

GSoC 2025 Proposal



AboutCode

VulnerableCode: On-demand live evaluation of packages and Integration with VulnTotal and its browser extension [Large (350 hours)]

March 2025

Personal Information

Name: Michael Ehab Mikhail

GitHub Username: [michaelehab](#)

Email: michaelehab16@gmail.com

Phone: +201210107756

Major: Computer Engineering

College: Cairo University

Country: Egypt

Time Zone: Cairo (GMT+2)

LinkedIn: <https://www.linkedin.com/in/michaelehab16>

Gitter username: @michaelehab:gitter.im

Internships



- **Software Engineering Intern** at [Dell Technologies](#) (Oct 2023 - Present)
 - Actively contributing to Project Alvarium, a trust fabric framework for secure data transmission.
 - Spearheading a reinforcement learning project with a focus on in-depth research, paper review, and implementation.



- **Software Engineering Intern** at [Microsoft](#) (Jul 2023 - Oct 2023)
 - Designed and implemented a solution for generating content using Large Language Models (LLMs).
 - Integrated the LLM prompts with internal tools to automate the generation process.
 - Ensured data quality by implementing and automating the junk detection process.



- **Software Engineering Intern** at [Brightskies inc](#) (Feb 2023 - Jul 2023)
 - Worked on designing, developing, and running a project that solved our client's challenge using Java, Spring Boot, Hibernate, JUnit, Maven, Gitlab, and Jira.
 - Proactively participated in weekly team meetings and conducted reports on the project's progress.



- **Back End Software Engineering Intern** at [Banque Misr](#) (Aug 2022 - Sep 2022)
 - Achieved 1st place out of 7 cross-functional intern teams by creating a REST API that powered Android, iOS, and Web apps using Java, Spring Boot, Hibernate, and PostgreSQL database.

Extracurricular Activities

- Software Engineer at [Cairo University Eco Racing Team](#) (Aug 2021 - Jul 2022)
 - Increased workshop utilization compared to last year's numbers by creating a REST API that served the racing team mobile app using Python, Django, and PostgreSQL.
- Web Development Instructor at GDSC Cairo University (Aug 2021 - Jul 2022)
 - Coached more than 20 computer science students to start coding and learn the basics of Web development by organizing weekly sessions to teach them HTML, CSS, and JavaScript.

Previous Open Source Contribution

Contributions to AboutCode organization

I've been a contributor to AboutCode VulnerableCode/Vulntotal since March 2023.

I also created **AboutCode** Vulntotal Browser Extension as part of **Google Summer of Code 2024**.

Detailed documentation, including PRs, can be found here:

https://aboutcode.readthedocs.io/en/latest/archive/qsoc/reports/2024/vulntotal_extension_michael.html

Example Merged PRs:

- <https://github.com/aboutcode-org/vulnerablecode/pull/1524>
- <https://github.com/aboutcode-org/vulnerablecode/pull/1157>
- <https://github.com/aboutcode-org/vulntotal-extension/pull/5>
- <https://github.com/aboutcode-org/vulntotal-extension/pull/6>
- <https://github.com/aboutcode-org/vulntotal-extension/pull/9>

Contributions outside AboutCode organization

I am an active contributor to [Project Alvarium, an LF Edge project](#).



Example Merged PRs:

- <https://github.com/project-alvarium/alvarium-sdk-go/pull/68>
- <https://github.com/project-alvarium/scoring-apps-go/pull/16>
- <https://github.com/project-alvarium/alvarium-pipelines/pull/7>
- <https://github.com/project-alvarium/alvarium-sdk-go/pull/66>
- <https://github.com/project-alvarium/scoring-apps-go/pull/14>
- <https://github.com/project-alvarium/alvarium-sdk-java/pull/135>
- <https://github.com/project-alvarium/scoring-apps-go/pull/12>
- <https://github.com/project-alvarium/alvarium-sdk-java/pull/133>
- <https://github.com/project-alvarium/alvarium-sdk-go/pull/54>
- <https://github.com/project-alvarium/scoring-apps-go/pull/10>
- <https://github.com/project-alvarium/alvarium-sdk-go/pull/50>
- <https://github.com/project-alvarium/scoring-apps-go/pull/8>

I participated in GSSoC 2022 and participated in [Awesome Chrome Extensions](#), [CalcHub](#), and [LearnCPP](#).

I built 2 Chrome extensions as a part of my contribution to Awesome Chrome Extensions, they are [Pomodoro Timer with nice UI](#) and [VAT Calculator](#).

