Decoding and Answering Developers' Questions about Web Services Managed by Marketplaces

Siddhi Baravkar*, Chi Zhang*, Foyzul Hassan*, Long Cheng†, and Zheng Song*
* Department of Computer and Information Science, University of Michigan at Dearborn, USA
† School of Computing Clemson University, USA
{siddhib, zhchi, foyzul}@umich.edu, lcheng2@clemson.edu, zhesong@umich.edu

2024 IEEE International Conference on Software Services Engineering (SSE) | 979-8-3503-6851-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/SSE62657.2024.00037

***Abstract*—Service registry, a key component of the service-oriented architecture (SOA), aids software developers in discovering services that meet specific functionality requirements. Recent years have witnessed the transition from the traditional service registries to its successor, the *Service Marketplaces*, which involves deeper engagement in the SOA software lifecycle and offers additional features, such as service request delegation and monitoring of services' Quality of Service (QoS). However, by analyzing developers' questions posted on online Q&A forums, we found that many developers struggle with such transition, leading to development inefficiencies and even security vulnerabilities. This paper presents the first empirical study aimed at uncovering the issues developers face with marketplaces, particularly those arising from the transition. Through a meticulous process of manually labeling and analyzing developers' questions, we develop a taxonomy of these issues, summarize the impacts caused by the transition, and provide actionable suggestions to App developers, service providers, and marketplaces. Utilizing the labeled questions and our insights, we fine-tune a Large Language Model (LLM) for providing answers to similar questions raised by developers and helping service providers and marketplaces extract useful information from these questions, such as service outages and key leakages. Our evaluation of the model's performance in answering and extracting pertinent information from a set of real-world questions demonstrates its effectiveness: it accurately classified 85% of the queries and successfully identified 88% of service names and 77% of key leakages. As the first empirical study in this domain, this work not only aids developers in navigating the transition more effectively but also sheds light on the under explored issue of service registry evolution, offering valuable insights for researchers.**

***Index Terms*—Service Marketplace, Q&A forums, Taxonomy, Large Language Model**

I. Introduction

Service-oriented architecture (SOA) has been widely adopted in modern software systems. It provides application developers a flexible and efficient framework for accessing remote domain-specific data and computational-intensive technologies hosted by third party service providers. Examples of such services include face recognition, text generation, translation [1], [2], and querying for real-time ticket/flight/game information [3]. With the blooming of Machine Learning as a Service (MLaaS) [4], SOA will be even more important, with an estimated market size of 26.5 billion USD by 2027 [5].

To connect service providers and application developers, service registries serve as an important component of SOA. It stores various information about available services and helps developers discover services with the required functionalities [6]. We observe that in recent years, the service registry has experienced a major migration in terms of its functionality, from traditional **static service registries** to the new service **marketplaces**. For example, RapidAPI is the most widely used marketplace, serving over 4 million users globally [1].

Unlike traditional service registries, which direct app developers to the websites of service providers, RapidAPI offers a streamlined, one-stop solution for app developers. On RapidAPI, developers can subscribe to services, send requests, and manage their service payments all in one place, simplifying the process of integrating and using external services. This major functionality change introduces several sequential adjustments to the overall workflow of using service registries.

However, these workflow differences are not widely known, which could potentially impact developers using the marketplaces. Our review of inquiries on StackOverflow(SO), a leading online question-and-answer platform for software developers, uncovered numerous instances where developers were puzzled about using RapidAPI. For example, one question that has been viewed 22K times asks about failing to access a free-to-use service via RapidAPI, despite possessing a valid key[2]. This confusion arises from a fundamental difference in how RapidAPI operates compared to traditional service registries. Traditionally, developers receive an invocation key only after subscribing to a specific service. Conversely, RapidAPI issues a universal key that grants access to all its services, yet mandates a subscription to individual services before they can be utilized. This discrepancy highlights a knowledge gap among application developers, who are accustomed to the conventional model of service registries. Bridging this gap could not only help service providers and application developers more efficiently use marketplaces but also provide marketplace owners insights to improve their platform design.

To the best of our knowledge, although previous studies [7], [8] have summarized developers' concerns and doubts when using traditional service registries, there hasn't been any research conducted on the challenges encountered by developers when using service marketplaces. The differences in their working mechanism require a dedicated study to understand the challenges faced by developers, especially those caused by the migration. The study will also generate insights for us to

[1]https://rapidapi.com/company/
[2]https://rb.gy/59mo95

979-8-3503-6851-2/24/$31.00 ©2024 IEEE
DOI 10.1109/SSE62657.2024.00037

train an AI-based automated tool that assists developers and researchers in mitigating these challenges. In particular, this study seeks answers to the following research questions:

- **RQ1:** What are the differences in workflow between traditional service registries and service marketplaces?
- **RQ2:** What are the developers' challenges, especially those due to migration from traditional service registry to service marketplace?
- **RQ3:** How can an AI-based automated tool assist developers and researchers in mitigating these challenges?

This paper is structured as follows: Section II introduces the background and related works; Section III introduces our approaches towards answering the questions; Section IV explains our findings, answers the RQs, and gives actionable suggestions to stakeholders; Section V concludes this paper. Additionally, we have open-sourced the dataset of questions collected and the process of fine-tuning the language model, which can be found in [3].

II. Background and Related Work

This section introduces the background of our research, along with a review of existing literature in this field.

A. Service Registry

A service registry is an important component of the service-oriented architecture. It was originally designed to help developers search and find the services that fit their functionality requirements. Our definition of the two generations of service registries was inspired by [9], which was published almost ten years ago. They categorized the first generation as an information gateway between service providers and consumers, and the second generation as "integrated throughout the entire software life cycle."

