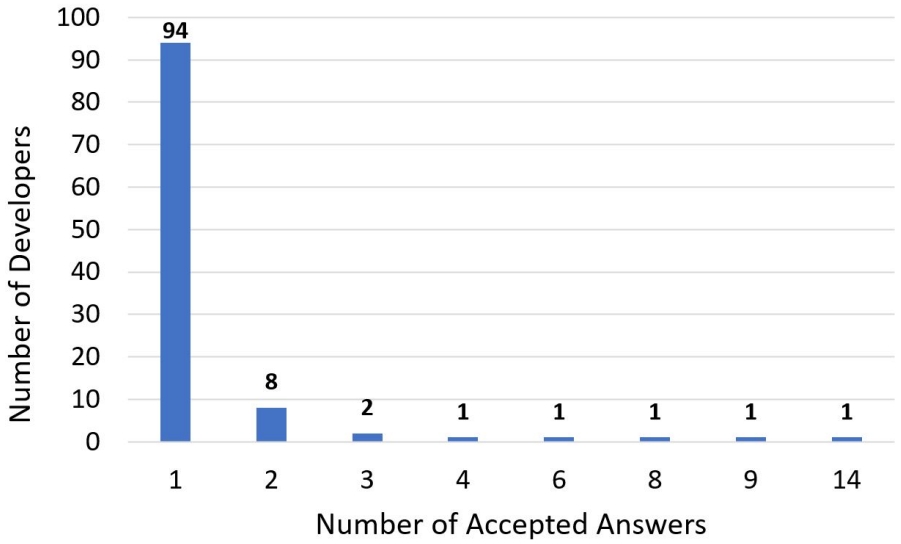


Figure 4.9: Number of developers that wrote an accepted answer to a given number of questions.

To further analyse the developers more easily, we do not want to treat them individually but instead, we will put them together based on similarities in order to handle them in groups. To achieve this, we are going to use the user reputation leagues from StackExchange [3]. Every developer has a reputation score that increases for good behaviour (your question or answer was upvoted; your answer was marked as accepted etc.) and decreases for bad behaviour (your question or answer was downvoted etc.). Although the reputation serves as a rough measurement of how much the community trusts a developer, we can also look at it in a way where the reputation score describes how much experience the user has on the platform. There is not a one-to-one correlation between the reputation score and the experience of a developer since a newly registered user that has a lot of experience will have to start from the very bottom as well, but it still gives some indication to the experience of a developer and it is the best source for us that we can use to group developers based on the knowledge and skills that they acquired.

The user reputation leagues classify developers based on their total reputation whereas a minimum score of one corresponds to the lowest league and a score above 100'000 reputation places the developer in the highest league. We use the same values for our system where a reputation of one or more corresponds to level 1 and 100'000+ to Level 11 as seen in the table below in Figure 4.10.

Min. Total Reputation	Level
0	Level 0
1	Level 1
200	Level 2
500	Level 3
1'000	Level 4
2'000	Level 5
3'000	Level 6
5'000	Level 7
10'000	Level 8
25'000	Level 9
50'000	Level 10
100'000	Level 11

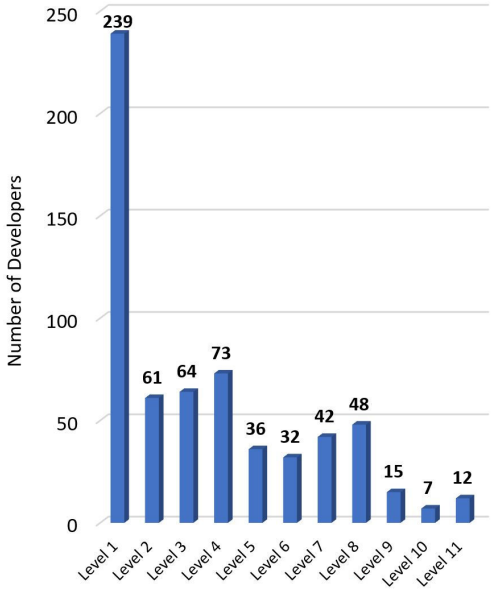


Figure 4.10: *Left table:* Minimum total reputation score required for the corresponding user level according to the reputation leagues on StackExchange. *Right chart:* Number of developers per user level.