Example Merged PRs:

- <https://github.com/ridsuteri/Awesome-Chrome-Extensions/pull/213>
- <https://github.com/0xvashishth/CalcHub/pull/137>
- <https://github.com/ridsuteri/Awesome-Chrome-Extensions/pull/82>
- <https://github.com/0xvashishth/CalcHub/pull/109>
- <https://github.com/Lakhankumawat/LearnCPP/pull/331>

Do you plan to have any other commitments during GSoC that may affect your work? Any vacations/holidays? Will you be available full-time to work on your project?

My final exams will end on June 1st, which is before the coding period starts, so I don't have any serious commitments that may affect my work. I plan to work full-time on my project.

Detailed Description of the Project Idea

VulnerableCode currently processes vulnerability data in **bulk** by running importers that fetch, structure, and store advisories for all available packages at once.

This approach is practical for large-scale vulnerability tracking, but it lacks the flexibility to retrieve and import advisories dynamically for a single package using its PURL (PackageURL).

This project aims to **introduce an API endpoint** that enables **on-demand** importing of vulnerability advisories for a specific **Package URL**.

By supporting dynamic, real-time querying, this feature will enhance efficiency and usability, particularly for use cases that require targeted vulnerability lookups.

This allows us to **improve** VulnerableCode/VulnTotal package which uses VulnerableCode as a data source, so it would support running VulnerableCode **locally** and given that VulnTotal is package-first, it presents a perfect opportunity to **utilize the new API endpoint** in VulnerableCode in an effective way and help VulnTotal users to have their local vulnerability database based on the packages they search with.

We can also build upon this and allow **VulnTotal browser extension** users to have their vulnerability database with **minimum setup**, The extension would utilize the addition to VulnTotal package and be able to call the new API endpoint in VulnerableCode.

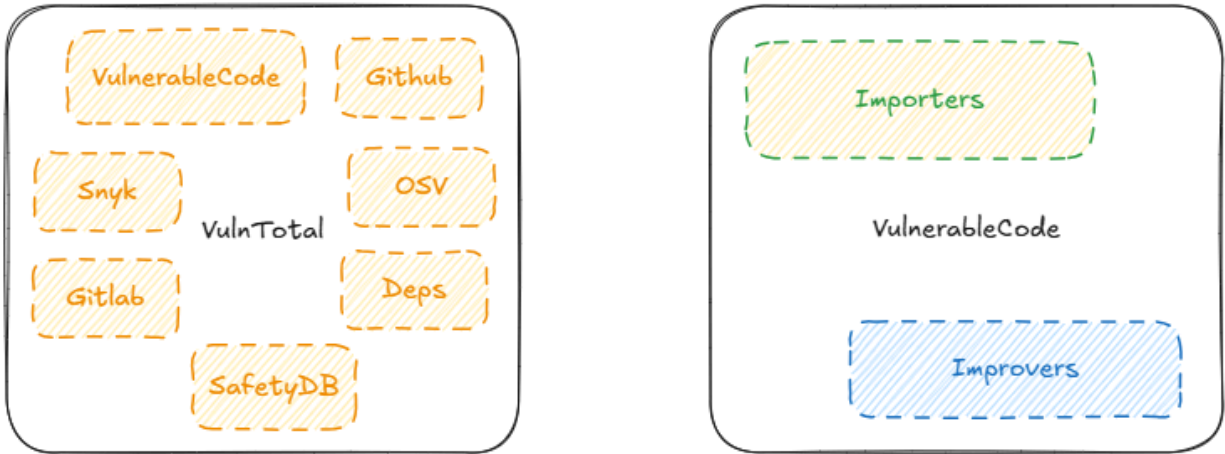
This all results in a **better integration** between VulnerableCode and VulnTotal, and **easier access** to vulnerability data for packages.

So, to summarize, this project consists of **3 main parts** that build on top of each others, first one is **adding an API endpoint** that imports vulnerability advisories for a single package given its PackageURL, then **use this API endpoint** to integrate VulnTotal python package with VulnerableCode, and lastly, **integrate the updated packages** and tools with the browser extension.

I discussed the project idea and approach during a couple of VulnerableCode community meetings with the maintainers to make sure it is applicable and needed since it bridges a gap

that helps user build local advisories db using VulnerableCode which use the advantage VulnTotal has which is accessing proprietary data sources (Since it is allowed in their licences).