Following this definition, we categorize *service marketplaces* as the second generation of service registry, as they serve as a delegation between App developers and service providers. Three basic user roles interact with marketplaces:

- **Service Provider:** The service providers are individuals or organizations offering services to third-party users. They run the servers that host the services, which usually require authentication to be accessed. Service providers register their services in the service registry, to be used by App developers.
- **Application Developer:** App developers discover services by interacting with the service registry. They select the service that best fits their software's requirements, develop service invocation logic, and add the authentication keys by which they pay for service usage.
- **End User:** End users are the consumers of the software, by whom the service requests are sent. End users usually have no knowledge and capability to modify the software they use.

[3]https://anonymous.4open.science/r/FineTuning-72E4/

B. Understanding Developers' Concerns

To the best of our knowledge, this paper is the first to study developers' questions about service marketplaces. The most related work studied developers' concerns related to traditional registries [7] by analyzing SO questions related to ProgrammableWeb and APIGuru. It outlines the taxonomy for traditional registries as "Authorization", "Function", "Documentation", and "Others". Correspondingly, we devised a different taxonomy for the service marketplace: our taxonomy includes the issues caused by the differences in their workflow (i.e., delegation), including security, usability, documentation, performance, etc.

Several works studied developers' concerns and patterns related to web service usage. Similar to our approach, many [8], [10], [11] analyzed developers questions posted on SO. Different from our approach that uses "RapidAPI" as keywords because all services need to be invoked via the uniform RapidAPI interface, they filtered the related posts either by pre-selecting services as keywords or by "service" hashtags. Some of their findings overlap with ours, such as QoS issues in services and issues related to API updates. Different from these approaches, our work further reveals developers' issues caused by marketplaces.

Our paper explained how the shift from a traditional service registry to a service marketplace has confused developers, introduced a developers' questions taxonomy as well as developed an LLM-based AI-assisted tool to identify and address developers' concerns related to API marketplaces to ensure smoother service utilization. Similar ideas of automatically answering questions by analyzing existing questions and answers have been widely explored [12]–[14], especially with the recent blooming of generative AI. This paper is the first to apply such a technique in the domain of web service, which makes it a useful tool for SOA developers and stakeholders.

III. Methodology

This section discusses data collection and the methodology for all three research questions mentioned in Sec. I. As illustrated by Figure 1, for *RQ1*, we explored the workflows of widely used service registries, categorized them into two generations, and summarized the major differences in their workflows. For *RQ2*, we analyzed and labeled the questions posted on various Q&A forums, by which we created a taxonomy. Lastly, for *RQ3*, we developed an LLM-based tool by fine-tuning and evaluating its effectiveness on a test dataset.

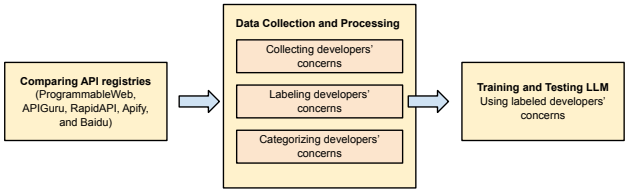

Fig. 1. Methodology Workflow

Features/Platforms	ProgrammableWeb	APIGuru	RapidAPI	Apify	Baidu
API Coverage	24000+ (before 2022)	2529	40000+	1558	730
Invocation Examples	No	No	19	3	6
QoS metrics	No	No	Yes	Yes	No
Request Delegation	No	No	Yes	Yes	Yes
Doc. by Service Providers	No	No	Yes	Yes	Yes
Generation	Static	Static	Marketplace	Marketplace	Marketplace

TABLE I
STATISTICS OF API REGISTRIES

A. Comparing Two Generations of Service Registry

We chose widely-used service registries, including ProgrammableWeb, APIGuru[4], RapidAPI, Apify [5], Baidu API [6], as the targets of our study. Table I compares the statistics of these registries. RapidAPI and programmableWeb host more services than the other three.

We further explored the features of these registries by using them, both as service providers listing services and as App developers searching for services and invoking them. For ProgrammableWeb which has been retired, we explored its features by reading papers and tutorials. Initially, we categorized them into two groups: static service registries and service marketplaces, distinguishing them based on their common patterns, w.r.t. 1) delegating end-user requests; 2) measuring and displaying the QoS of services; 3) allowing service providers to customize documentation; and 4) providing example codes for invoking services in various programming languages. Sec. IV further analyzes the differences in their workflows in detail.

B. Collecting and Categorizing Developers' Questions

To answer *RQ2*, we collected data from online Q&A platforms and categorized them. In particular, we used these platforms' search interface to find data related to our study.

1) Data Collection: The two most popular online Q&A platforms for App developers are SO [7] and the issue panel of GitHub repositories [15]. We also included data from G2 [8], a platform for developers to publish reviews related to software and services. Among the three marketplaces we mentioned earlier, we chose to only study questions related to RapidAPI and Apify, as Baidu API is targeted for Chinese software developers and we cannot find questions about it in English-speaking online communities.

To get the most recent questions related to our topic, we used "RapidAPI" and "Apify" as keywords to search on these online communities. Altogether, we collected approximately 2000 random questions.

2) Data Categorization: We conducted a manual examination of each question to gain a comprehensive understanding of developers' doubts and concerns. Three of the authors, each with 3 to 5 years of industrial software development experience, individually reviewed each question and identified recurring patterns. Beyond just summarizing developers' doubts and concerns, we also inspected the questions for technical issues overlooked by developers and delved into the root causes.

As the next step, we conducted a group discussion to reach an agreement on the patterns, root causes, and impacts of developers' questions. We applied majority voting to resolve the disagreement in our opinion. We assigned names to each category of questions and labeled the questions themselves accordingly, aligning them with the commonly observed issues.