Let us look at the distribution of all developers within our dataset (they either wrote a question, answer or comment) after we assigned their corresponding level. From the 631 developers two have deleted their accounts which means we can no longer access their reputation scores, so we exclude them from the above chart. We notice that the majority of developers have a low reputation level (38% have Level 1) and more than half of all developers are Level 3 or lower.

Now let us separate the developers into two groups as seen in Figure 4.11: One where we gather all developers that asked a question and another with only developers that either wrote an answer or comment. The "Questions only" (short "Q only") collection has a relatively higher proportion of developers with a lower level compared to "Answers or Comments only" (short "A/C only") (48% Level 1 for "Q only" versus 36% for "A/C only"). Similarly, we see that the developers of a higher level (Level 7-11) are relatively much rarer in the "Q only" collection than in "A/C only" with 8% in Level 7-11 in "Q only" versus 22% in Level 7-11 for "A/C only".

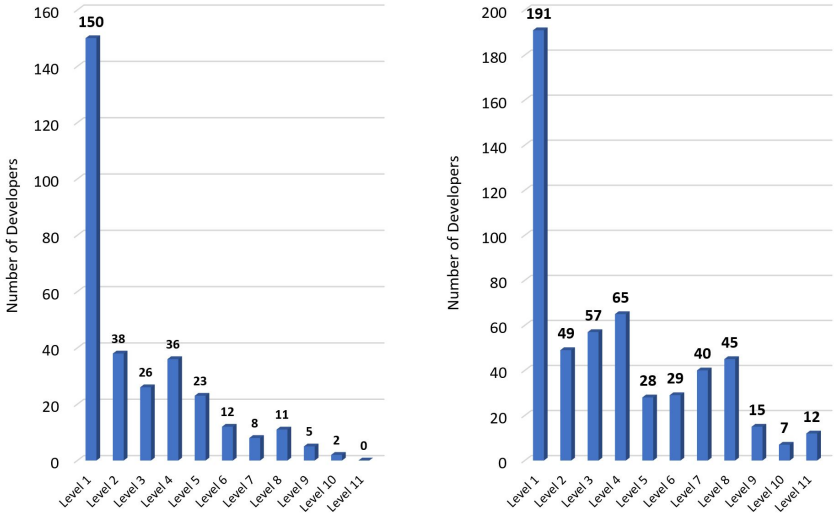


Figure 4.11: *Left chart:* Number of developers per user level that asked a question. *Right chart:* Number of developers per user level that wrote either an answer or a comment.

In the following, we look at the relationship between the questioner and answerer who wrote the accepted answer. In the heatmap in Figure 4.12, each cell shows the number of questions asked by a developer of level Y (along the vertical axis) that received an accepted answer from a developer of level X (along the horizontal axis). Since we only want to consider cases where an interaction between two developers has taken place, we have excluded from this heatmap all questions that have received an accepted answer from the questioner himself. Under the described conditions, the heatmap thus contains 106 different questions.

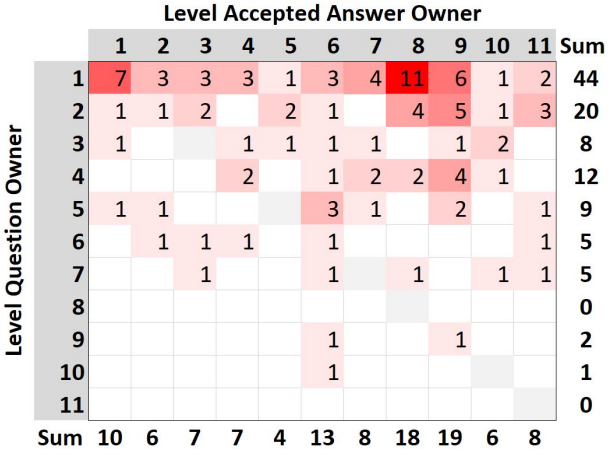


Figure 4.12: Relation between the level of the questioner (vertical axis) and the level of the writer of the question's accepted answer (horizontal axis).