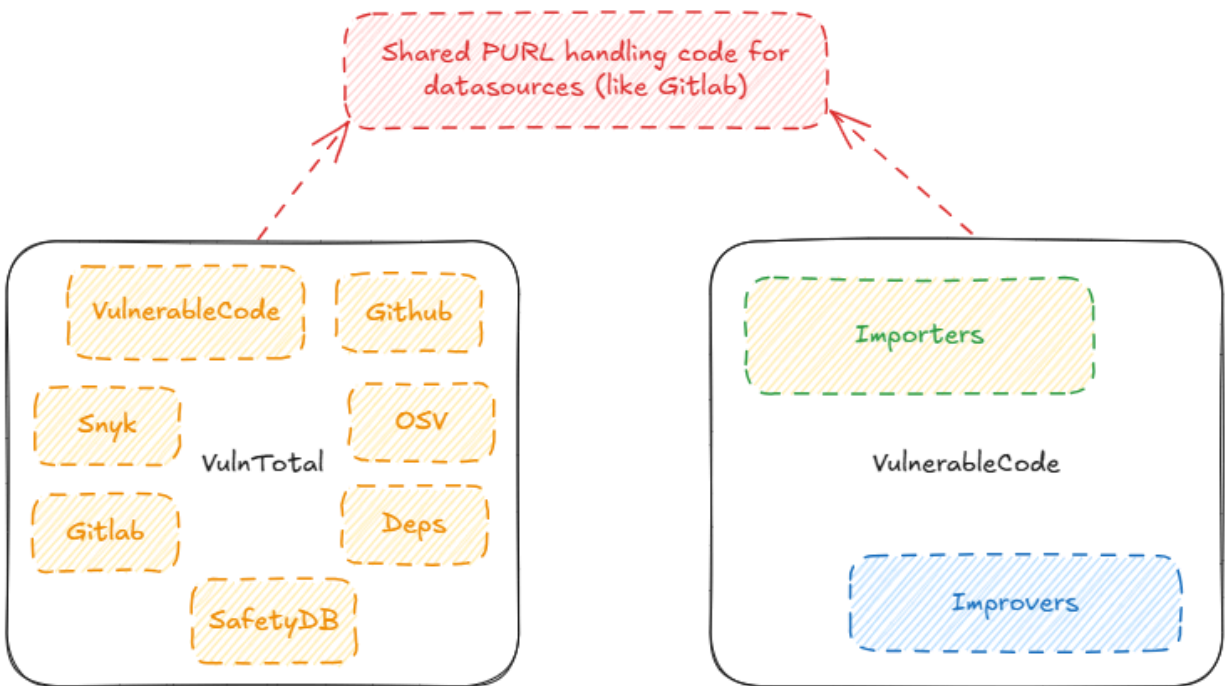
Duplicate Code between VulnTotal and VulnerableCode



We can reduce duplicate code by providing **shared pieces of code** for data source management for PURL that both VulnTotal and VulnerableCode could use now that VulnerableCode will support the package-first approach through the live evaluation feature

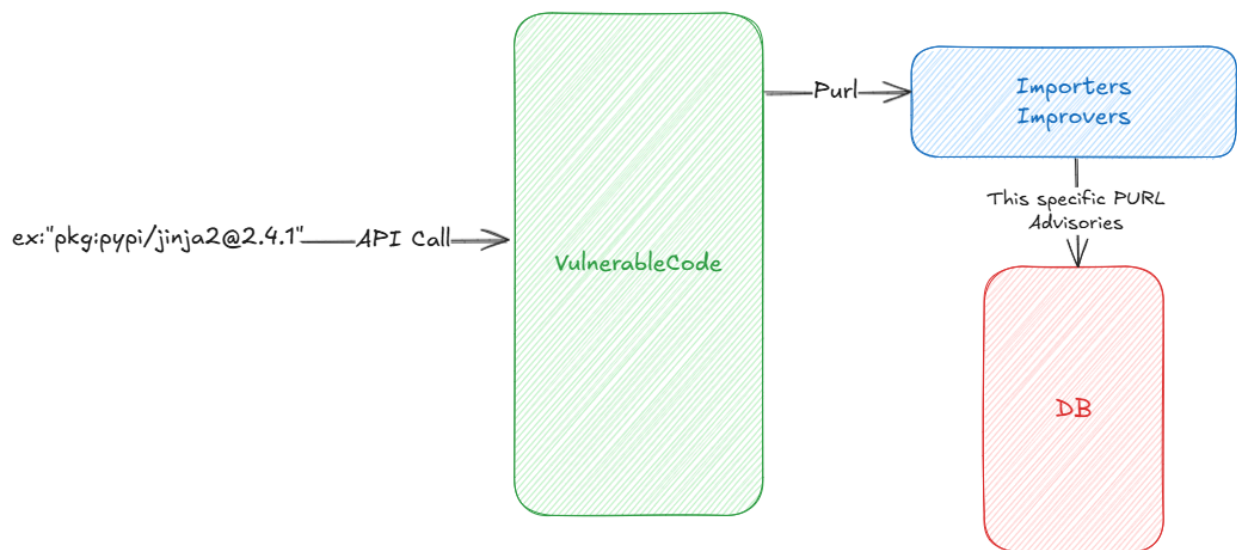
But it is important to note that based on one of the discussions we made it clear that there might still be some differences, for example, in the GitLab data source, we use a VCS approach in VulnerableCode and in a previous PR I modified the GitLab data source in VulnTotal to use an API approach, both methods are valid but in VulnerableCode's current state, it is way more efficient to use a VCS approach for batch processing, and on the other hand, VulnTotal is way more optimized using an API approach since it serves its use case. So, it is essential to maintain each unique aspect while reducing duplicate code and maintaining full functionality.