Upon categorizing the data, it was clear that some questions—such as those about coding issues with various HTTP libraries and frameworks, project ticket trackers created by users to track their work progress, and discussions not pertinent to API marketplaces—did not align with the focus of our research. To determine the relevance of a question, we employed a straightforward criterion: would a developer encounter the same issue using a traditional registry? If the answer was affirmative, we deemed the question irrelevant and excluded it from our analysis. This filtering process resulted in a refined dataset comprising 103 records.

Among all the questions we collected, a majority of them (95%) are related to simple coding, e.g., " *Can't use RapidAPI with Retrofit (HTTP client for Android and Java)...*" Given that many software developers first encounter HTTP requests early in their careers [16], the prevalence of fundamental coding inquiries suggests that a substantial number of RapidAPI users are novice software engineers. This observation underscores how marketplaces like RapidAPI empower beginning developers to incorporate complex functionalities into their applications, democratizing access to sophisticated software development tools.

C. Developing an LLM Tool to Answer Developers' Questions

After categorizing the data, we discovered that despite the variety of ways developers pose questions, the underlying issues often fall into a few distinct categories. This revelation led us to utilize our categorized questions, complete with their identified root causes and proposed solutions, to fine-tune a Large Language Model (LLM). The enhanced LLM can now use the insights from this analysis to answer developers' questions with greater accuracy. Additionally, it has the capability to sift through developers' inquiries to track performance problems related to specific APIs and detect potential key leakages. For instance, if the refined model spots a question

[4]https://apis.guru/
[5]https://apify.com/
[6]https://apis.baidu.com/, in Chinese
[7]stackoverflow.com
[8]g2.com

highlighting performance issues with a certain API, it can flag this to the attention of either the service provider or the platform, facilitating a more proactive approach to issue resolution.

1) Data for Training the LLM Tool: As 95% of all questions are general questions related to coding, we picked 7 questions from the ones we removed, representing different types of coding-related issues, and added them back to form 110 questions as inputs for fine-tuning.

Following the openAI guides [9], all 110 questions are put into a single file, to be fed into the LLM fine-tuning interface. Each record in our dataset is structured according to a "role-completion" format, which is divided into three distinct components: system role, user role, and assistant role. The system role is specifically designated as "Act as an API marketplace expert to address user concerns effectively," positioning it to function in an assistant capacity. Within this framework, the inquiries posed by developers are categorized under the user role, illustrating the perspective from which questions are asked. On the other side, the role of the "Assistant" is assigned to encapsulate the expected LLM outcomes, encompassing both actionable recommendations in response to the developers' queries and pertinent information aimed at API marketplaces and service providers. This role-completion method is designed to simulate a realistic interaction scenario, where the LLM adopts the role of an informed assistant, providing targeted advice and insights based on the user's (developer's) inquiries and the predefined system role's expertise.

2) Training the LLM tool: As we mentioned earlier, we expect the fine-tuned model to 1) answer developers' questions automatically without requiring human developers to get involved; and 2) extract useful information from these questions. Hence, for each of the questions in the training set, we created answers (assistant role) following the procedure given below:

- Manually labeling each question: in particular, we labeled the following information for each question: 1) what category/subcategory it belongs to; 2) which service is mentioned; and 3) if an invocation key is exposed.
- Adding solutions to the answer: we created general suggested solutions for each category and subcategory of questions, and formed the answers using these suggestions. For example, for the example question given in the introduction section and other questions with the same root cause, we give the same suggestion of "the problem is caused by RapidAPI requires to subscribe to a service before invoking it, even if the service is free. To solve this problem, login to RapidAPI and subscribe to the service you are using."
- Adding information to be extracted to the answer, including its service and whether an invocation key is exposed, using the Chain-of-Thought (COT) prompts.

CoT prompts generate intermediate reasoning steps before arriving at a final answer, mimicking human problem-solving processes. It enhances the model's ability to tackle complex questions by making its reasoning transparent and interpretable, thereby improving the accuracy and reliability of its outputs. When generating inputs, we follow the CoT mechanism by giving the logic behind how we reason the information from questions, which helps LLM accurately extract information from differently formatted developers' questions. To demonstrate the difference between prompting with and without CoT, we give two examples below.

1) Without CoT:

Listing 1. Without CoT

```
User: Authentication never works... fetch(https://
    covid-19-data.p.rapidapi.com/totals, {method:
    GET, headers: { x-rapidapi-key:
    keyHiddenInManuscript, x-rapidapi-host: covid
    -19-data.p.rapidapi.com}}).then(response => {
    console.log(response);}).catch(err => {console.
    error(err);}); will return 403 {message: You are
     not subscribed to this API

Assistant: This is a Usability issue because it
    mentions in the user question - You are not
    subscribed to this API and 403. \n Please
    subscribe to the API. The API is X-RapidAPI-Host
    : covid-19-data.p.rapidapi.com. Please hide the
    key to avoid security issues. The code-practice
    is not good as X-RapidAPI-Key
    keyHiddenInManuscript, is exposed causing
    security issue.
```

2) With CoT:

Listing 2. With CoT

```
User: Authentication never works... fetch(https://
    covid-19-data.p.rapidapi.com/totals, {method:
    GET, headers: { x-rapidapi-key:
    keyHiddenInManuscript, x-rapidapi-host: covid
    -19-data.p.rapidapi.com}}).then(response => {
    console.log(response);}).catch(err => {console.
    error(err);}); will return 403 {message: You are
     not subscribed to this API.}

Assistant: The user question has a usability issue,
    noting that the user has an error You are not
    subscribed to this API. The problem is caused by
     RapidAPI requiring to subscribe to a service
    before invoking it, even if the service is free.
     To solve this problem, log in to RapidAPI and
    subscribe to the service you are using. In the
    analysis, it is recognized that the X-RapidAPI-
    Host is covid-19-data.p.rapidapi.com, and it is
    noted negatively that the X-RapidAPI-Key is
    exposed as keyHiddenInManuscript, indicating
    wrong security practices, User must hide the API
     key. The exposed key is a 50-character mix of
    numbers and lowercase letters, which is a
    security concern. However, the main concern is
    regarding API subscriptions. To address this, it
     is suggested user should generate a new, valid
    API key. Additionally, it's advisable to double-
    check the header parameters and API subscription
     to ensure they are correctly configured. These
    actions could potentially identify the API
    subscription problem, addressing the usability
    category issue effectively.
```

In particular, we chose gpt-3.5-turbo-1106 among all other models due to its large token limit of 16K (1 token is 4

[9]https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset

words) for each training input stream. Being able to preserve more question details enhances contextual comprehension and optimizes model performance. The runtime for fine-tuning 110 records is almost 15 minutes with an epoch size of 3. We run the fine-tuning two rounds, one without CoT and one with CoT. The cost for two rounds of training is minimal, i.e., around $10.

3) Test Dataset Collection for Evaluation: To test the performance of our fine-tuned LLM, we further retrieved additional SO questions from a historical dataset hosted on Google BigQuery for test dataset generation. Different from our training dataset which was partially collected from the search interface of SO, we can query all SO questions using BigQuery without being limited to the first 500 results as in the search interface. We removed the overlapping data and still got around 1,000 questions left, from which we selected the most recent 100 records for testing. We applied these 100 questions to our fine-tuned model and recorded the responses. As the ground truth, we further manually labeled their categories/subcategories, as well as other information like the impacted service names and key leakages.

IV. Results

In this section, we give the detailed answers to the RQs, followed by observations and actionable suggestions to web service stakeholders, including service providers, App developers, and marketplaces. Below we first briefly summarize the answers to the RQs.

- RQ1: Marketplace differs from static registry in terms of service subscription, service registration, service invocation, billing, and service QoS monitoring.
- RQ2: The taxonomy is quite different from that of static registry, with more issues related to usability, security, documentation, and performance.
- RQ3: A LLM-based model fine-tuned with the labeled data can achieve 85% accuracy for answering developers' questions, 88% accuracy for identifying service names, and 77% accuracy for detecting key leakages.

A. Differences Between Two Generations of Service Registry

Fig. 2 and 3 describe the workflows of the traditional service registry and service marketplace, respectively. In both figures, we use *green* to denote the workflow of service selection and *orange* to denote service invocation. We observe that the workflows are very similar. For service selection, service providers first register their services, and App developers search for services. App developers then subscribe to the service they select and distribute the invocation key with software to end users. For service invocation, end users use the invocation key to send service requests.

1) Workflow Comparison: We compare the workflow of two generations in detail.

1) Service Subscription: Service subscription is the main difference in the service selection workflow between both generations. While both generations display service information on the registry, the traditional registry redirects developers to the service provider's website, for further purchasing and subscription. In contrast, marketplaces provide a one-stop experience for service selection, the service developers are supposed to fill in all descriptive information about their services on the marketplaces, and the App developers will subscribe to the service on the marketplace and also pay to the marketplace.
2) Service Registration: In a service marketplace, the process of subscribing to APIs differs from that of a service registry. Developers are required to subscribe to the necessary API directly within the marketplace itself. Therefore, the marketplace must provide comprehensive information. Without such information, users may not be directed to the service provider's website
3) Service Invocation: The service invocation requests for traditional service registries are sent to the service provider's servers, but for service marketplace requests are sent to servers owned by marketplaces. The marketplace then delegates the request from the end users to the service providers.
4) Billing: In a traditional service registry, service providers handle billing before a request is invoked. However, the billing process differs in marketplaces like RapidAPI. Here, users can subscribe to different APIs offered on the platform and are billed on the marketplace itself according to the pricing plans established by the API providers. Charges are determined by factors such as the volume of API requests and the specific endpoints accessed.
5) Service QoS monitoring: QoS monitoring is introduced in the service marketplace and involves tracking various parameters such as popularity, latency, and service level. When the end user invokes a service, they send a request to the service marketplace along with an API key. The service marketplace then verifies the API key and forwards the request to the service provider. Following this, it proceeds to update the Quality of Service (QoS).

In summary, the static service registry helps developers find services but does not play a role in the subsequent service invocation, while the service marketplace is involved throughout the entire software lifecycle of SOA applications. The distinction in workflow between traditional service registries and service marketplaces, particularly in the service marketplace's role as an intermediary between App developers and service providers, has raised concerns among developers. With millions of developers now relying on service marketplaces, it's imperative to address and resolve these commonly faced issues so that platform providers can cultivate trust with users, promote favorable user experiences, and cultivate enduring relationships that will strengthen the platform's success and expansion.