Among the questioners, beginners with level 1-2 are particularly strongly represented, accounting for a significant amount with 64 questions (60%). Among the answerers on the other hand, the more experienced half with level 6-11 has a stronger representation with a share of 72 answers (68%). In 83 questions (78%), the accepted answer comes from a developer with a higher level than that of the questioner. Conversely, eleven questions (10%) were answered by a developer with a lower level than the questioner.

4.4.2 Developer Results

Motivation: Lastly, we would like to focus on the people behind the questions and answers: The developers. We would like to tell if there is a small group within the community of developers who ask security-related WebAssembly questions that is particularly active and asks a lot of questions on the topic or, on the contrary, writes a lot of answers.

Approach: For each question, answer, and comment, there is a user ID property on Stack Overflow, and thus in our dataset, that references the developer who created that content. We now separately collect all user IDs for the questions and accepted answers so that we have the developers who created these posts. Then we count how often each of these IDs occur so we know how often a particular developer created one of these pieces of content. We collate the results as for Figure 4.8 and 4.9 and see if the top end of the scale is a group of developers whose share of all questions asked or accepted answers written is particularly high.

Results: Among the developers who asked a question, there is no group of developers who asked a significant number of questions, as 88% alone asked just one question. This is in contrast to the developers who wrote an accepted answer to a question. There we found a small group of developers (6%) who wrote almost a third (28%) of all accepted answers.

Summary: While among developers who ask questions there is no group that asked a significant proportion of them, as the vast majority only asked a single question, among developers who wrote an accepted answer there is a small group of 6% that is responsible for just under a third of those accepted answers.

5 Threats to Validity

Since many parameters of our work have been affected by internal and external threats, we would like to address them in the following.

5.1 Internal Threats

Internal threats mainly include personal limitations. Depending on how much experience you have in the field, you know how to assess and evaluate the Stack Overflow questions differently. Also, the approach, how the data was evaluated, are influential on the achieved results. The final categories of the questions asked as well as the actual topics and issues of the questions have therefore been influenced by our chosen open card sorting approach and are consequently also susceptible to personal views and tendencies on how we prioritize and evaluate certain aspects of a question differently.

5.2 External Threats

There are also some external threats that have had a decisive impact on our research work. An originally different selection of security-related keywords could have resulted in a different dataset where our evaluation steps could have lead to different results. Also, our choice of data source with Stack Overflow influenced our results, as when considering other platforms, there is a possibility of encountering other developer groups whose questions differ from those on Stack Overflow.

6 Conclusion

In this bachelor thesis, we performed an analysis on 359 security-related WebAssembly questions on Stack Overflow. We found that most of the questions on this topic are help on bug fixes and guidance on how to implement a particular feature, as both appear in about 50% of the questions. Overall, we identified five relevant question types using the open card sorting approach.

Second, we examined the topics that were covered in the questions and came up with a list of 19 different topics. Blazor and authentication were by far the most widespread of these. For the topics with the most questions, we then attempted to find those specific questions that were asked the most. We found six questions that were most common within security-related WebAssembly questions.

Regarding developers, among those who wrote accepted answers, we found a small group (6%) that captured a relatively large proportion of them (28%). This was not the case among the questioners themselves.

The release of Blazor WebAssembly in particular has led to a very strong and so far consistently high number of security-related WebAssembly questions on Stack Overflow, with questions about authentication standing out in particular.

7 Anleitung zu wissenschaftlichen Arbeiten

In this chapter, we focus on the practical implementations that we have developed as part of our research work. After a brief overview of the technical requirements, we look over the different endpoints and data objects from the StackAPI which we then use in our Python scripts to collect all Stack Overflow questions, answers and comments and later also perform statistical evaluations on.