See VulnerableCode [Gitlab Pipeline Importer](#) and [VulnTotal Gitlab Datasource](#) for reference.

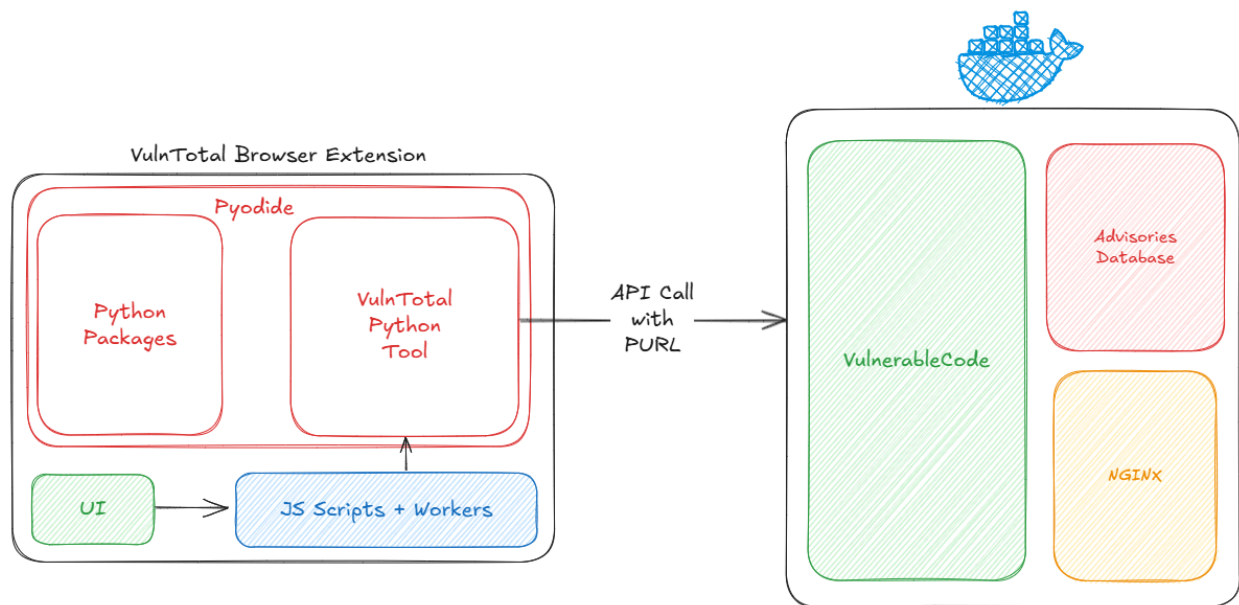


If needed, the new API endpoint could be disabled by default for license reasons and needs manual interaction to be enabled by users on their local versions (Which is an accepted case)

