Project 2 final deliverable -- Report

As a Boilermaker pursuing academic excellence, we pledge to be honest and true in all that we do. Accountable together – We are Purdue.

Type or sign your names: Ryan Villarreal, Owen Prince, Mohammed Fashola

Write today's date: 8 December 2021

Assignment Goal

In this assignment, you will deliver your Project 2 implementation (*software*) and communicate the final status of your Project 2 (*report*).

This document provides a template for the report. This is a form of documentation that your customer can use to decide (a) whether you've met the contract, and (b) re-negotiate the contract based on deviations therefrom.

Relevant Course Outcomes

A student who successfully completes this assignment will have demonstrated the ability to

- *Outcome ii*: the ability to conduct key elements of the software engineering process, including…deployment
- *Outcome iii*: Develop an understanding of the social aspects of software engineering… including…communication [and] teamwork.

Assignment

Fill out each of the following sections.

Location of project

Provide a URL that we can use to interact with your team's deployed service:

API Home: https://ece-461-project-2-team-4.uc.r.appspot.com/

Front-End Home: https://ece-461-project-2-team-4.uc.r.appspot.com/home/

Provide a link to your team's code repository on GitHub:

https://github.com/Purdue-ECE-461/project-2-project-2-4

Succinct description

In a 5-7 sentence paragraph, describe the system that you have implemented.

We have created a database for NPMJS modules that allows users the ability to upload, download, update, rate, and delete packages using a rating application (referred to as the "trustworthiness module"). Our system allows users to interact with it via a RESTful API or a user-friendly, accessible, ADA-compliant front-end. At ingestion, packages are tested to ensure they are trustworthy enough to be stored in the system using the trustworthiness module. From there, users can browse packages using a paginated, searchable, package viewer. Packages can be requested from the system and rated by providing the package's ID. When needed, the entire system may be reset.

Functional requirements

Baseline metric

You were required to implement a new metric to compute the degree of version pinning in a package. This implementation needed to build on another team's Project 1 implementation.

Describe any changes you made to the existing Project 1 design or implementation,[1] divided into two kinds:

1. Changes to allow you to implement the new metric

Change 1: Added new class that represents the dependency ratio metric.
Justification: They used classes to hold the methods and values of each metric. We used their structure to implement the new dependency ratio metric.

Change 2: Added the JSON parsing functionality.
Justification: We had to extend the current project to add new methods that would allow us more flexibility to parse the list of dependencies in JSON. Their current JSON parsing capabilities were not extendible or reusable.

Change 3: Added JUnit testing suite for the dependency ratio metric.
Justification: Junit allowed us to create unit tests to test the functionality of the metric across a wide variety of inputs. We generated a list of many of the normal and corner cases of possible version strings.

2. Changes to improve the reliability of the component so that your Project 2 implementation would satisfy the customer's requirements.

Change 1: Fixed issue with invalid URL response
Justification: System would crash every time it was run with the invalid URL response.

Change 2: Fixed issue with invalid URL response
Justification: System would crash every time it was run with the invalid URL response.

Change 3: Re-created the Maven project

[1] Hint: It might be helpful to examine your team's PRs or git logs to recall these changes. I assume, of course, that you followed an appropriate engineering process so that you can answer these questions.

Justification: Needed to create a new Maven project from scratch and add code back to get rid of build issues so I could build testing suites.

Change 4: Change JSON parsing library.
Justification: Original JSON library was limited in nature and did not support much. In order to be able to parse the JSON input string, we needed to change libraries and change the way we parsed JSON.

Change 5: Changed the input format of the trustworthiness module.
Justification: Needed to use the data from the "package.json" file from the npmjs package instead of a raw URL.

Change 6: Added JUnit testing suite features to other unreliable metrics of the module.
Justification: We used JUnit to check the reliability of the metrics to determine how much we needed to fix. We then used unit tests to ensure that everything worked after making significant changes to the module to support the new input type (using a json string instead of just a URL).

Baseline API

In your *Project 2 Plan* document, you described the system features and requirements you planned to implement. Here you will describe how things went.

Fill in the following table for each of the *baseline behavioral features* (e.g. "ingest a package") and the degree to which you've met each of them. Make one copy of the table per feature.

API_URL refers to the URL of the API.
Publicly available version of our automated end-to-end test suite can be found at the link below:
https://www.postman.com/winter-astronaut-506543/workspace/project-2-team-4-api/collection/18631555-055b7a24-fafd-40f8-8cb3-21e90877d35e

Tests in repository (Json format):
https://github.com/Purdue-ECE-461/project-2-project-2-4/blob/main/project%202%20rest%20API%20tests.postman_collection.json

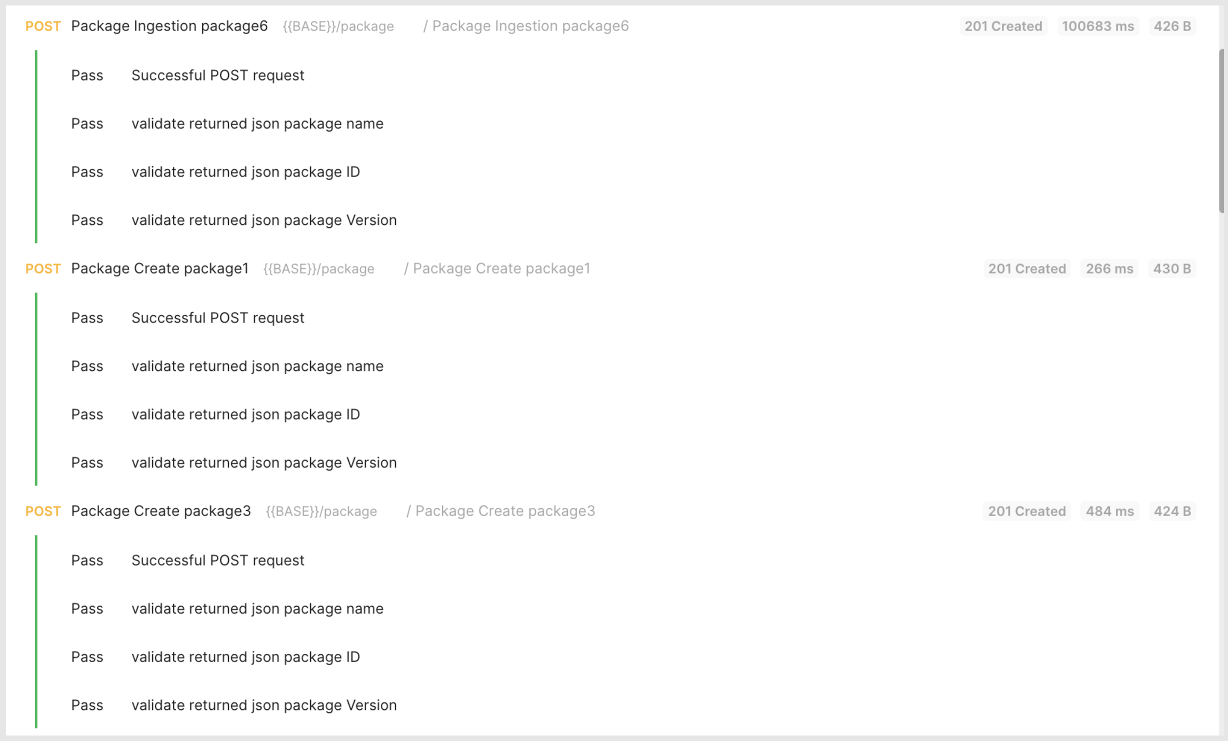
Most recent test results(Json format):
https://github.com/Purdue-ECE-461/project-2-project-2-4/blob/main/project%202%20rest%20API%20tests.postman_test_run.json

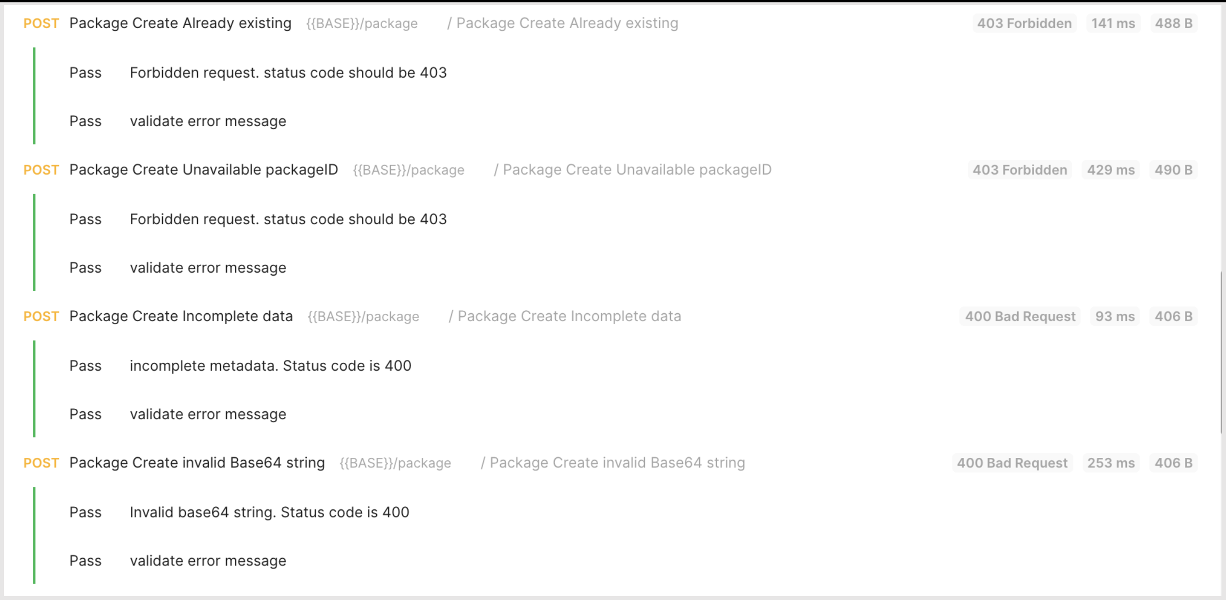
Feature: Ingest/Create Package
Relevant endpoint(s):
https://ece-461-project-2-team-4.uc.r.appspot.com/package

How completely is it implemented? We have completely implemented the ingestion and creation of packages. Packages will be created when raw content data is sent, and packages will be ingested and rated when GitHub links are sent to the system.