7.1 GitHub Repository

To share the resources from our research work (including the list of security related keywords, Python scripts to request data from StackAPI and perform statistical analysis, JSON files from the questions, answers and comments from our dataset as well as our notes that we took when processing the questions) we created a GitHub repository to make them publicly available:

`https://github.com/PascalMarcAndre/WebAssembly-Security`

7.2 Setup and Installation Guide

In order to run the scripts featured in this chapter, a recent installation of Python (between version 2.7-3.7) is required. Python can be downloaded from its official download page [2] using a guide from Real Python [6] as guidance. The additional StackAPI library can be installed using `pip install stackapi` or following other installation guides on its official page [4].

7.3 StackAPI

StackAPI [4] serves as a Python wrapper for the official Stack Exchange API [1]. Using this library, we can write simple Python scripts to access various endpoints of the Stack Exchange API to request its content. In this section, we mostly focus on the endpoints and data objects (JSON of questions, answers and comments) that are relevant to our work to see how they are structured and what properties they provide.

7.3.1 Question JSON

Whenever we request data for a question with full details from the StackAPI (e.g. when using the `/questions/{ids}` endpoint where `ids` is to be replaced with the question id), it will be returned in the following basic JSON structure:

```
{
  "tags": string[]
  "owner": {
        "reputation": number,
        "user_id": number,
        "user_type": string,
        "profile_image": string,
        "display_name": string,
        "link": string
    },
    "is_answered": boolean,
    "view_count": number,
    "accepted_answer_id": number,
    "answer_count": number,
    "score": number,
    "last_activity_date": number,
    "creation_date": number,
    "last_edit_date": number,
    "question_id": number,
    "content_license": string,
    "link": string,
    "title": string,
    "body": string
}
```

Listing 1: Shows the structure and JSON of a question object

A brief description of all question properties assigned to each question object can be found in table 7.1. The `owner` property describes the user object of the developer who wrote the question and has other nested properties described in table 7.2.

Property	Description
tags	List of keywords that have been added to the question
owner	Developer that is the question owner who originally created this question
is_answered	Boolean representing whether the issues from the question has been resolved or not
view_count	Number of times the question has been viewed
accepted_answer_id	Unique id number of the answer that has been marked as the "accepted" one
answer_count	Number of answers that the question received
score	Number of upvotes minus number of downvotes that the question received
last_activity_date	Timestamp of the most recent activity to the question
creation_date	Timestamp of when the question was originally posted / created
last_edit_date	Timestamp of when the question was last edited
question_id	Unique id number identifying this question
content_license	Type of license that was attributed to the question (e.g. "CC BY-SA 3.0")
link	URL of the question
title	Title of the question
body	Content including problem description of the question

Table 7.1: Brief descriptions of all question properties

Table 7.2 describes the JSON properties that are returned whenever data for a user is returned from the StackAPI. Requesting data for questions, answers, comments etc. always include details about its owner who posted the content which contain the following properties describing this user.

Property	Description
reputation	Reputation score according to the Stack Exchange User Reputation Leagues
user_id	Unique id number identifying this developer
user_type	Either "unregistered", "registered", "moderator", "team_admin" or "does_not_exist"
profile_image	URL of the profile image picture
display_name	Displayed user name of the developer
link	URL of the developer's profile page

Table 7.2: User property descriptions

An example of a JSON returned by the StackAPI containing full details of a question can be found in listing 2. For simplicity, we shortened the value of the `body` property which would otherwise contain the full content of the question body.


```
{
    "tags": [
        "security",
        "electron",
        "webassembly"
    ],
    "owner": {
        "reputation": 713,
        "user_id": 1508479,
        "user_type": "registered",
        "accept_rate": 64,
        "profile_image": "https://i.stack.imgur.com/K8WcV.jpg?s=128&g=1",
        "display_name": "Vinay",
        "link": "https://stackoverflow.com/users/1508479/vinay"
    },
    "is_answered": true,
    "view_count": 403,
    "accepted_answer_id": 56507571,
    "answer_count": 1,
    "score": 0,
    "last_activity_date": 1563201029,
    "creation_date": 1559997717,
    "last_edit_date": 1563201029,
    "question_id": 56506468,
    "content_license": "CC BY-SA 4.0",
    "link": "https://stackoverflow.com/questions/56506468",
    "title": "Is wasm safe to store client side secrets?",
    "body": "The security context of my question is as follows: (...)"
}
```