B. Developers' Questions Taxonomy

The outcome of our effort is a taxonomy of developers' concerns (Fig.4), as well as a few observations that could ben-

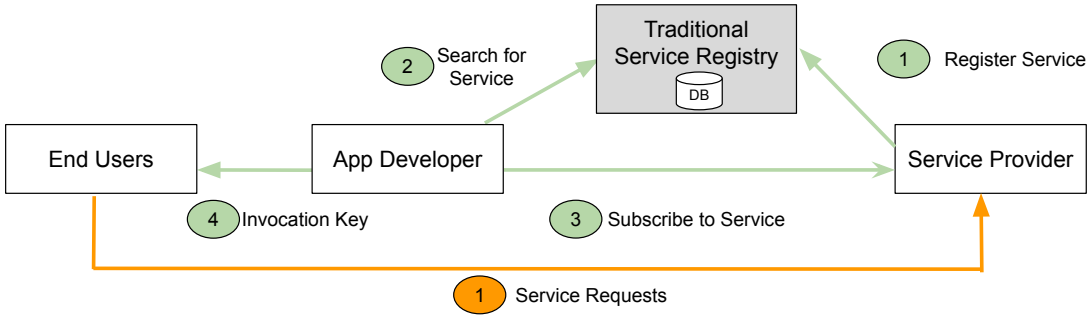

Fig. 2. Workflow of Traditional Service Discovery

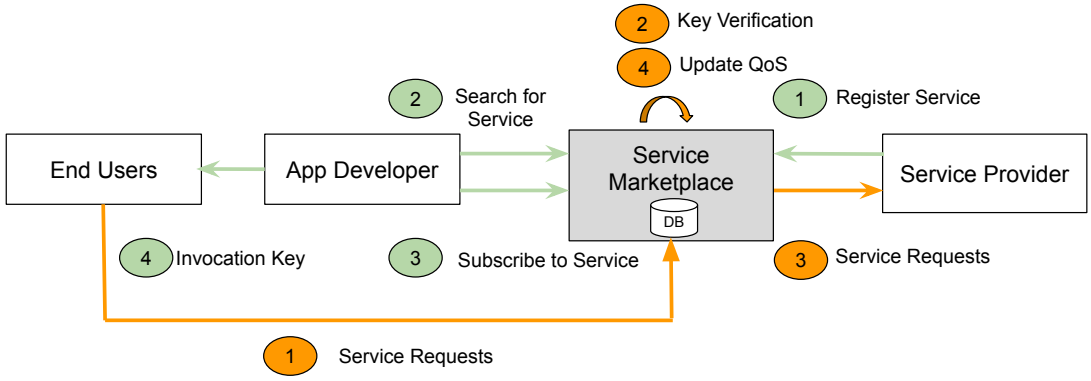

Fig. 3. Workflow of Service Marketplaces

efit stakeholders including the marketplace platform, service developers, and App developers.

Table II describes the issue count from platforms like GitHub, SO, and G2.com. We observed Documentation or general help/support, as well as performance related inquiries, are predominantly found on GitHub. On the other hand, usability concerns and key leakages in the queries posted by users tend to be noticed on SO. Next we introduce these categories in detail.

Platforms	General Queries	Usability	Performance	Security
GitHub	29	12	15	3
SO	-	20	8	12
G2.com	-	-	4	-

TABLE II
ISSUE DISTRIBUTION IN GITHUB, SO, AND G2.COM

1) GENERAL QUERIES (Help/Support):
This category encompasses issues regarding assistance related to services on the marketplace.

- *Integration/Migration/Upgradation:* This subcategory involves incorporating the API into an existing system or application, transferring the API from one environment to another, and updating the API to a newer version.
- *Finding Similar APIs:* In this subcategory, developers want to locate other APIs within the platform that offer comparable functionality, services, or features to a specific API. This allows users to explore alternative options or identify additional resources that may better suit their needs.
- *Guides/ Tutorials/ Documents:* This subcategory involves users seeking help with some uncertainties, enhancing understanding, and acquiring practical knowledge on utilizing APIs effectively.

An example of such issue is *"(where can I find) Documentation for API TMDB*?"

As indicated in the table I, while being able to offer customized documentation in the service marketplace, many service providers still fail to provide adequate documentation concerning endpoints, integration methods, and other details. Consequently, App developers express concerns about the need for API documentation.

Migration Impact 1

While App developers are no longer redirected to service providers' websites and rely only on documentation listed on marketplaces, many service providers fail to provide adequate documentation when listing their services. App developers often struggle to search for API documentation.

Actionable Suggestions: According to our analysis of RapidAPI data, only 56% of RapidAPI services include listed resource links containing information about the API product,

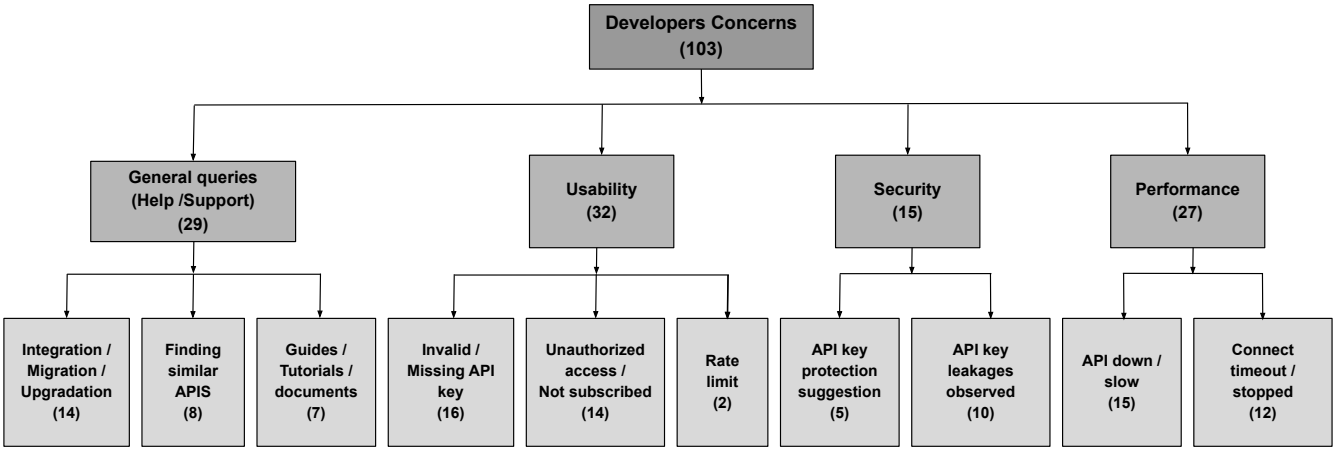

Fig. 4. API marketplaces user concerns

while 22% of services provide documentation details outlining service endpoint information, integration steps, and more. The platform should monitor APIs lacking sufficient documentation support by offering resource links, documentation, and tutorials for the service. It should also notify API providers to improve their documentation by incorporating detailed information on service endpoints, API integration methods, and more.