How did you validate it?

Endpoint | Verb(s) | Payload option(s)	*Validation approach(es)**	*Test records***
/package | POST | As JSON: package metadata, package content as 64-bit encoded string	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
/package | POST | As JSON: package metadata, publicly available GitHub link to package	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab


```
POST  Package Ingestion package6   {{BASE}}/package   / Package Ingestion package6     201 Created   100683 ms   426 B
      Pass   Successful POST request
      Pass   validate returned json package name
      Pass   validate returned json package ID
      Pass   validate returned json package Version
POST  Package Create package1   {{BASE}}/package   / Package Create package1           201 Created   266 ms   430 B
      Pass   Successful POST request
      Pass   validate returned json package name
      Pass   validate returned json package ID
      Pass   validate returned json package Version
POST  Package Create package3   {{BASE}}/package   / Package Create package3           201 Created   484 ms   424 B
      Pass   Successful POST request
      Pass   validate returned json package name
      Pass   validate returned json package ID
      Pass   validate returned json package Version
```

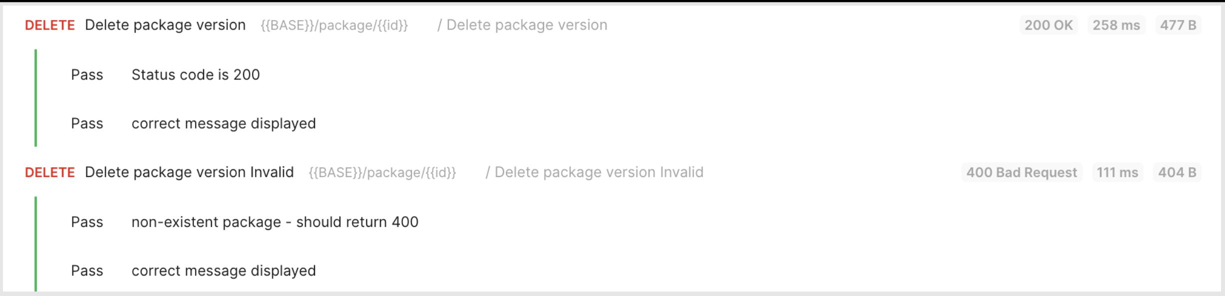


```
POST  Package Create Already existing   {{BASE}}/package   / Package Create Already existing        403 Forbidden   141 ms   488 B
      Pass   Forbidden request. status code should be 403
      Pass   validate error message
POST  Package Create Unavailable packageID   {{BASE}}/package   / Package Create Unavailable packageID   403 Forbidden   429 ms   490 B
      Pass   Forbidden request. status code should be 403
      Pass   validate error message
POST  Package Create Incomplete data   {{BASE}}/package   / Package Create Incomplete data          400 Bad Request   93 ms   406 B
      Pass   incomplete metadata. Status code is 400
      Pass   validate error message
POST  Package Create invalid Base64 string   {{BASE}}/package   / Package Create invalid Base64 string   400 Bad Request   253 ms   406 B
      Pass   Invalid base64 string. Status code is 400
      Pass   validate error message
```

Feature: Delete Package
Relevant endpoint(s):
https://ece-461-project-2-team-4.uc.r.appspot.com/package/byName/<string:packageName>
https://ece-461-project-2-team-4.uc.r.appspot.com/package/<string:packageId>

How completely is it implemented?
This feature is completely implemented. A single package can be deleted by sending the package ID in the URL, or all versions and ID's of a package can be deleted by sending the package name.
How did you validate it?

Endpoint | Verb(s) | Payload option(s)	*Validation approach(es)**	*Test records***
/package/byName/<string:packageName> | DELETE | <empty> no payload required, package name specified in URL	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
/package/<string:packageId> | DELETE | <empty> no payload required, package ID specified in URL	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab



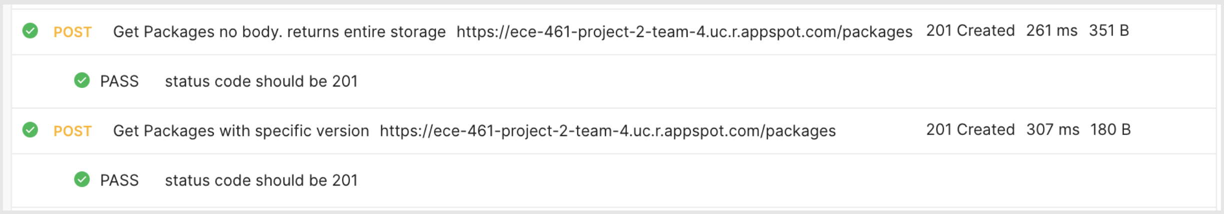
```
DELETE  Delete package version   {{BASE}}/package/{{id}}   / Delete package version                200 OK   258 ms   477 B
    Pass    Status code is 200
    Pass    correct message displayed
DELETE  Delete package version Invalid   {{BASE}}/package/{{id}}   / Delete package version Invalid   400 Bad Request   111 ms   404 B
    Pass    non-existent package - should return 400
    Pass    correct message displayed
```

Feature: View Packages in Paginated List
Relevant endpoint(s):
https://ece-461-project-2-team-4.uc.r.appspot.com/packages
https://ece-461-project-2-team-4.uc.r.appspot.com/packages?offset=<int offset>

How completely is it implemented?
This feature is completely implemented. If a user provides an offset, a paginated list of results at that page number is returned. If that page number is out of bounds, the nearest acceptable page is returned. Each page is limited to 10 results. If an empty payload is sent in the request, all packages are returned in a paginated format. If a package name and version are sent, then only packages matching that name and version are returned. Version logic such as (*, ^, ~, >, >=, <, <=, x.y.z-x.y.z) is supported.
How did you validate it?

Endpoint | Verb(s) | Payload option(s)	*Validation approach(es)**	*Test records***
/packages | POST | <empty>, will return all packages in system in paginated format	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
/packages | POST | As JSON: Package name, package version, will return all packages that match name and version	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
/packages?offset=<int offset> | POST | <empty>, will return all packages in system in paginated format using offset as page number, or closest valid page number	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
/packages?offset=<int offset> | POST | As JSON: Package name, package version, will return all packages that match name and version and returns	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab

paginated list using offset as page number, or closest valid page number		

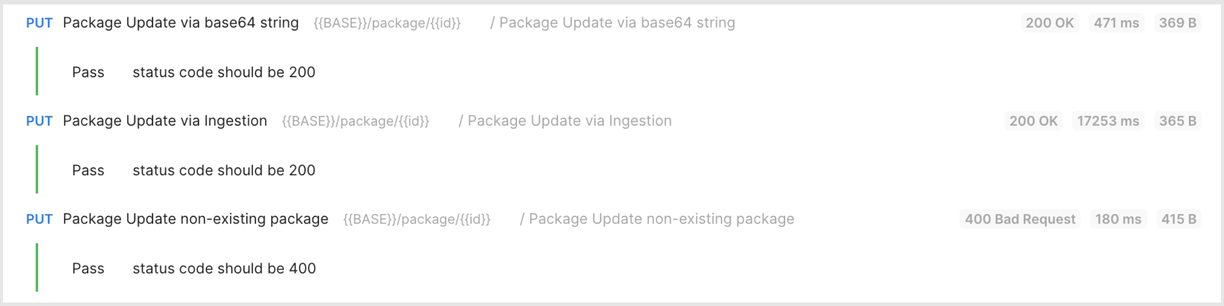


```
POST  Get Packages no body. returns entire storage  https://ece-461-project-2-team-4.uc.r.appspot.com/packages  201 Created  261 ms  351 B
      PASS  status code should be 201
POST  Get Packages with specific version  https://ece-461-project-2-team-4.uc.r.appspot.com/packages  201 Created  307 ms  180 B
      PASS  status code should be 201
```

Feature: Update Package
Relevant endpoint(s):
https://ece-461-project-2-team-4.uc.r.appspot.com/package/<string:packageId>

How completely is it implemented?
This feature is fully implemented and will update a package given the package's ID and the content to update it with.
How did you validate it?

Endpoint | Verb(s) | Payload option(s)	*Validation approach(es)**	*Test records***
/package/<string:packageId> | PUT | Package ID in URL and as JSON: package metadata, package content	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
/package/<string:packageId> | PUT | Package ID in URL and as JSON: package metadata, GitHub URL	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab

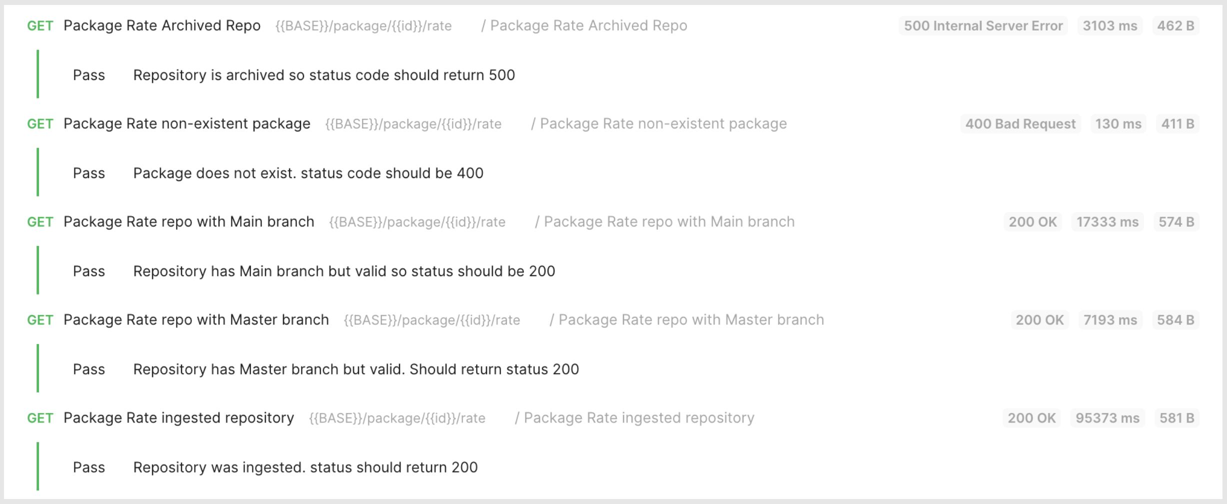


```
PUT  Package Update via base64 string  {{BASE}}/package/{{id}}  / Package Update via base64 string  200 OK  471 ms  369 B
     Pass  status code should be 200
PUT  Package Update via Ingestion  {{BASE}}/package/{{id}}  / Package Update via Ingestion  200 OK  17253 ms  365 B
     Pass  status code should be 200
PUT  Package Update non-existing package  {{BASE}}/package/{{id}}  / Package Update non-existing package  400 Bad Request  180 ms  415 B
     Pass  status code should be 400
```

Feature: Rate Package
Relevant endpoint(s):
https://ece-461-project-2-team-4.uc.r.appspot.com/package/<string:packageId>/rate

How completely is it implemented?
This feature is fully implemented. Sending the package ID with this URL will return the metrics from the trustworthiness module.
How did you validate it?