Listing 2: JSON example for Question ID 56506468

7.3.2 Answer JSON

Whenever we request data for an answer with full details from the StackAPI (e.g. when using the `/answers/{ids}` endpoint where `ids` is to be replaced with the answer id or when requesting all answers for a question using the `/questions/{ids}/answers` endpoint where `ids` is to be replaced with the question id), it will be returned in the following basic JSON structure:

```
{
    "owner": User,
    "is_accepted": boolean,
    "score": number,
    "last_activity_date": number,
    "creation_date": number,
    "answer_id": number,
    "question_id": number,
    "content_license": string,
    "body": string
}
```

Listing 3: Shows the structure and JSON of an answer object

A brief description of all answer properties assigned to each answer object can be found in table 7.3. The `owner` property describes the user object of the developer who wrote the answer and has other nested properties as previously shown in table 7.2.

Property	Description
owner	Developer that is the answer owner who originally created this answer
is_accepted	Boolean representing whether this answer solved the issue or not
score	Number of upvotes minus number of downvotes that the answer received
last_activity_date	Timestamp of the most recent activity to the answer
creation_date	Timestamp of when the answer was originally posted / created
answer_id	Unique id number identifying this answer
question_id	Unique id number identifying the question to which this answer was posted
content_license	Type of license that was attributed to the answer (e.g. “CC BY-SA 3.0”)
body	Content which describes the solution of this answer

Table 7.3: Brief descriptions of all answer properties

An example of a JSON returned by the StackAPI containing full details of an answer can be found in listing 4. For simplicity, we shortened the value of the `body` property which would otherwise contain the full content of the answer body.


```
{
    "owner": {
        "reputation": 7044,
        "user_id": 7670262,
        "user_type": "registered",
        "profile_image": "https://i.stack.imgur.com/3zdDY.jpg?s=128&g=1",
        "display_name": "Azeem",
        "link": "https://stackoverflow.com/users/7670262/azeem"
    },
    "is_accepted": false,
    "score": 2,
    "last_activity_date": 1586607164,
    "creation_date": 1586607164,
    "answer_id": 61156588,
    "question_id": 61015985,
    "content_license": "CC BY-SA 4.0",
    "body": "How to detect (at JS side) an 'uncaught exception' (...)"
}
```


Listing 4: JSON example for Answer ID 61156588

7.3.3 Comment JSON

Whenever we request data for a comment with full details from the StackAPI (e.g. when using the `/comments/{ids}` endpoint where `ids` is to be replaced with the comment id or when requesting all comments for an answer using the `/answers/{ids}/comments` endpoint where `ids` is to be replaced with the answer id or when requesting all comments for a question using the `/questions/{ids}/comments` endpoint where `ids` is to be replaced with the question id), it will be returned in the following basic JSON structure:

```
{
    "owner": User,
    "edited": boolean,
    "score": number,
    "creation_date": number,
    "post_id": number,
    "comment_id": number,
    "content_license": string,
    "body": string
}
```

Listing 5: Shows the structure and JSON of a comment object

A brief description of all comment properties assigned to each comment object can be found in table 7.4. The `owner` property describes the user object of the developer who wrote the comment and has additional nested properties as previously shown in table 7.2.

Property	Description
owner	Developer that is the comment owner who originally created this comment
edited	Boolean whether this comment has been edited or not
score	Number of upvotes minus number of downvotes that the comment received
creation_date	Timestamp of when the comment was originally posted / created
post_id	Unique id number identifying the post (question or answer) to which this comment was posted
comment_id	Unique id number identifying this comment
content_license	Type of license that was attributed to the comment (e.g. “CC BY-SA 3.0”)
body	Content of the comment

Table 7.4: Brief descriptions of all comment properties

If the comment is a response to another comment, the JSON will include an additional property called `reply_to_user` which is similar to the `owner` property but includes the nested user properties as seen in listing 7.2 to describe the developer to which this comment was addressed to.