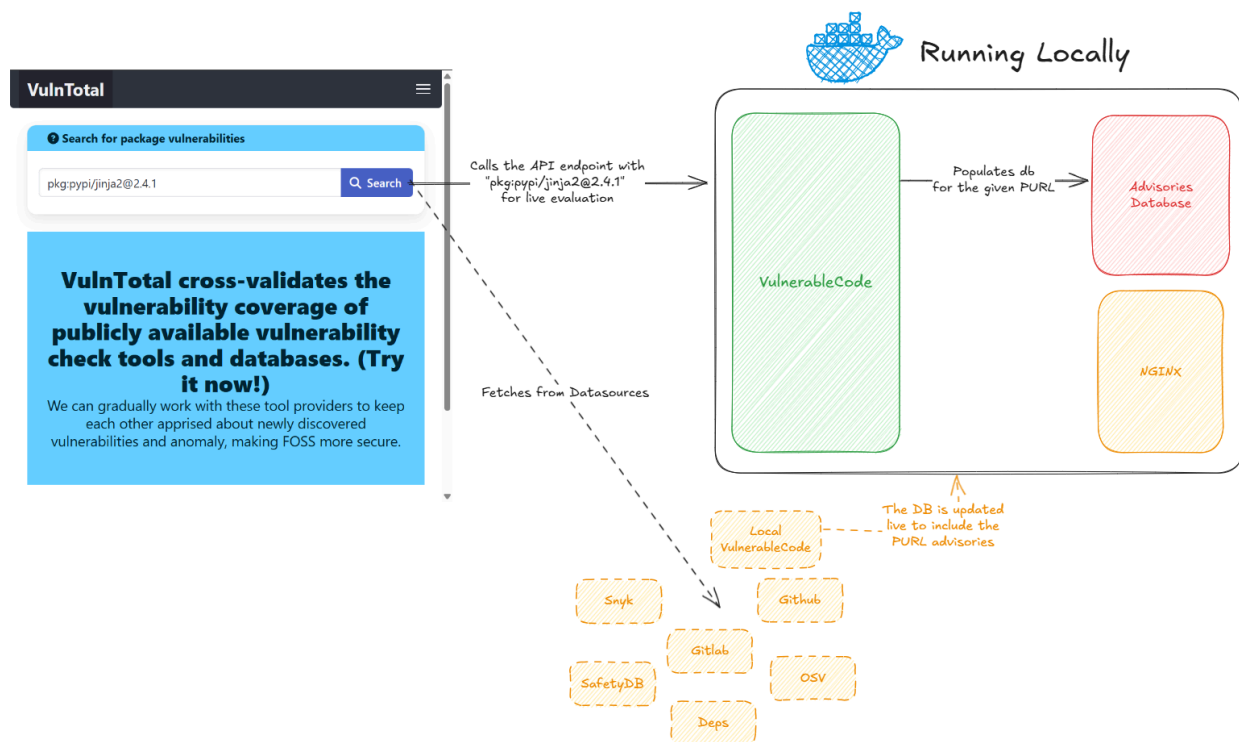
The new API endpoint would allow users to gradually build their advisories db



Example: Consuming the new API endpoint through the VulnTotal browser extension



Detailed Scenario of interaction



So the project would solve the duplicate code issue between VulnerableCode and VulnTotal, introduce a new API call in VulnerableCode for live package evaluation, use the new API endpoint in VulnTotal by adding a local VulnerableCode option, and make things even more easier and more accessible by making this possible within the browser extension I developed during last year's Google Summer of Code for AboutCode.

Demo of the Solution for the first part of the project

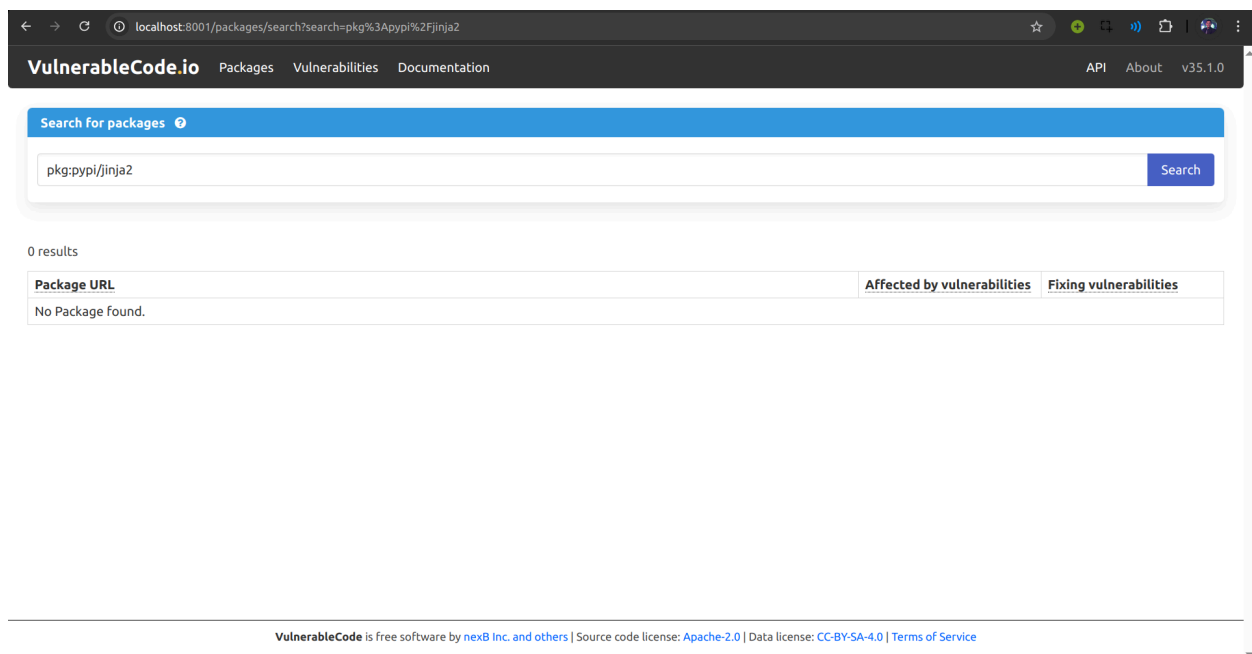
In this demo I focused on showing the feasibility of using VulnerableCode in a package-first way.