Endpoint | Verb(s) | Payload option(s)	*Validation approach(es)**	*Test records***
package/<string:packageId>/rate | GET | Rate Archived Repo	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
package/<string:packageId>/rate | GET | Rate Non-existent package	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
package/<string:packageId>/rate | GET | Rate repo with main branch	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
package/<string:packageId>/rate | GET | Rate repo with master branch	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab
package/<string:packageId>/rate | GET | Rate ingested repo	Automated end-to-end tests	Postman Collection, for test results, go to "Monitor" tab



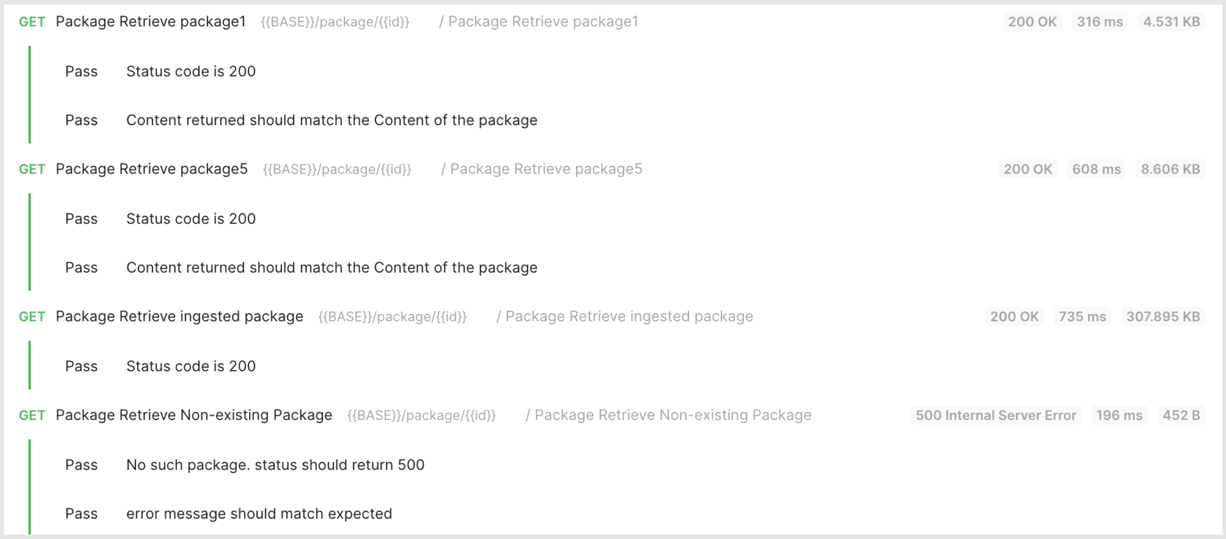
```
GET  Package Rate Archived Repo  {{BASE}}/package/{{id}}/rate  / Package Rate Archived Repo  500 Internal Server Error  3103 ms  462 B
     Pass  Repository is archived so status code should return 500
GET  Package Rate non-existent package  {{BASE}}/package/{{id}}/rate  / Package Rate non-existent package  400 Bad Request  130 ms  411 B
     Pass  Package does not exist. status code should be 400
GET  Package Rate repo with Main branch  {{BASE}}/package/{{id}}/rate  / Package Rate repo with Main branch  200 OK  17333 ms  574 B
     Pass  Repository has Main branch but valid so status should be 200
GET  Package Rate repo with Master branch  {{BASE}}/package/{{id}}/rate  / Package Rate repo with Master branch  200 OK  7193 ms  584 B
     Pass  Repository has Master branch but valid. Should return status 200
GET  Package Rate ingested repository  {{BASE}}/package/{{id}}/rate  / Package Rate ingested repository  200 OK  95373 ms  581 B
     Pass  Repository was ingested. status should return 200
```

Feature: Retrieve Package by ID
Relevant endpoint(s):
https://ece-461-project-2-team-4.uc.r.appspot.com/package/<string:packageId>

How completely is it implemented?
This feature is fully implemented. Given a package ID and a GET request, the system will return the content and metadata of a package.

How did you validate it? *(Fill out this table for the feature – This should include the relevant kinds of error cases you tested)*

Endpoint | Verb(s) | Payload option(s)	*Validation approach(es)**	*Test records***
/package/<string:packageId> | GET | packageID as part of URL request.	Automated end-to-end tests	[Postman Collection, for test results, go to "Monitor" tab]()

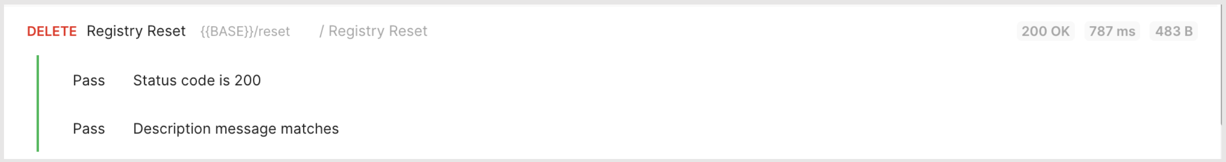


```
GET  Package Retrieve package1  {{BASE}}/package/{{id}}   / Package Retrieve package1   200 OK  316 ms  4.531 KB
     Pass   Status code is 200
     Pass   Content returned should match the Content of the package
GET  Package Retrieve package5  {{BASE}}/package/{{id}}   / Package Retrieve package5   200 OK  608 ms  8.606 KB
     Pass   Status code is 200
     Pass   Content returned should match the Content of the package
GET  Package Retrieve ingested package  {{BASE}}/package/{{id}}   / Package Retrieve ingested package   200 OK  735 ms  307.895 KB
     Pass   Status code is 200
GET  Package Retrieve Non-existing Package  {{BASE}}/package/{{id}}   / Package Retrieve Non-existing Package   500 Internal Server Error  196 ms  452 B
     Pass   No such package. status should return 500
     Pass   error message should match expected
```

Feature: Reset System
Relevant endpoint(s):
https://ece-461-project-2-team-4.uc.r.appspot.com/reset

How completely is it implemented?
This function is entirely implemented. No known instances of failure to wipe the Cloud Storage Bucket.
How did you validate it?

Endpoint | Verb(s) | Payload option(s)	*Validation approach(es)**	*Test records***
/reset | DELETE | <empty>	Automated end-to-end tests	[Postman Collection, for test results, go to "Monitor" tab]()



```
DELETE  Registry Reset  {{BASE}}/reset   / Registry Reset   200 OK  787 ms  483 B
     Pass   Status code is 200
     Pass   Description message matches
```

*Validation approaches: For example, (None ; Manual ; Automated unit tests ; Automated end-to-end tests)

**Test records: For example, link to the relevant tests in your repository; link to the most recent relevant run of your CI; write the date of the most recent manual test.

Non-baseline: API

If you implemented additional behaviors in your team's web API, (e.g. package groups; traceability features; security features) they should be reflected in your team's OpenAPI specification on GitHub.

Provide a link to your team's OpenAPI specification:

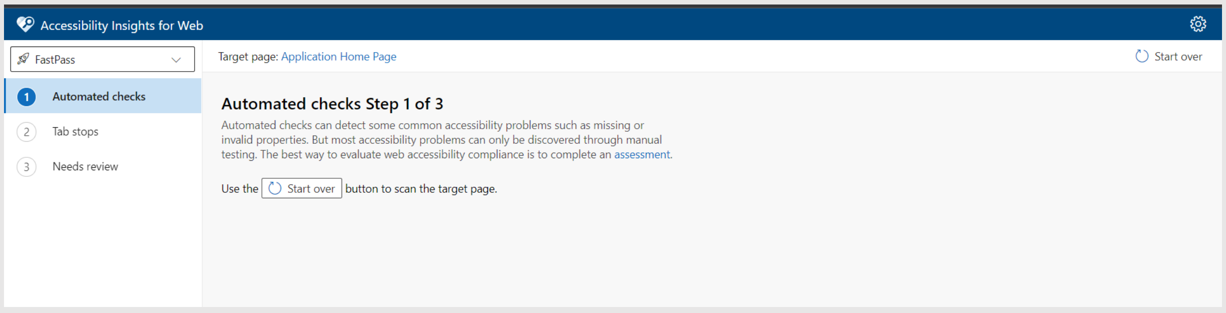
apicur.io page: https://studio-ws.apicur.io/sharing/ad9fc925-c1eb-400d-a7b7-7fc005e4a5ca

Non-baseline: Browser-based interface

If you implemented a browser-based web interface, it should be ADA-compliant (WCAG 2.1 at level AA).

1. Show the result of the automated tests implemented by https://github.com/microsoft/accessibility-insights-web, as applied to each of your web page(s). Screenshots are fine.