An example of a JSON returned by the StackAPI containing full details of a comment can be found in listing 6. For simplicity, we shortened the value of the `body` property which would otherwise contain the full content of the comment body.


```
{
    "owner": {
        "reputation": 13400,
        "user_id": 2940908,
        "user_type": "registered",
        "accept_rate": 60,
        "profile_image": "https://i.stack.imgur.com/E79WU.jpg?s=128&g=1",
        "display_name": "agua from mars",
        "link": "https://stackoverflow.com/users/2940908/agua-from-mars"
    },
    "reply_to_user": {
        "reputation": 96,
        "user_id": 6830857,
        "user_type": "registered",
        "profile_image": "https://i.stack.imgur.com/sXQMo.jpg?s=128&g=1",
        "display_name": "Kasper Olesen",
        "link": "https://stackoverflow.com/users/6830857/kasper-olesen"
    },
    "edited": false,
    "score": 1,
    "creation_date": 1574109159,
    "post_id": 58875620,
    "comment_id": 104106310,
    "content_license": "CC BY-SA 4.0",
    "body": "Yes, that act as if (user.IsAuthorized) (...)"
}
```


Listing 6: JSON example for Comment ID 104106310

7.4 Python Scripts

In this section we present some of our most important Python scripts that we used to query the data via the StackAPI at the beginning of the project as well as to perform statistical analysis of the dataset at a later stage of our research work.

7.4.1 Get all Search Results using Keywords

The purpose of our first Python script as seen in listing 7 was to fetch all Stack Overflow questions matching the search term *WebAssembly "keyword"* where the *"keyword"* gets replaced one by one with one of the keywords from our separate text file containing all variations of our security-related keywords. Once we have a list of all unique questions matching at least one of our search terms, we save all the available and necessary data of the retrieved question properties in a CSV file for further processing.

For simplicity, we removed some lines from the script that cover similar repeating actions for the different question properties. In the version below, only the question properties of the title and score are included, but we commented the lines of code where similar actions for the remaining question properties should be performed.

```
# Import required libraries
from stackapi import StackAPI
import datetime as dt
import csv
import json

# Create list that holds all keywords from our separate keywords text file
keywords = []
# Create empty list to which we will add all unique questions from the search results
questions = []

# Parse text file with all keywords and store them in a list
with open('../keywords/keywords.txt') as keywords_file:
    for line in keywords_file:
        keyword = line.strip("\n")
        if len(keyword) > 0:
            keywords.append(keyword)

# Setup StackAPI to use Stack Overflow (=SO) as its source
SITE = StackAPI('stackoverflow')

# Search questions on SO with "WebAssembly [keyword]" and print to CSV-file
with open('all_questions.csv', 'w', encoding='utf-8', newline='') as questions_file:
    # Create CSV writer to convert data into delimited strings
    writer = csv.writer(questions_file, delimiter=';')

    # Write header row with labels/titles
    writer.writerow(['Index','Title','Score'
        # CUT LINES: Add header for other question properties here
    ])

```

```
    # Iterate over keywords and request questions to create final of unique questions
    for keyword in keywords:
        # Build search term using "WebAssembly" and the currently selected keyword
        searchTerm = 'webassembly ' + keyword

        # Use StackAPI endpoint "search/advanced" to fetch matching questions
        # Use 'q' parameter with search term to match any property (title, body...)
        new_questions = SITE.fetch('search/advanced', q=searchTerm)
        # Possible params: sort='relevance, has_more=1, max_pages=1000, page_size=100

        # Check if any new result was matched in previous search
        for new_question in new_questions['items']:
            # We assume new question is not in our list yet (not an old questions)
            old_question = False

            # If an old question matches new question's ID, we mark new question as old
            for question in questions:
                if question['question_id'] == new_question['question_id']:
                    old_question = True
                    break

            # If new question is not an old/existing one, we add it to questions list
            if not old_question:
                questions.append(new_question)

    # Iterate over list of unique SO questions and print their data into THE CSV file
    for index, question in enumerate(questions, start=1):
        # Pre-process some data including dates and nested JSON data
        creation_date
          = dt.date.fromtimestamp(question['creation_date']).strftime('%Y-%m-%d')
        last_activity_date
          = dt.date.fromtimestamp(question['last_activity_date']).strftime('%Y-%m-%d')
        owner = question['owner']

        q_title = ''
        q_score = ''
        # CUT LINES: Add more variables for other question properties here

        try:
            q_title = question['title']
        except KeyError:
            print(str(index) + ' q_title')
        try:
            q_score = question['score']
        except KeyError:
            print(str(index) + ' q_score')
        # CUT LINES: Check values of other question properties here

        # Extract question properties from JSON-fields (only write if no error exists)
        writer.writerow([index,q_title,q_score
            # CUT LINES: Print other other question properties to CSV file here
        ])

# Print raw JSON data into separate file for further inspection
with open('all_questions.json', 'w', encoding='utf-8') as json_file:
     json.dump(questions, json_file, indent=4)

# Success message
print('Printed data of ' + str(len(questions)) + ' questions to CSV-file.')
```