Individual importers will need their specific approach depending on their API and specific issues/discussions, so for the sake of this demo, I will focus on Gitlab since it is a pipeline importer and does not have direct package-first API.

By the end of the demo, I aim to show the [test case proposed in AboutCode GSoC 2025 Project Idea](#), which is “A good test case would be to start with a completely empty database. Then we call the new API endpoint for one PURL, and the vulnerability data is fetched, imported/stored on the fly, and the API results are returned live to the caller. After that API call, the database should now have vulnerability data for that one PURL.”

Starting with an empty database

So, I started by making a fresh clone of VulnerableCode, starting with an empty database.



As we can see, when I search using a PackageURL of “pkg:pypi/jinja2” I get 0 results because the database is empty.

After calling the API endpoint, we will do the same search, and our goal is to find packages for this PackageURL and confirm that the database only includes advisories for that exact PackageURL.

Implementing the new API endpoint (Passing the PURL to the pipeline importers because our target is the “Gitlab” importer)

```
class ImporterRunSerializer(serializers.Serializer):
    importer_name = serializers.CharField(
        help_text="Name of the importer to run"
    )
    purl_string = serializers.CharField(
        help_text="PackageURL string to filter the importer run",
    )

class ImporterRunViewSet(viewsets.ViewSet):

    serializer_class = ImporterRunSerializer

    @extend_schema(
        request=ImporterRunSerializer,
        responses={
            202: {"description": "Importer started successfully"},
            400: {"description": "Invalid request"},
            500: {"description": "Internal server error"}
        }
    )
    @action(
        detail=False,
        methods=['post'],
        url_path='run-importer'
    )
    def run_importer(self, request):
        serializer = ImporterRunSerializer(data=request.data)
        if not serializer.is_valid():
            return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

        importer_name = serializer.validated_data.get('importer_name')
        purl_string = serializer.validated_data.get('purl_string')
        purl = PackageURL.from_string(purl_string) if purl_string else None
```

```

try:
    importer = IMPORTERS_REGISTRY[importer_name]
except KeyError:
    return Response(
        {'error': f'Importer {importer_name} not found.'},
        status=status.HTTP_400_BAD_REQUEST
    )

# Handle pipeline importers
if issubclass(importer, VulnerableCodeBaseImporterPipeline):
    pipeline_instance = importer(purl) # Pass the PURL to the
importer
    status_code, error = pipeline_instance.execute()
    if status_code != 0:
        return Response(
            {'error': f'Importer {importer_name} failed: {error}'},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )
    return Response(
        {'message': f'Importer {importer_name} executed
successfully.'},
        status=status.HTTP_202_ACCEPTED
    )

# Handle regular importers
try:
    ImportRunner(importer).run()
    return Response(
        {'message': f'Importer {importer_name} executed
successfully.'},
        status=status.HTTP_202_ACCEPTED
    )
except Exception as e:
    error_msg = f"Error running importer {importer_name}: {str(e)}"

    return Response(
        {'error': error_msg},
        status=status.HTTP_500_INTERNAL_SERVER_ERROR
    )

```

Register the new API endpoint

```
api_v2_router.register(  
    "importer-runs",  
    ImporterRunViewSet,  
    basename="importer-runs"  
)
```

Modify the Gitlab importer to use the PURL if it exists

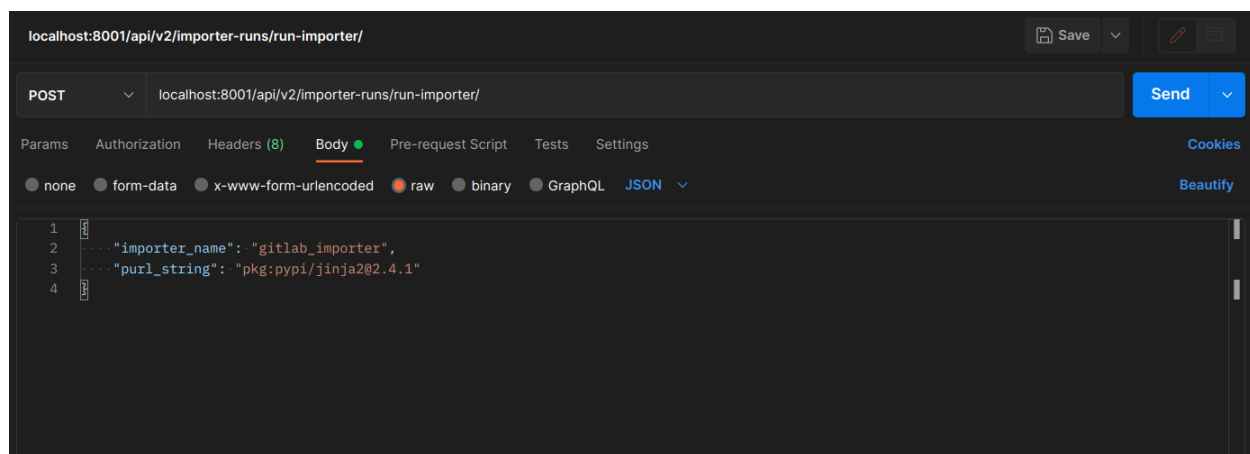
Modify the constructor to take the PURL

```
def __init__(self, purl: Optional[PackageURL] = None):  
    super().__init__()  
    self.purl = purl
```