Page 1: https://ece-461-project-2-team-4.uc.r.appspot.com/home
Outcome of tests:

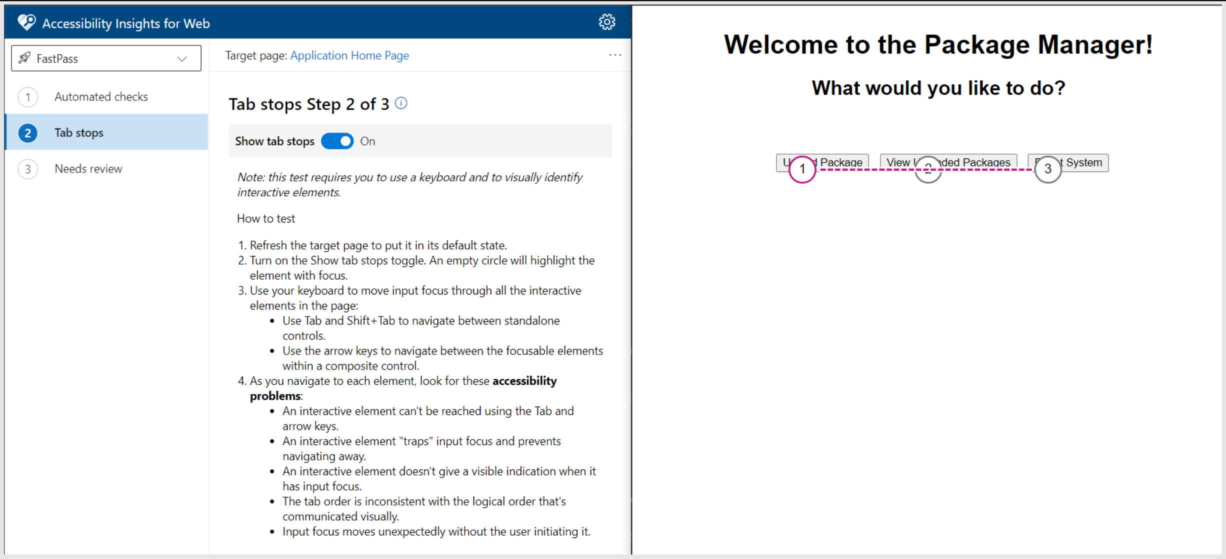


Target page: Application Home Page

Automated checks Step 1 of 3

Automated checks can detect some common accessibility problems such as missing or invalid properties. But most accessibility problems can only be discovered through manual testing. The best way to evaluate web accessibility compliance is to complete an assessment.

Use the Start over button to scan the target page.


Accessibility Insights for Web
FastPass
1 Automated checks
2 Tab stops
3 Needs review
Welcome to the Package Manager!
What would you like to do?
1
2
3

Target page: Application Home Page

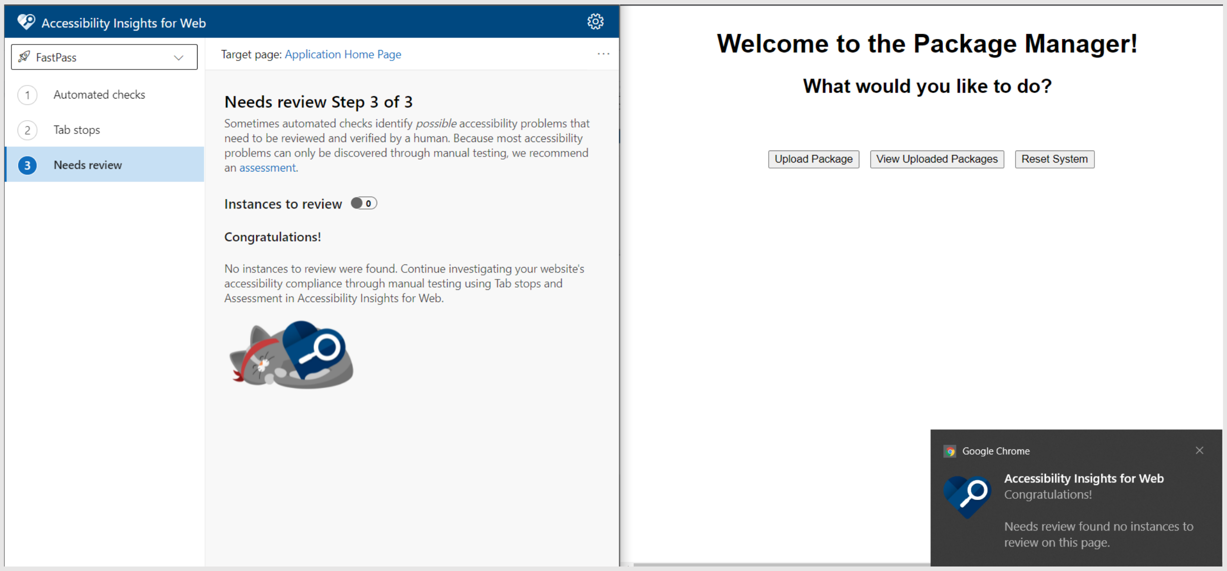
Tab stops Step 2 of 3

Show tab stops: On

Note: this test requires you to use a keyboard and to visually identify interactive elements.

How to test

1. Refresh the target page to put it in its default state.
2. Turn on the Show tab stops toggle. An empty circle will highlight the element with focus.
3. Use your keyboard to move input focus through all the interactive elements in the page:
   - Use Tab and Shift+Tab to navigate between standalone controls.
   - Use the arrow keys to navigate between the focusable elements within a composite control.
4. As you navigate to each element, look for these **accessibility problems**:
   - An interactive element can't be reached using the Tab and arrow keys.
   - An interactive element "traps" input focus and prevents navigating away.
   - An interactive element doesn't give a visible indication when it has input focus.
   - The tab order is inconsistent with the logical order that's communicated visually.
   - Input focus moves unexpectedly without the user initiating it.



Target page: Application Home Page

Needs review Step 3 of 3

Sometimes automated checks identify *possible* accessibility problems that need to be reviewed and verified by a human. Because most accessibility problems can only be discovered through manual testing, we recommend an assessment.

Instances to review 0

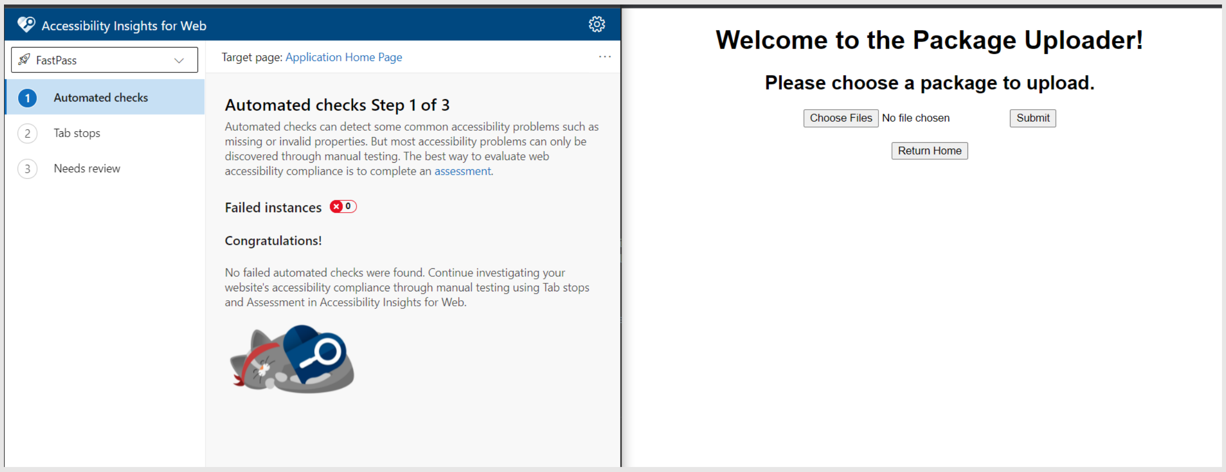
Congratulations!

No instances to review were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Accessibility Insights for Web
Congratulations!

Needs review found no instances to review on this page.

Page 2: https://ece-461-project-2-team-4.uc.r.appspot.com/home/upload-package
Outcome of tests:


Accessibility Insights for Web
FastPass
1 Automated checks
2 Tab stops
3 Needs review
Welcome to the Package Uploader!
Please choose a package to upload.
Choose Files
No file chosen
Submit
Return Home

Target page: Application Home Page

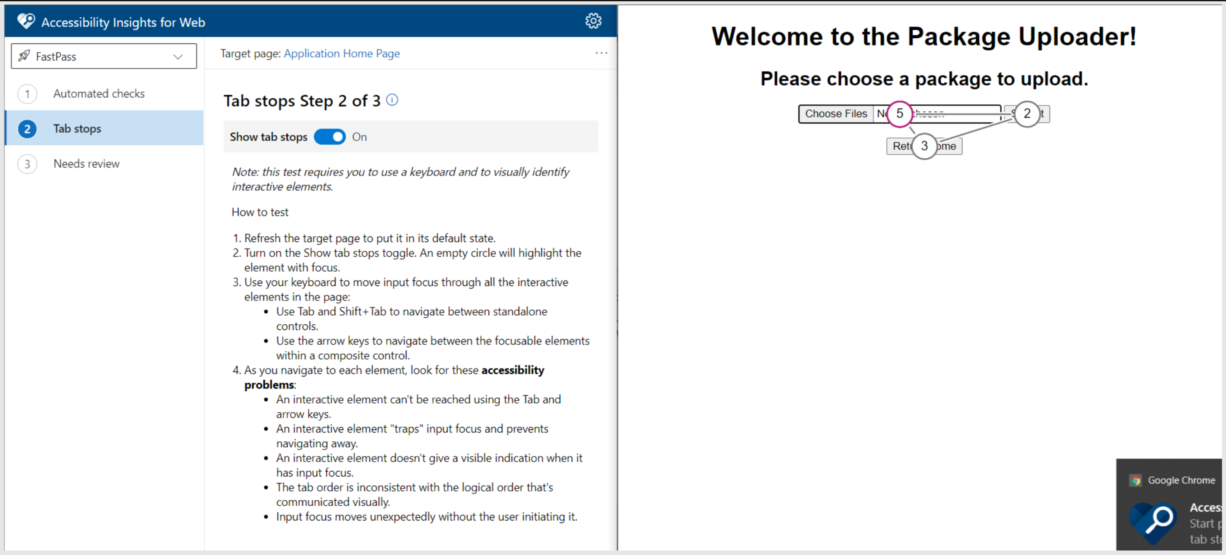
Automated checks Step 1 of 3

Automated checks can detect some common accessibility problems such as missing or invalid properties. But most accessibility problems can only be discovered through manual testing. The best way to evaluate web accessibility compliance is to complete an assessment.

Failed instances 0

Congratulations!

No failed automated checks were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.