2) USABILITY:

Usability issues are associated with problems or difficulties that users encounter when interacting with a product, system, or interface.

- *Invalid / Missing API key:* An Invalid API key error occurs when the API key provided by the user is either incorrect or expired and a "Missing API key" error occurs when the user fails to include the required API key in the request.
- *Unauthorized access / Not Subscribed:* The usability issue "Unauthorized access" or "Not Subscribed" typically occurs when users attempt to access certain features or resources within an API without proper authorization or subscription status.
- *Rate limit:* Rate limiting is a mechanism APIs use to control the number of requests made by a client within a specific period. Within Rapid API, every API service is bundled with pricing packages, each of which provides a specific allocation of requests either on a daily or monthly basis.

Fig, 5 shows two example questions falling into this category. These examples show App developers are often confused by the resulting return codes, wondering whether the problem lies with the HTTP library usage, the platform itself, or the service provider. For instance, a 403 error code may be caused by errors in crafting the request URL sent to the marketplace, key authentication failures on the marketplace, or the service

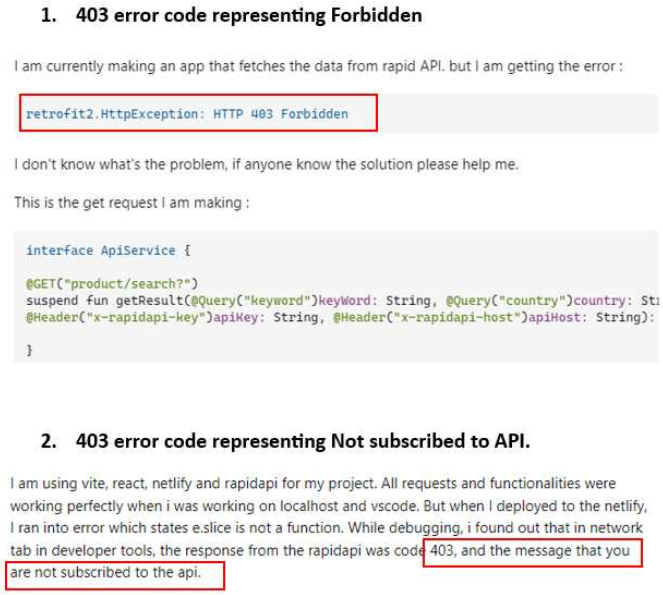

Fig. 5. HTTP Error code ambiguity

provider's server going down.

Migration Impact 2

The marketplace's involvement in service invocation procedure adds complexity, making it challenging for developers to identify the causes of invocation failures and debug them.

Fig. 6 shows an example question related to the new subscription model. As explained in Sec. IV-A1, static registry redirects App developers to service providers' platforms, where developers subscribe to the service and get invocation keys. In the marketplace, developers are granted an invocation

3. **Service invocation mechanism in service marketplace.**

Fig. 6. Service Invocation mechanism

key by default. However, to invoke services, they need to further subscript to these services on the marketplace. Many of them are confused by such a transition, as they think having a key is enough for invoking services. However, when we inspected the detail page of web services managed by RapidAPI, which is usually the landing page when App developers search for keywords like "free face recognition web services," we discovered no hints suggesting that developers must subscribe to the services before invoking them. Furthermore, the executable code, tailored with the developer's unique key, is readily supplied.

Migration Impact 3

The new subscription model confuses developers.

Actionable Suggestions: To help App developers debug the problem, we suggest the marketplaces define standard and fine-grained error codes for all the services they manage. Instead of relying on the default HTTP response codes, Baidu API already allows service providers to specify their fine-grained error codes in documentation. We believe defining uniform error codes can help automate the debugging procedure. Besides, many of the questions related to the new subscription model could be easily answered by providing proper guidance information to developers.

3) SECURITY:

Security issues in API marketplaces involve vulnerabilities such as inadequate authentication, data exposure, and insufficient encryption, potentially leading to unauthorized access and data breaches.

- *API key protection suggestion:* Some questions seek help in safeguarding API keys. Many answers suggest securely storing them in environment variables or utilizing a dedicated secrets management service. Some answers also recommend avoiding hard-coded keys directly into source code or public repositories to reduce the risk of exposure.
- *API key leakages observed:* Within this specific issue subcategory, developers have accidentally exposed API keys when posting their questions.

Figure. 7 gives a code snippet that has the key leakage issue. When invoking a service, App developers need to specify its host and a developer-specific API invocation key. Although static registries and the marketplace both face such issues, their impact on the marketplace is more severe. This is because one developer may subscribe to multiple services, with some being free services and others requiring payments, using the same key. We call such a mechanism "one-key-for-all-service." Even if a developer's question is about invoking a free service, a malicious third party could use the key to access all paid services the developer has subscribed to, with some containing sensitive customer data. Besides, as RapidAPI now plays an important role in service invocations, if attackers use the captured keys to perform a DDoS attack and cause RapidAPI delegation servers to fail, all services managed by RapidAPI will be impacted.

```
headers: {
  'X-RapidAPI-Host': 'coinranking1.p.rapidapi.com',
  'X-RapidAPI-Key': '[redacted]',
},
```

Fig. 7. Key Leakage code snippet example

Migration Impact 4

Developers unintentionally expose their invocation keys in question, posing significant consequences due to the 'one-key-for-all-service' mechanism in marketplaces.