Listing 7: Python script to fetch all SO questions matching *WebAssembly "keyword"*

7.4.2 Get all Individual Questions

Based on the list of question ids and some initial details that we had gathered during the question search using the script in listing 7, we used the following script in listing 8 to request full details on each of these questions. This script saves all data for a question within a separate JSON file named after the question's id number.

```
# Import required libraries
from stackapi import StackAPI
import json

# Setup StackAPI to use Stack Overflow as its source
SITE = StackAPI('stackoverflow')

# Open text file with all the ids from the relevant questions
with open('../data_files/questions/all_relevant_question_id.txt') as id_file:
    # Load a list of question ids from the text file
    lines = [line.rstrip() for line in id_file]

    # Iterate over all question ids
    for line in lines:
        # Use StackAPI to fetch full question details (incl. body)
        # Endpoint: /questions
        new_question = SITE.fetch(
            'questions',
            ids={line},
            filter='withbody'
        )

        # Puts all questions into JSON-file named after its question id
        with open('../data_files/questions/json_files/' + line + '.json',
                  'w', encoding='utf-8') as question_file:
            json.dump(new_question['items'][0], question_file, indent=4)
```

Listing 8: Python script to fetch full details for a single question

With this script, we collected full details for all 359 questions using the StackAPI endpoint `/questions/{ids}`, where `ids` gets replaced by the question ids from the relevant questions.

7.4.3 Get all Individual Answers

For our analysis of the answers we used the following Python script to fetch all individual answers for each Stack Overflow question within our dataset. This script saves all answers for a question within a separate JSON file named after the question's id number.

```
# Import required libraries
from stackapi import StackAPI
import json

# Setup StackAPI to use Stack Overflow as its source
SITE = StackAPI('stackoverflow')

# Open text file with all the ids from the relevant questions
with open('../data_files/questions/all_relevant_question_id.txt') as id_file:
    # Load a list of question ids from the text file
    lines = [line.rstrip() for line in id_file]

    # Iterate over all question ids
    for line in lines:
        # Use StackAPI to fetch full question details (incl. body)
        # Endpoint: /questions/{ids}/answers
        new_answer = SITE.fetch(
            'questions/{ids}/answers',
            ids={line},
            sort='creation',
            filter='withbody'
        )

        # Puts all answers into JSON-file named after its question id
        with open('../data_files/answers/json_files/' + line + '.json',
                  'w', encoding='utf-8') as answer_file:
            json.dump(new_answer['items'], answer_file, indent=4)
```

Listing 9: Python script to fetch all individual answers to all our questions

With this script, we collected full details for all 357 answers that have been written to our 359 SO questions using the StackAPI endpoint `/questions/{ids}/answers` where `ids` gets replaced by the question ids from our relevant questions.

Full details for all comments that have either been made to questions or answers have been collected in a very similar way by requesting the data from the corresponding StackAPI endpoint.