Accessibility Insights for Web

FastPass

1 Automated checks

2 Tab stops

3 Needs review

Target page: Application Home Page

Tab stops Step 2 of 3

Show tab stops On

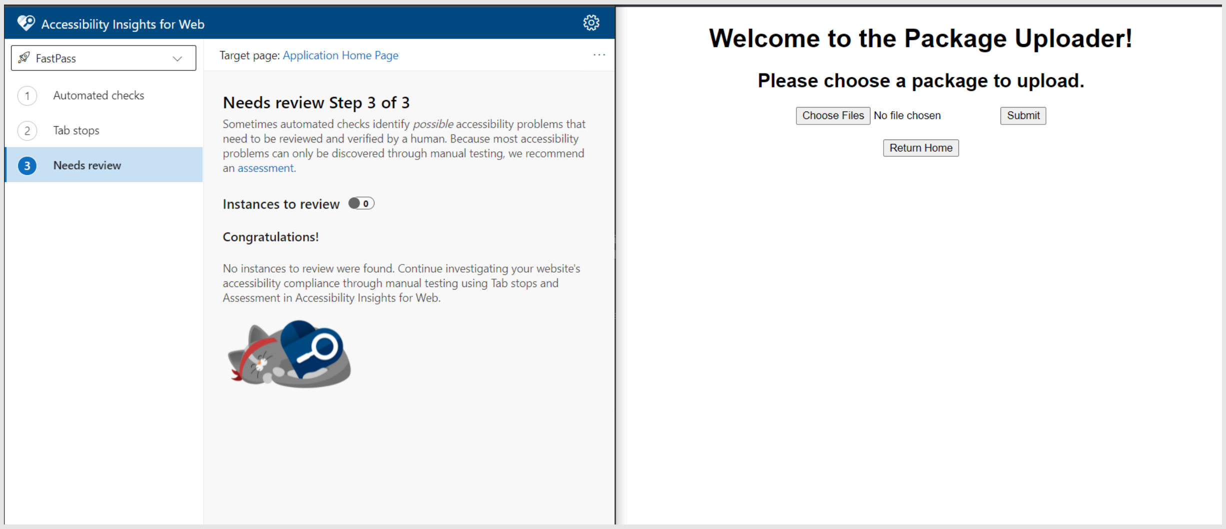
Note: this test requires you to use a keyboard and to visually identify interactive elements.

How to test

1. Refresh the target page to put it in its default state.
2. Turn on the Show tab stops toggle. An empty circle will highlight the element with focus.
3. Use your keyboard to move input focus through all the interactive elements in the page:
   - Use Tab and Shift+Tab to navigate between standalone controls.
   - Use the arrow keys to navigate between the focusable elements within a composite control.
4. As you navigate to each element, look for these **accessibility problems**:
   - An interactive element can't be reached using the Tab and arrow keys.
   - An interactive element "traps" input focus and prevents navigating away.
   - An interactive element doesn't give a visible indication when it has input focus.
   - The tab order is inconsistent with the logical order that's communicated visually.
   - Input focus moves unexpectedly without the user initiating it.

Welcome to the Package Uploader!

Please choose a package to upload.



Accessibility Insights for Web

FastPass

1 Automated checks

2 Tab stops

3 Needs review

Target page: Application Home Page

Needs review Step 3 of 3

Sometimes automated checks identify *possible* accessibility problems that need to be reviewed and verified by a human. Because most accessibility problems can only be discovered through manual testing, we recommend an assessment.

Instances to review 0

Congratulations!

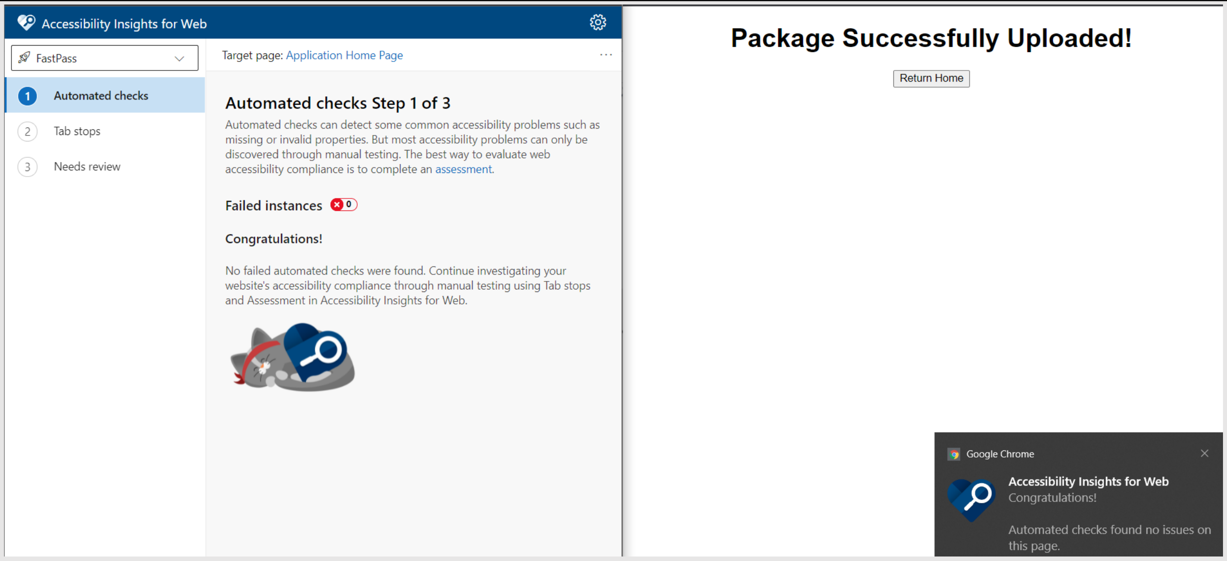
No instances to review were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Welcome to the Package Uploader!

Please choose a package to upload.

Page 3: https://ece-461-project-2-team-4.uc.r.appspot.com/home/upload

Outcome of tests:



Accessibility Insights for Web

FastPass

1 Automated checks
2 Tab stops
3 Needs review

Target page: Application Home Page

Automated checks Step 1 of 3

Automated checks can detect some common accessibility problems such as missing or invalid properties. But most accessibility problems can only be discovered through manual testing. The best way to evaluate web accessibility compliance is to complete an assessment.

Failed instances 0

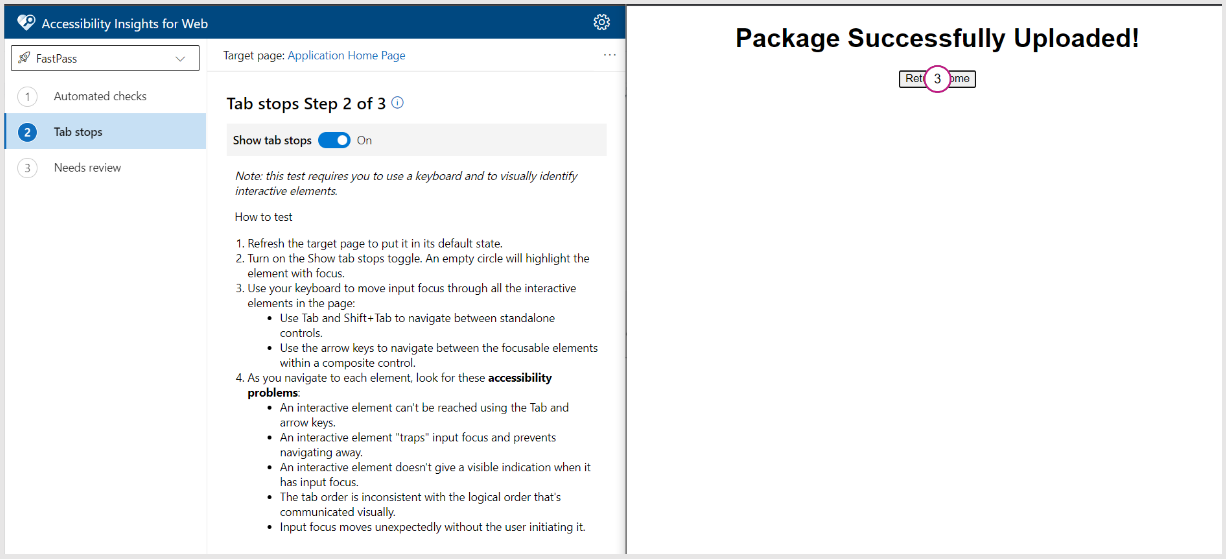
Congratulations!

No failed automated checks were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Package Successfully Uploaded!

Return Home

Google Chrome
Accessibility Insights for Web
Congratulations!
Automated checks found no issues on this page.



Accessibility Insights for Web

FastPass

1 Automated checks
2 Tab stops
3 Needs review

Target page: Application Home Page

Tab stops Step 2 of 3

Show tab stops On

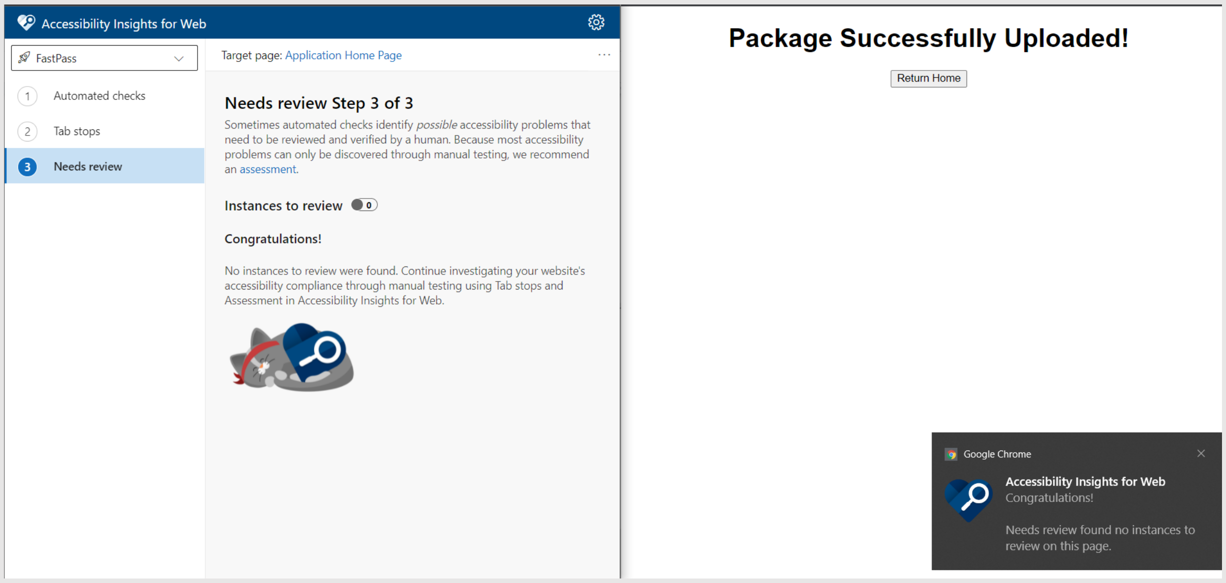
Note: this test requires you to use a keyboard and to visually identify interactive elements.