Actionable Suggestions: We believe it's the platform's responsibility to prevent exposure of the invocation key. The platform can prevent key leakages by 1) Marketplaces should find safer alternatives to the 'one-key-for-all-service' model; 2) being able to identify instances where keys are exposed and prompt users to securely hide them, which is one of the reasons why we developed the LLM-based tool; and 3) educating application developers.

4) PERFORMANCE:

Performance is the efficiency and responsiveness of a system or application. Within the Performance issue category, developers express concerns regarding slow response times or connection timeouts.

- *API down /slow:* The developers may face this issue when the API is either experiencing downtime or operating at a reduced speed.
- *Connect timeout/ stopped:* This typically happens when there's a delay in the connection process, or when the connection attempt is terminated due to certain conditions such as network issues, server unavailability, or misconfiguration. In some instances, the service may have completely ceased to function on the side of the API provider.

A specific example is*"(why) Ecoindex.fr Backend API was down*?"

	Dimension	Correctly classified(CoT)	Correctly classified(w/o CoT)	Total Issues
Classification	General Queries Issues	6	4	6
	Usability Issues	28	23	28
	Performance Issues	9	6	9
	Coding Issues	42	42	57
	Total Test data	85	75	100
Information Extraction	Service Detection from Total Test data	84	37	95
	Key Leakage Identified from Total Test Data	20	7	26

TABLE III
FINE-TUNED MODEL ACCURACY FOR TEST DATASET

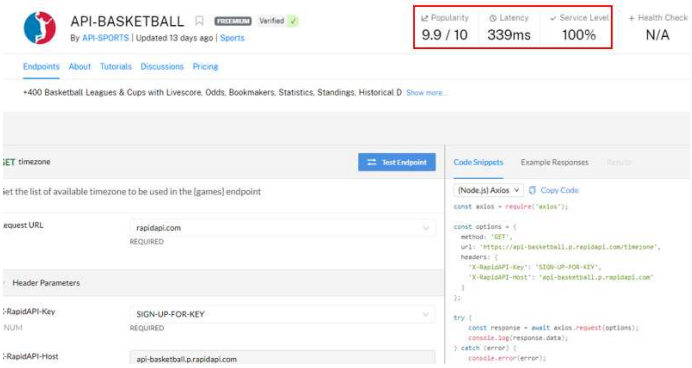

Fig. 8. RapidAPI QoS

The services provided by the API providers are poorly coded and inadequately tested by the platform before being added to the platform. Despite marketplaces like as shown in Fig. 8 RapidAPI being designed to aid developers in monitoring Quality of Service (QoS), they mainly focus on monitoring the long-term QoS and respond to short-term QoS issues not swiftly enough. Consequently, we observe some developers inquiring whether the service is temporarily down or why latency is high, due to the platform's slow updates of these parameters.

Migration Impact 5

The platform provides QoS updates from long-term monitoring but developers expect immediate QoS alerts.

Actionable Suggestions: API marketplaces should identify these services and create Service Level Agreements (SLAs) with service providers. The API marketplace must communicate to service providers the necessity of maintaining well-coded and thoroughly tested APIs. Our LLM tool aids application developers in understanding the cause of service downtime.

C. Evaluation of Fine-tuned Automated Tool

We evaluated our fine-tuned model to ascertain its proficiency in accurately identifying issue categories, impacted APIs, and key leakages. By comparing the model's classification results with the manual labels, Table III lists the number of records correctly identified in each category. Interestingly, the model successfully categorized issues related to Usability,

Performance Issues	Key Leakages
calorieninjas.p.rapidapi.com	euromillions.p.rapidapi.com
api-football-v1.p.rapidapi.com	yummly2.p.rapidapi.com
omgvamp-hearthstone-v1.p.rapidapi.com	lambda-face-recognition.p.rapidapi.com
elenasport-io1.p.rapidapi.com	omgvamp-hearthstone-v1.p.rapidapi.com
bayut.p.rapidapi.com	api-football-v1.p.rapidapi.com
coinranking1.p.rapidapi.com	community-open-weather-map.p.rapidapi.com
matchilling-chuck-norris-jokes-v1.p.rapidapi.com	booking-com.p.rapidapi.com
textanalysis-keyword-extraction-v1.p.rapidapi.com	webknox-trivia-knowledge-facts-v1.p.rapidapi.com
	corona-virus-world-and-india-data.p.rapidapi.com
	google-translate1.p.rapidapi.com
	devru-times-of-india.p.rapidapi.com
	alpha-vantage.p.rapidapi.com
	indeed-indeed.p.rapidapi.com
	v1-sneakers.p.rapidapi.com
	coinranking1.p.rapidapi.com
	covid-19-data.p.rapidapi.com
	apidojo-yahoo-finance-v1.p.rapidapi.com
	faceplusplus-faceplusplus.p.rapidapi.com
	cryptocurrency.p.rapidapi.com
	dev132-cricket-live-scores-v1.p.rapidapi.com

TABLE IV
IMPACTED APIS IDENTIFIED FROM TEST DATA

Performance, and General Queries. However, it erroneously classified 15 coding issues into different categories and failed to identify 11 API names and 6 instances of key leakages. Overall we achieved an accuracy of 85% for question classification, 88% for identifying service names, and 77% for identifying key leakages. We noticed that approximately 30-40% of API services were affected by key leakages.

Table IV lists all the impacted APIs that we've manually confirmed from the questions in the test dataset, with the help of our tool. It demonstrates that the tool can be used to detect performance outages and key leakages, and give warnings to service providers and marketplaces.