7.4.4 List Tags with Number of Occurrences

During our general analysis of our dataset we used the following Python script to list all tags along with their number of questions in which the tag occurred in descending order and export them into a CSV file:

```
# Import required libraries
from operator import itemgetter
import json import csv

# Create empty list that holds all tags and their total number of appearances
tags = []

# Open text file containing all question ids of the relevant questions to be processed
with open('../../data_files/questions/all_relevant_question_id.txt') as text_file:
    # Each 'line' corresponds to one question id in the text file to be processed
    for line in text_file:
        # Open the JSON-file of the question from the current question id ('line')
        with open('../../data_files/questions/json/' + line.strip() + '.json')as file:
            # Loads the JSON-file content into the 'question' variable
            question = json.load(file)

        # Iterate over all tags from the current question
        for new_tag in question['tags']:
            # Assume this tag appears for the first time
            is_new_tag = True

            # Iterate over all tags within our 'tags' list
            for existing_tag in tags:
                # If a tag in our list matches the currently processed tag...
                if existing_tag[0] == new_tag:
                    # ...increase its counter of questions by one
                    existing_tag[1] += 1
                    # ...update that the currently processed tag is not a new one
                    is_new_tag = False
                    break

            # If the currently processed tag is a new one...
            if is_new_tag:
                # ...add it to our list of tags and set its question count to 1
                tags.append([new_tag, 1])

    # Sort all tags according to their count
    tags.sort(key=itemgetter(1))
    # Reverse order to have results in descending order
    tags.reverse()

    # Open CSV-file to write the results
    with open('all_tags.csv', 'w', encoding='utf-8', newline='') as csv_file:
        # Create CSV writer to convert data into delimited strings
        writer = csv.writer(csv_file, delimiter=';')
        # Set column headers to 'Tag' and 'Count'
        writer.writerow(['Tag','Count'])

        # Iterate over all identified tags
        for tag in tags:
            # Write each tag with its count to the CSV writer
            writer.writerow([tag[0],tag[1]])
```

Listing 10: Python script to list all tags with number of occurrences in descending order

7.4.5 List Number of Questions per Quarter

To find out how popularity on security-related WebAssembly questions changed over time, we used the following Python script during our general analysis to count the number of questions that have been asked in each quarter of the year.

```
# Import required libraries
import json
import datetime as dt
import pandas as pd

# Create empty list that holds all quarters with its number of questions
quarters = []

# Open text file containing all ids of questions to be processed
with open('../question_id.txt') as text_file:
    # Each 'line' corresponds to one question id in the text file
    for line in text_file:
        # Open the JSON-file of the question related to the current id
        with open('../questions/' + line.strip() + '.json')as json_file:
            # Loads the JSON-file content into the 'question' variable
            question = json.load(json_file)

            # Creates date-variable from the question's creation date
            date = dt.date.fromtimestamp(question['creation_date'])

            # Finds the year from the creation date timestamp
            year = date.strftime('%Y')
            # Finds the quarter number from the creation date timestamp
            quarter_number = pd.Timestamp(date).quarter

            # Build quarter name based on previously found year and quarter
            quarter_name = str(year) + "-Q" + str(quarter_number)

            # Assume this is the first question within this quarter
            is_new_quarter = True

            # Iterate over all quarters in our list
            for quarter in quarters:
                # If a quarter in our list matches the identified quarter...
                if quarter[0] == quarter_name:
                    # ...increase its counter of questions by one
                    quarter[1] += 1
                    # ...update that the identified quarter is not a new one
                    is_new_quarter = False
                    break

            # If the currently identified quarter is a new one...
            if is_new_quarter:
                # ...add it to our list of quarters and set its counter to 1
                quarters.append([quarter_name, 1])

    # Print the results of all quarters
    print(quarters)
```

Listing 11: Python script to count the number of question per quarter

Bibliography

[1] Official stack exchange api. URL: https://api.stackexchange.com/.

[2] Python download page. URL: https://www.python.org/downloads/.

[3] Stack exchange user reputation leagues. URL: https://stackexchange.com/leagues/1/week/stackoverflow.

[4] Stackapi. URL: https://stackapi.readthedocs.io/en/latest/.

[5] Jeff Atwood. Introducing stackoverflow.com, 2008. Article at https://blog.codinghorror.com/introducing-stackoverflow-com/.

[6] Real Python. Python 3 installation guide for various platforms. URL: https://realpython.com/installing-python/.