How to test

1. Refresh the target page to put it in its default state.
2. Turn on the Show tab stops toggle. An empty circle will highlight the element with focus.
3. Use your keyboard to move input focus through all the interactive elements in the page:
   - Use Tab and Shift+Tab to navigate between standalone controls.
   - Use the arrow keys to navigate between the focusable elements within a composite control.
4. As you navigate to each element, look for these **accessibility problems**:
   - An interactive element can't be reached using the Tab and arrow keys.
   - An interactive element "traps" input focus and prevents navigating away.
   - An interactive element doesn't give a visible indication when it has input focus.
   - The tab order is inconsistent with the logical order that's communicated visually.
   - Input focus moves unexpectedly without the user initiating it.

Package Successfully Uploaded!

Ret 3 ome



Accessibility Insights for Web

FastPass

1 Automated checks
2 Tab stops
3 Needs review

Target page: Application Home Page

Needs review Step 3 of 3

Sometimes automated checks identify *possible* accessibility problems that need to be reviewed and verified by a human. Because most accessibility problems can only be discovered through manual testing, we recommend an assessment.

Instances to review 0

Congratulations!

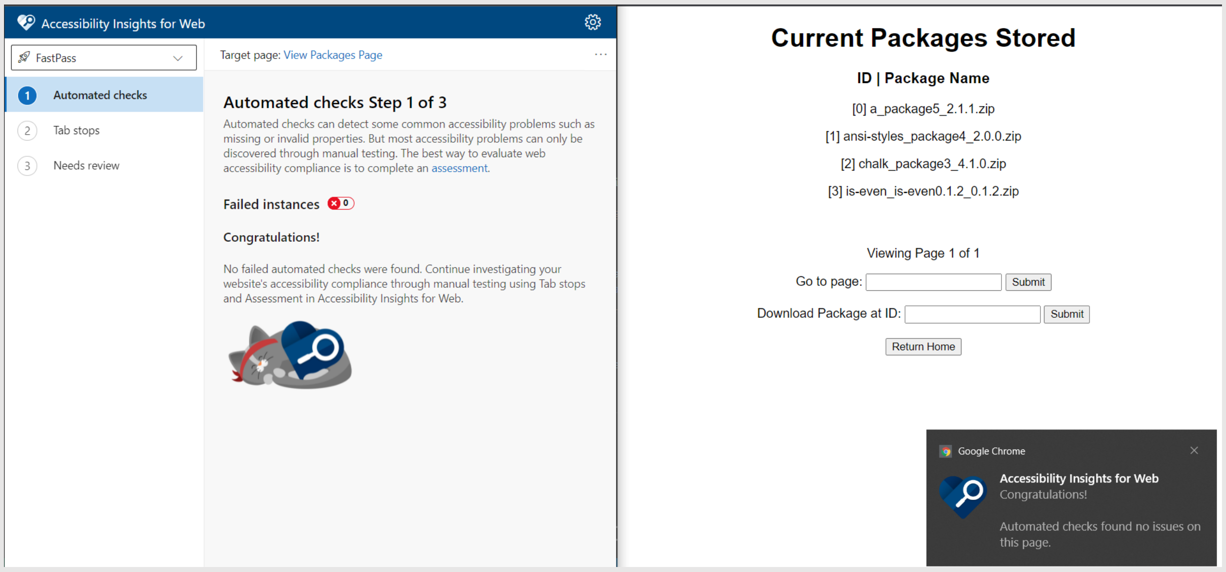
No instances to review were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Package Successfully Uploaded!

Return Home

Google Chrome
Accessibility Insights for Web
Congratulations!
Needs review found no instances to review on this page.

Page 4: https://ece-461-project-2-team-4.uc.r.appspot.com/home/view-packages
Outcome of tests:


Accessibility Insights for Web
FastPass
Target page: View Packages Page
1 Automated checks
2 Tab stops
3 Needs review

Automated checks Step 1 of 3

Automated checks can detect some common accessibility problems such as missing or invalid properties. But most accessibility problems can only be discovered through manual testing. The best way to evaluate web accessibility compliance is to complete an assessment.

Failed instances 0

Congratulations!

No failed automated checks were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Current Packages Stored

ID | Package Name

[0] a_package5_2.1.1.zip
[1] ansi-styles_package4_2.0.0.zip
[2] chalk_package3_4.1.0.zip
[3] is-even_is-even0.1.2_0.1.2.zip

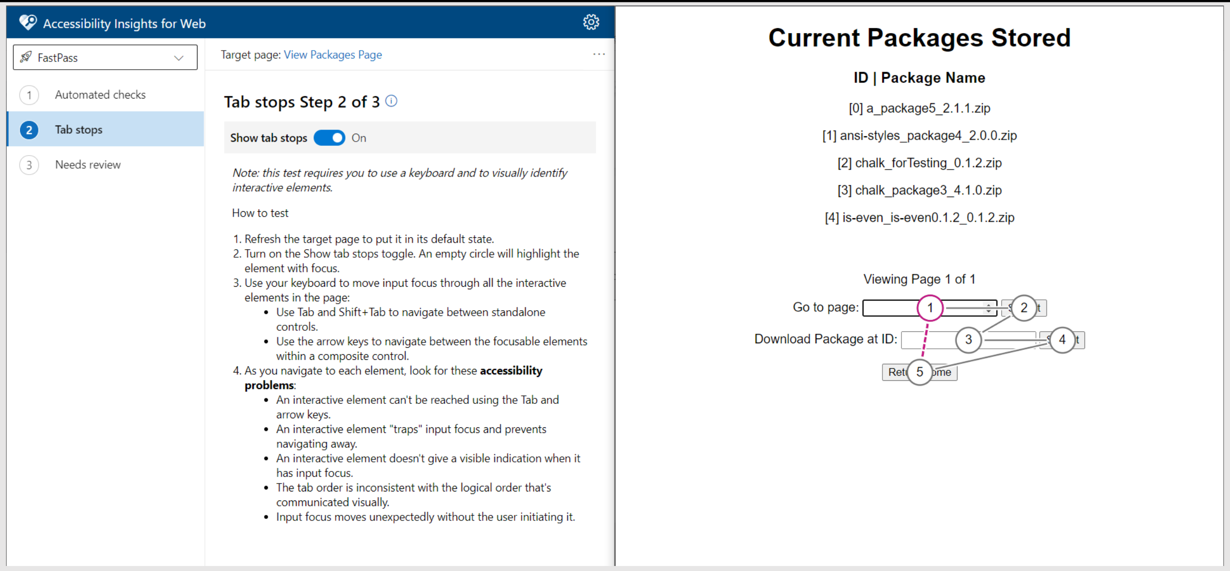
Viewing Page 1 of 1

Go to page: Submit

Download Package at ID: Submit

Return Home

Google Chrome
Accessibility Insights for Web
Congratulations!
Automated checks found no issues on this page.



Accessibility Insights for Web

FastPass

1 Automated checks

2 Tab stops

3 Needs review

Target page: View Packages Page

Tab stops Step 2 of 3

Show tab stops On

Note: this test requires you to use a keyboard and to visually identify interactive elements.

How to test

1. Refresh the target page to put it in its default state.
2. Turn on the Show tab stops toggle. An empty circle will highlight the element with focus.
3. Use your keyboard to move input focus through all the interactive elements in the page:
   - Use Tab and Shift+Tab to navigate between standalone controls.
   - Use the arrow keys to navigate between the focusable elements within a composite control.
4. As you navigate to each element, look for these **accessibility problems**:
   - An interactive element can't be reached using the Tab and arrow keys.
   - An interactive element "traps" input focus and prevents navigating away.
   - An interactive element doesn't give a visible indication when it has input focus.
   - The tab order is inconsistent with the logical order that's communicated visually.
   - Input focus moves unexpectedly without the user initiating it.

Current Packages Stored

ID | Package Name

[0] a_package5_2.1.1.zip

[1] ansi-styles_package4_2.0.0.zip

[2] chalk_forTesting_0.1.2.zip

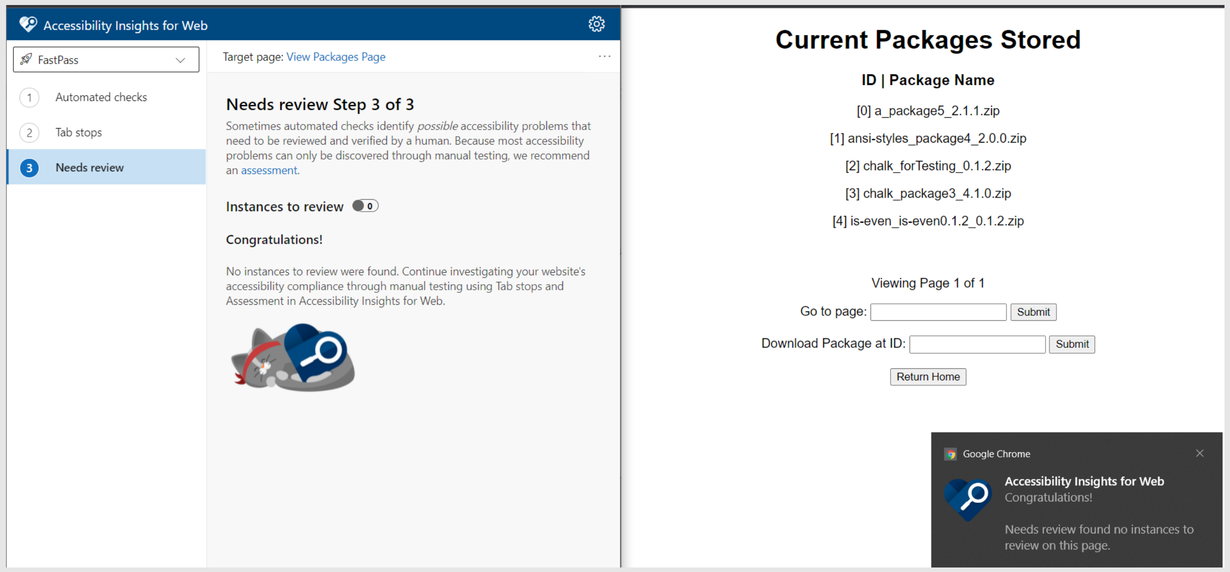
[3] chalk_package3_4.1.0.zip

[4] is-even_is-even0.1.2_0.1.2.zip

Viewing Page 1 of 1

Go to page:

Download Package at ID:



Accessibility Insights for Web

FastPass

1 Automated checks

2 Tab stops

3 Needs review

Target page: View Packages Page

Needs review Step 3 of 3

Sometimes automated checks identify *possible* accessibility problems that need to be reviewed and verified by a human. Because most accessibility problems can only be discovered through manual testing, we recommend an assessment.