Modify the collect_advisories method to use the PURL

```
advisory = parse_gitlab_advisory(  
    file=file_path,  
    base_path=base_path,  
    gitlab_scheme_by_purl_type=self.gitlab_scheme_by_purl_type,  
    purl_type_by_gitlab_scheme=self.purl_type_by_gitlab_scheme,  
    logger=self.log,  
)  
  
    if advisory and (self.purl is None or any(pkg.package.name ==  
self.purl.name for pkg in advisory.affected_packages)):  
        yield advisory
```

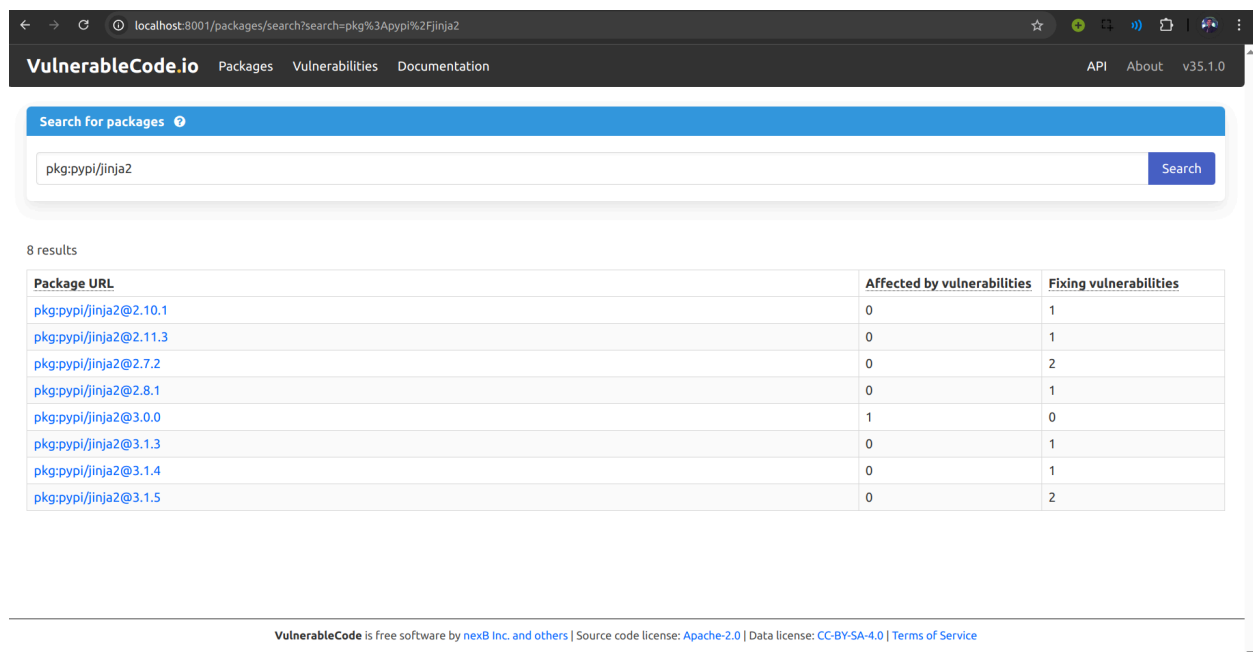
Send a request to start the Gitlab importer with the "pkg:pypi/jinja2" PackageURL



If we check VulnerableCode logs, we will see that it imported only 10 new advisories

```
INFO 2025-03-24 22:11:00.241 Successfully collected 10 advisories
INFO 2025-03-24 22:11:00.241 Step [collect_and_store_advisories] completed in 13 seconds
INFO 2025-03-24 22:11:00.241 Step [import_new_advisories] starting
INFO 2025-03-24 22:11:00.248 Importing 10 new advisories
INFO 2025-03-24 22:11:00.545 Successfully imported 10 new advisories
INFO 2025-03-24 22:11:00.545 Step [import_new_advisories] completed in 0 seconds
INFO 2025-03-24 22:11:00.545 Step [clean_downloads] starting
INFO 2025-03-24 22:11:00.545 Removing cloned repository
INFO 2025-03-24 22:11:01.039 Step [clean_downloads] completed in 0 seconds
INFO 2025-03-24 22:11:01.039 Pipeline completed in 34 seconds
```

Search using the PURL again



The screenshot shows the VulnerableCode.io website with a search bar containing 'pkg:pypi/jinja2'. Below the search bar, it indicates '8 results' and displays a table with three columns: 'Package URL', 'Affected by vulnerabilities', and 'Fixing vulnerabilities'.

