# Triage Portal

Objective

Establish functionality that will support the upload of a single SARIF file via API Endpoint and parse an Omega analyzer SARIF report that contents scan results and the assertion report.

Use Cases

1. Support an API Endpoint within the Triage Portal that will accept a single SARIF file-based conditions.
   1. The accepted SARIF file data will be parsed and stored into its respected database.
      1. Assertion data must be stored in the assertion table.
      2. Scan Results data must be stored in the Findings table.
2. A user must be a trusted party to submit a SARIF file.
3. (Nice to have) Updating the Postgres db must be performed through an API endpoint per table.

Diagram

A picture containing text, diagram, plan, line

Description automatically generated

Requirements

**Use Case #1**

* Within the Triage Portal, create a Restful API (or graphql) in the Django framework to accept a SARIF file.
  + User should be trusted via authentication method (jwt token)
    - Exception with HTTP code if not a trusted user
    - “Admin” to enable user checking.
* Endpoint must validate that it is a SARIF file.
  + Exception with HTTP code if not support file.
* Limit the size of the SARIF file (to be determined in the design)
  + Document size limitation in readme
* API should leverage Triage Portal functionality to parse the scan results.
* Functionality to parse the assertion report will be required (limit code redundant and try to reuse the scan parser)
* Check if assertion data is available before parsing.
* Asynchronously parse out the scan and the assertion data
* Asynchronously store the parsed data into their appropriate database
* If either parse or storage failures, do not prevent the other from completing, but send an appropriate log/error message to stdout.
  + Send a successful HTTP code with an error message notifying “partial success with x failed to store.”
* On the UI, add a notification that the SARIF file has been uploaded.
* Design the API to use the upload functionality (like the UI upload), arguments should be the SARIF file and the package name/version.
* Display the information of the scan result for each individual finding in the Triage Portal Finding page.
  + Parse out the key-value for each finding.
  + Design the page for accessibility and readability on the scan results data.
* Triage Portal Wiki allows transitioning a wiki from closed or deleted to another status.
  + Closed wiki should be shown in the UI.
  + Deleted Wiki should not be shown in the UI.
  + Deleted wiki should be accessible via the URL and their Wiki ID
* In the Wiki page UI, separate the wiki sections into Unresolved Wiki’s (New, Not specified, active) and Resolved Wiki’s (Closed, Resolved)
  + The Resolved Wiki Section should be collapsible. Default should be collapsed.

Bugs fixes and additional functionalities implemented.

* Inside the Tool Defects tap fix error not querying the assigned to me and the active defects (Pull request # 72)
  + https://github.com/ossf/omega-triage-portal/commit/edf41b2b293391a02a1f2bf31c91d895a14c7e6b#diff-0a4e88ef8789d3ffde7c78b98ade254f8f922d58793ec106838af814a746611e
* Fix the edit and add wiki state dropdown for the Wiki tap (Pull request # 77)
  + https://github.com/ossf/omega-triage-portal/pull/77
* Server should not crash/end after 404 Wiki error (Pull request # 77)
  + https://github.com/ossf/omega-triage-portal/pull/77
* Error handling for Closed and Deleted Wiki Status (Pull request # 77)
  + https://github.com/ossf/omega-triage-portal/pull/77
* Upload button not working because OSS Gadget could not download the package. Changed to the latest version of the OSS Gadget (Pull request # 77)
  + https://github.com/ossf/omega-triage-portal/pull/77

Security Requirements

* Making sure that only users that are authorize can access the triage portal (Authorization)
* Users that are part of different groups should have different permissions and access to the triage portal (Authentication)
* To prevent compromising the data of the SARIF file the API will support a checksum field and validate that the checksum matches the calculated checksum of the file.

Acceptance Criteria

* Triage Portal has an accessible endpoint to upload SARIF files.
* Exception and error handling has been designed for credentials and file validation.
* The Triage Portal uploads and parses the scan results into its own database table.
* The triage Portal uploads and parses the assertion results into its own database table.
* Exception and error handling has been implemented for parsing errors.
* The UI displays a module after success or failure of SARIF file upload.
* The Triage Portal should allow any user or groups to upload a SARIF file via API or UI.
* The Triage Portal upload endpoint has limitations on the size of SARIF files that can be uploaded.
* The postgres database has a table for scan results.
* The postgres database has a table for assertion results.
* The Finding page has been designed for accessibility and readability of the scan results and related data.

Future Improvements

* Password management and policy.
* Apply appropriate security measures to protect sensitive data transmitted via the API.
* Validate input received by the API to prevent potential attacks.