Instances to review 0

Congratulations!

No instances to review were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Current Packages Stored

ID | Package Name

[0] a_package5_2.1.1.zip

[1] ansi-styles_package4_2.0.0.zip

[2] chalk_forTesting_0.1.2.zip

[3] chalk_package3_4.1.0.zip

[4] is-even_is-even0.1.2_0.1.2.zip

Viewing Page 1 of 1

Go to page: Submit

Download Package at ID: Submit

Return Home

Google Chrome

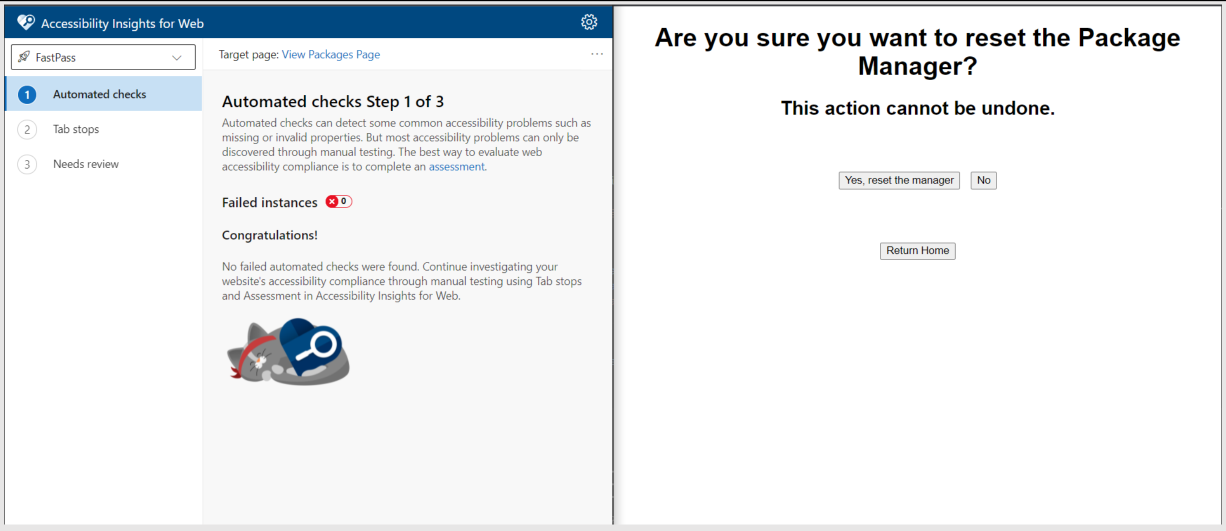
Accessibility Insights for Web

Congratulations!

Needs review found no instances to review on this page.

Page 5: https://ece-461-project-2-team-4.uc.r.appspot.com/home/reset

Outcome of tests:



Accessibility Insights for Web

FastPass

Target page: View Packages Page

1 Automated checks
2 Tab stops
3 Needs review

Automated checks Step 1 of 3

Automated checks can detect some common accessibility problems such as missing or invalid properties. But most accessibility problems can only be discovered through manual testing. The best way to evaluate web accessibility compliance is to complete an assessment.

Failed instances 0

Congratulations!

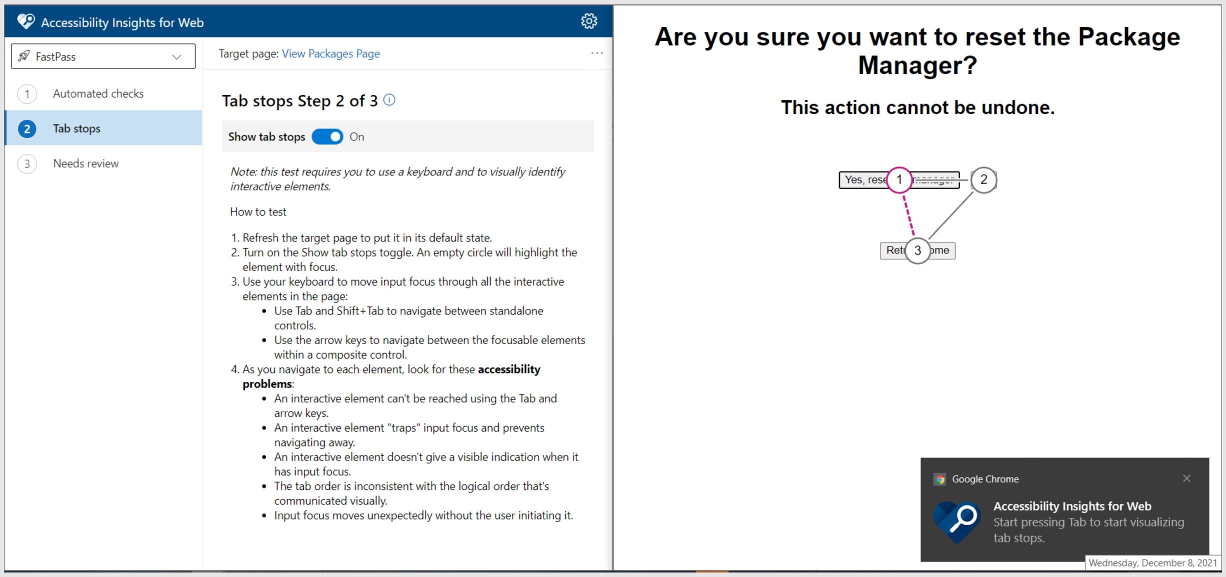
No failed automated checks were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Are you sure you want to reset the Package Manager?

This action cannot be undone.

Yes, reset the manager | No

Return Home



Accessibility Insights for Web

FastPass

Target page: View Packages Page

1 Automated checks
2 Tab stops
3 Needs review

Tab stops Step 2 of 3

Show tab stops On

Note: this test requires you to use a keyboard and to visually identify interactive elements.

How to test

1. Refresh the target page to put it in its default state.
2. Turn on the Show tab stops toggle. An empty circle will highlight the element with focus.
3. Use your keyboard to move input focus through all the interactive elements in the page:
    - Use Tab and Shift+Tab to navigate between standalone controls.
    - Use the arrow keys to navigate between the focusable elements within a composite control.
4. As you navigate to each element, look for these **accessibility problems**:
    - An interactive element can't be reached using the Tab and arrow keys.
    - An interactive element "traps" input focus and prevents navigating away.
    - An interactive element doesn't give a visible indication when it has input focus.
    - The tab order is inconsistent with the logical order that's communicated visually.
    - Input focus moves unexpectedly without the user initiating it.

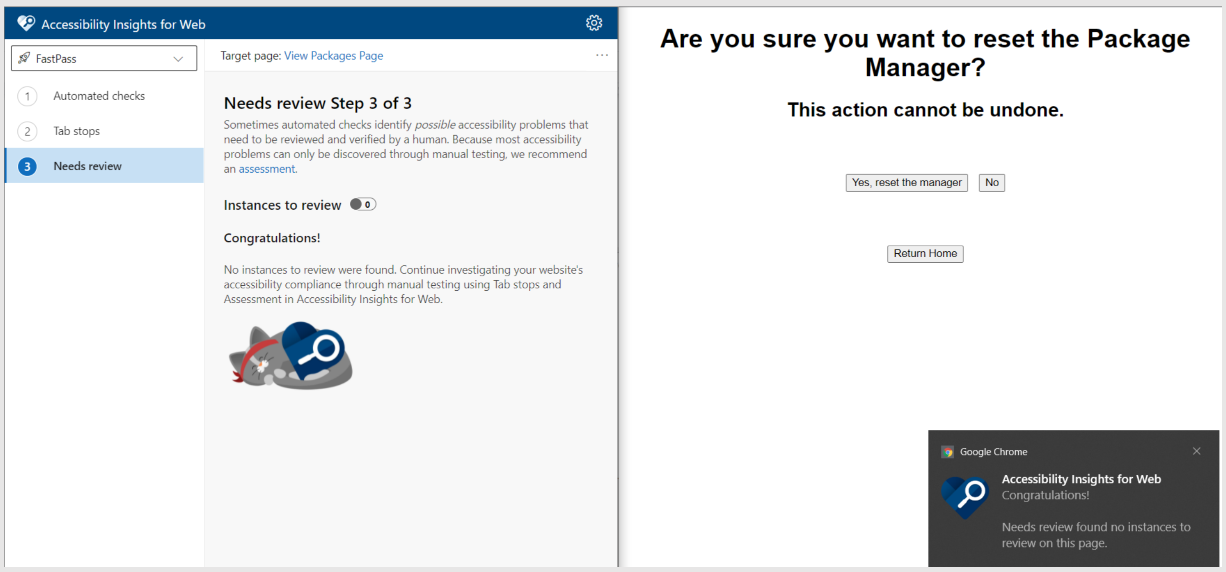
Are you sure you want to reset the Package Manager?

This action cannot be undone.

Google Chrome

Accessibility Insights for Web
Start pressing Tab to start visualizing tab stops.