Package URL	Affected by vulnerabilities	Fixing vulnerabilities
pkg:pypi/jinja2@2.10.1	0	1
pkg:pypi/jinja2@2.11.3	0	1
pkg:pypi/jinja2@2.7.2	0	2
pkg:pypi/jinja2@2.8.1	0	1
pkg:pypi/jinja2@3.0.0	1	0
pkg:pypi/jinja2@3.1.3	0	1
pkg:pypi/jinja2@3.1.4	0	1
pkg:pypi/jinja2@3.1.5	0	2

At the bottom of the page, a footer states: 'VulnerableCode is free software by [nexB Inc. and others](#) | Source code license: [Apache-2.0](#) | Data license: [CC-BY-SA-4.0](#) | [Terms of Service](#)'.

If we check the VulnerableCode database for the advisories, we will find that we only have the 10 advisories for that specific PackageURL:

```
vulnerablecode=# select count(*) from vulnerabilities_advisory;
count
-----
      10
(1 row)

vulnerablecode=#
```

As we can see, that concludes our demo for the first part of the project, which is the API call for VulnerableCode.

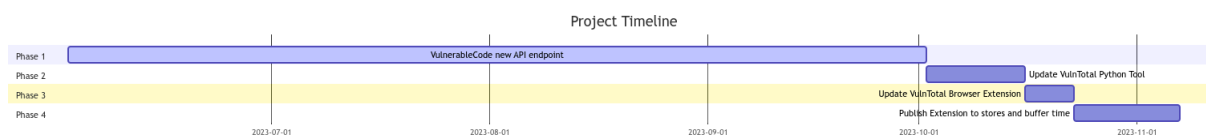
Again, other importers will need to be modified in a more specific way in case they already support package-first fetching like PyPi, the vulnerabilities may be available when querying the main API.

Project Plan

Deliverables

- **VulnerableCode should support live package evaluation:** Importers need to be modified so they can support fetching advisories for a single given PackageURL.
- **VulnTotal should support local VulnerableCode usage:** VulnTotal python package needs to have an option to use local VulnerableCode as a datasource when querying for a PackageURL.
- **VulnTotal browser extension should use the updated VulnTotal extension:** the browser extension needs to get updated to reflect the package update, this allows easier way to interact with the new API added in VulnerableCode.

Project Timeline



Phase 1: VulnerableCode new API endpoint (4 months)

June 2 - October 2

- Reduce duplicate code between VulnerableCode and VulnTotal, especially with the need to modify the VulnerableCode importers to support live package evaluation.
- Modify VulnerableCode importers (both regular and pipeline importers) to allow the usage of a specific PURL.
- Some importers might already support ways to fetch using a package, while others will need to use batch fetching with proper filtering and optimizations.
- **Deliverable:** VulnerableCode supporting on demand live evaluation of packages.

Phase 2: Update VulnTotal Python Tool (2 Weeks)

October 3 - October 17

- VulnTotal should depend on the shared code with VulnerableCode so we have less duplicate code.
- VulnTotal should have a new local VulnerableCode datasource alongside the public VulnerableCode version, and the local version should utilize the new API to help users gradually fill their local advisories database.
- **Deliverable:** VulnTotal should support a local VulnerableCode datasource and use the new API for the on demand live evaluation of packages feature.

Phase 3: Update VulnTotal Browser Extension (2 Weeks)

October 18 - October 25

- The VulnTotal browser extension is the easiest way to interact with the new VulnerableCode API endpoint, so it should be modified to help users use the new feature with minimal setup (setting up the api port of their local VulnerableCode instance).
- **Deliverable:** VulnTotal browser extension should be updated to use the updated VulnTotal python tool and integrate seamlessly with local VulnerableCode.

Phase 4: Publish Extension to stores and buffer time

October 26 - November 10

- Modify the extension metadata as needed to meet stores criteria to publish the extension.
- **Deliverable:** The updated browser extension will be published successfully to available stores (to be determined based on accounts availability).

Existing Solutions

- There are no existing solutions as VulnerableCode currently does not support live package evaluation.