We further evaluate the impact of CoT in fine-tuning. The accuracy achieved by the training model without CoT was 75% for classification and around 40% for identifying service names and key leakages. It shows that CoT improves the accuracy of fine-tuning, especially for identifying service names and key leakages which requires understanding the logic behind the reasoning.

D. Implications

Our empirical analysis identified key differences between traditional service registries and Service Marketplaces. Moreover, the study also identified a taxonomy of common mi-

gration challenges that developers often face. Further, we developed a proof of concept of how the LLM model can be useful to assist the developers. The findings and technique of the study can lead to the following implications.

For SOA Stakeholders Our analysis identified key differences between traditional service registries and Service Marketplaces. Such findings could be beneficial for developers to make decisions on migrating from traditional service registries and Service Marketplaces. Moreover, the taxonomy can guide developers on common issues of Service Marketplaces migration, their root causes, and the solution.

For Researchers The taxonomy of Service Marketplaces migration could help researchers to develop advanced research techniques such as recommendation systems [17], [18] and automated program repair [19], [20] techniques to assist developers in fixing migration issues more efficiently and in a timely manner.

For Tool Builders The evaluation showed that our proposed approach of utilizing an LLM-based issue categorization technique can effectively categorize migration issues. The tool builders can take further initiative to integrate that technique with IDEs and development tools so that developers can instantly get feedback on the issues they face during migration.

V. Conclusion

In summary, this paper presents three key contributions. Firstly, we extensively detailed how the transition in workflow from service registry to service marketplace is the primary reason for developers' confusion when utilizing service marketplaces. Secondly, we established a taxonomy of user concerns, delineating their origin from the shift from traditional service registries to service marketplaces. Lastly, we introduced an automated tool capable of identifying these concerns and offering 3-dimensional assistance to App developers, service providers, and the platform. Through thorough evaluation, we validated the tool's accuracy and summarized the implications of this research.

Acknowledgement

This research is supported by NSF through the grants #2104337 and # 2152819.

References

[1] A. Tewari, P. Nagdev, and A. Sahitya, "Sky computing: the future of cloud computing," *IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 6, no. 4, pp. 3861–3864, 2015.

[2] N. T. Yazdi and C. H. Yong, "A potential way for efficient information sharing based on mobile text messaging," in *2012 International Conference on Green and Ubiquitous Technology*, 2012, pp. 172–175.

[3] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 223–235, 2010.

[4] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "{MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 945–960.

[5] "Global machine learning as a service (mlaas) industry research report, competitive landscape, market size, regional status and prospect," shorturl.at/mKMY3, accessed: 2023-12-30.

[6] E. Al-Masri and Q. H. Mahmoud, "Investigating web services on the world wide web," in *WWW'08*, 2008, pp. 795–804.

[7] N. Zhang, Y. Zou, X. Xia, Q. Huang, D. Lo, and S. Li, "Web apis: Features, issues, and expectations–a large-scale empirical study of web apis from two publicly accessible registries using stack overflow and a user survey," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 498–528, 2022.

[8] K. Mahmood, G. Rasool, F. Sabir, and A. Athar, "An empirical study of web services topics in web developer discussions on stack overflow," *IEEE Access*, vol. 11, pp. 9627–9655, 2023.

[9] R. Akolkar, T. Chefalas, J. Laredo, C.-S. Perng, A. Sailer, F. Schaffa, I. Silva-Lepe, and T. Tao, "The future of service marketplaces in the cloud," in *2012 IEEE Eighth World Congress on Services*, 2012, pp. 262–269.

[10] P. K. Venkatesh, S. Wang, F. Zhang, Y. Zou, and A. E. Hassan, "What do client developers concern when using web apis? an empirical study on developer forums and stack overflow," in *ICWS'16*. IEEE, 2016, pp. 131–138.

[11] G. Ajam, C. Rodríguez, and B. Benatallah, "Api topics issues in stack overflow q&as posts: An empirical study," in *2020 XLVI Latin American Computing Conference (CLEI)*, 2020, pp. 147–155.

[12] H. Zhang, S. Wang, T.-H. Chen, Y. Zou, and A. E. Hassan, "An empirical study of obsolete answers on stack overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 4, pp. 850–862, 2021.

[13] B. Xu, T.-D. Nguyen, T. Le-Cong, T. Hoang, J. Liu, K. Kim, C. Gong, C. Niu, C. Wang, B. Le *et al.*, "Are we ready to embrace generative ai for software q&a?" in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 1713–1717.

[14] Y. Feng, S. Vanam, M. Cherukupally, W. Zheng, M. Qiu, and H. Chen, "Investigating code generation performance of chat-gpt with crowdsourcing social data," in *Proceedings of the 47th IEEE Computer Software and Applications Conference*, 2023, pp. 1–10.

[15] H. Li, F. Khomh, M. Openja *et al.*, "Understanding quantum software engineering challenges an empirical study on stack exchange forums and github issues," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 343–354.

[16] F. Corno, L. De Russis, and J. P. Sáenz, "On the challenges novice programmers experience in developing iot systems: A survey," *Journal of Systems and Software*, vol. 157, p. 110389, 2019.

[17] D. Kim, Y. Tao, S. Kim, and A. Zeller, "Where should we fix this bug? a two-phase recommendation model," *IEEE transactions on software Engineering*, vol. 39, no. 11, pp. 1597–1610, 2013.

[18] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 72–81.

[19] C. L. Goues, M. Pradel, and A. Roychoudhury, "Automated program repair," *Communications of the ACM*, vol. 62, no. 12, pp. 56–65, 2019.

[20] Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang, "The strength of random search on automated program repair," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 254–265.