Wednesday, December 8, 2021



Accessibility Insights for Web

FastPass

Target page: View Packages Page

1 Automated checks
2 Tab stops
3 Needs review

Needs review Step 3 of 3

Sometimes automated checks identify *possible* accessibility problems that need to be reviewed and verified by a human. Because most accessibility problems can only be discovered through manual testing, we recommend an assessment.

Instances to review 0

Congratulations!

No instances to review were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Are you sure you want to reset the Package Manager?

This action cannot be undone.

Yes, reset the manager | No

Return Home

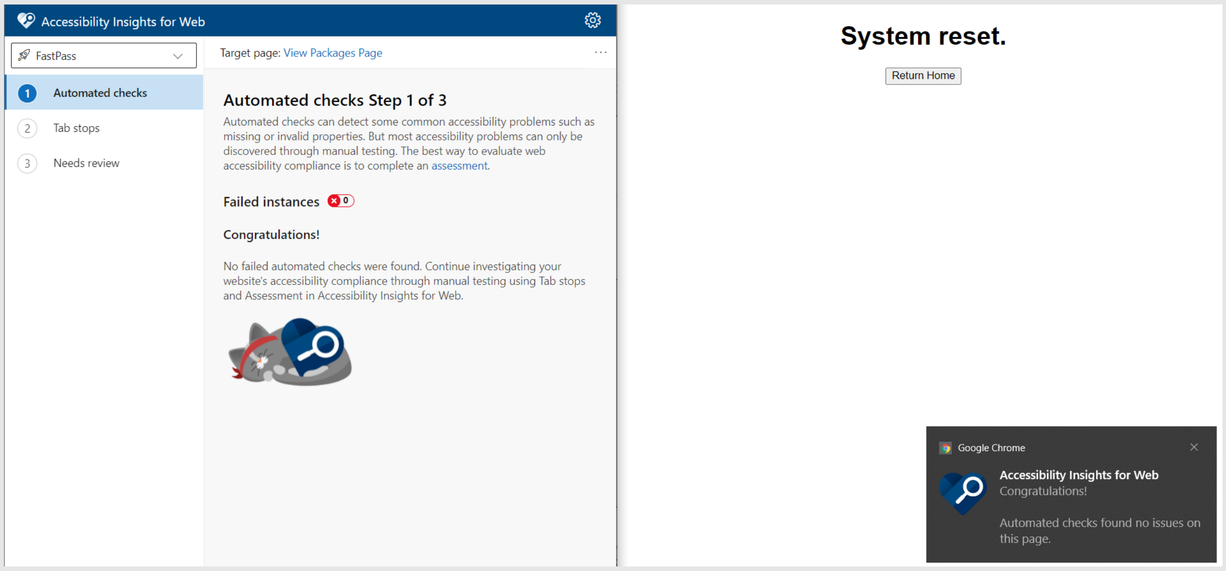
Google Chrome

Accessibility Insights for Web
Congratulations!

Needs review found no instances to review on this page.

Page 6: https://ece-461-project-2-team-4.uc.r.appspot.com/home/reset

Outcome of tests:



Accessibility Insights for Web

FastPass

1 Automated checks

2 Tab stops

3 Needs review

Target page: View Packages Page

Automated checks Step 1 of 3

Automated checks can detect some common accessibility problems such as missing or invalid properties. But most accessibility problems can only be discovered through manual testing. The best way to evaluate web accessibility compliance is to complete an assessment.

Failed instances ✕ 0

Congratulations!

No failed automated checks were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

System reset.

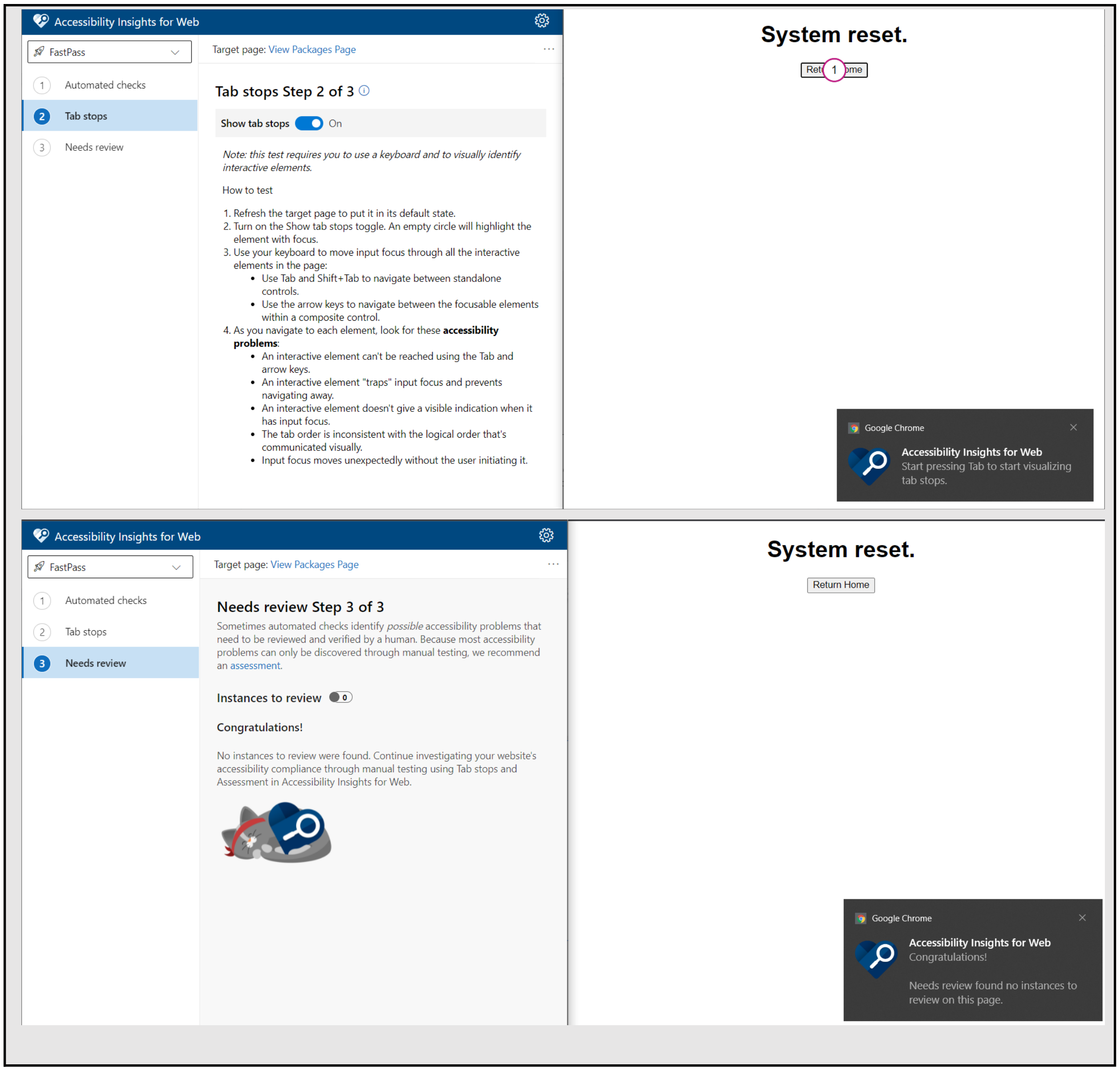
Return Home

Google Chrome

Accessibility Insights for Web

Congratulations!

Automated checks found no issues on this page.



Target page: View Packages Page

Tab stops Step 2 of 3

Show tab stops On

Note: this test requires you to use a keyboard and to visually identify interactive elements.

How to test

1. Refresh the target page to put it in its default state.
2. Turn on the Show tab stops toggle. An empty circle will highlight the element with focus.
3. Use your keyboard to move input focus through all the interactive elements in the page:
   - Use Tab and Shift+Tab to navigate between standalone controls.
   - Use the arrow keys to navigate between the focusable elements within a composite control.
4. As you navigate to each element, look for these **accessibility problems**:
   - An interactive element can't be reached using the Tab and arrow keys.
   - An interactive element "traps" input focus and prevents navigating away.
   - An interactive element doesn't give a visible indication when it has input focus.
   - The tab order is inconsistent with the logical order that's communicated visually.
   - Input focus moves unexpectedly without the user initiating it.

Accessibility Insights for Web
Start pressing Tab to start visualizing tab stops.

Target page: View Packages Page

Needs review Step 3 of 3

Sometimes automated checks identify *possible* accessibility problems that need to be reviewed and verified by a human. Because most accessibility problems can only be discovered through manual testing, we recommend an assessment.

Instances to review 0

Congratulations!

No instances to review were found. Continue investigating your website's accessibility compliance through manual testing using Tab stops and Assessment in Accessibility Insights for Web.

Accessibility Insights for Web
Congratulations!

Needs review found no instances to review on this page.

2. Describe any additional steps you took to consider accessibility (e.g. educational resources you consulted; design choices you made; implementation decisions that are not covered by Microsoft's automated tests).

In order to investigate how the front end should be more accessible, a few design choices were made and a resource was consulted. According to the Bureau of Internet Accessibility, "Currently, Section 508 of the Rehabilitation Act of 1973 does not specify the requirements for choosing an accessible website typeface. However, the US Department of Health & Human Services unofficially recommends the following fonts for PDF files: Times New Roman, Verdana, Arial, Tahoma, Helvetica, and Calibri" (Bureau of Internet Accessibility, 2017). Based on this recommendation, all fonts on the front end use Helvetica to ensure ease of use.

In addition, large header fonts are used to make sure it is readable without needing to zoom. All URLs for the front end contain simple, memorizable words to ensure they can easily be navigated back to. Black font on a white background was chosen to maximize the contrast between the two. Another design decision was to center the content. Users will expect to see their content directly in front of them, especially on mobile devices. Centering horizontally allows us to ensure no elements are missed off to the side.

Non-functional requirements

Baseline

Security: STRIDE analysis

The terms used in this section are defined here.

System model. Present one or more data-flow diagrams of your deployed system. (A whiteboard picture is fine, but use the correct symbols please).

- You may provide multiple DFDs to capture different aspects of the system.
- You may indicate multiple trust boundaries, e.g. for different classes of users.
- Each diagram should indicate at least the following entities: data flow; data store; process; trust boundary. You may include interactors and multi-process if needed.
- Each diagram should number the entities for reference later on.

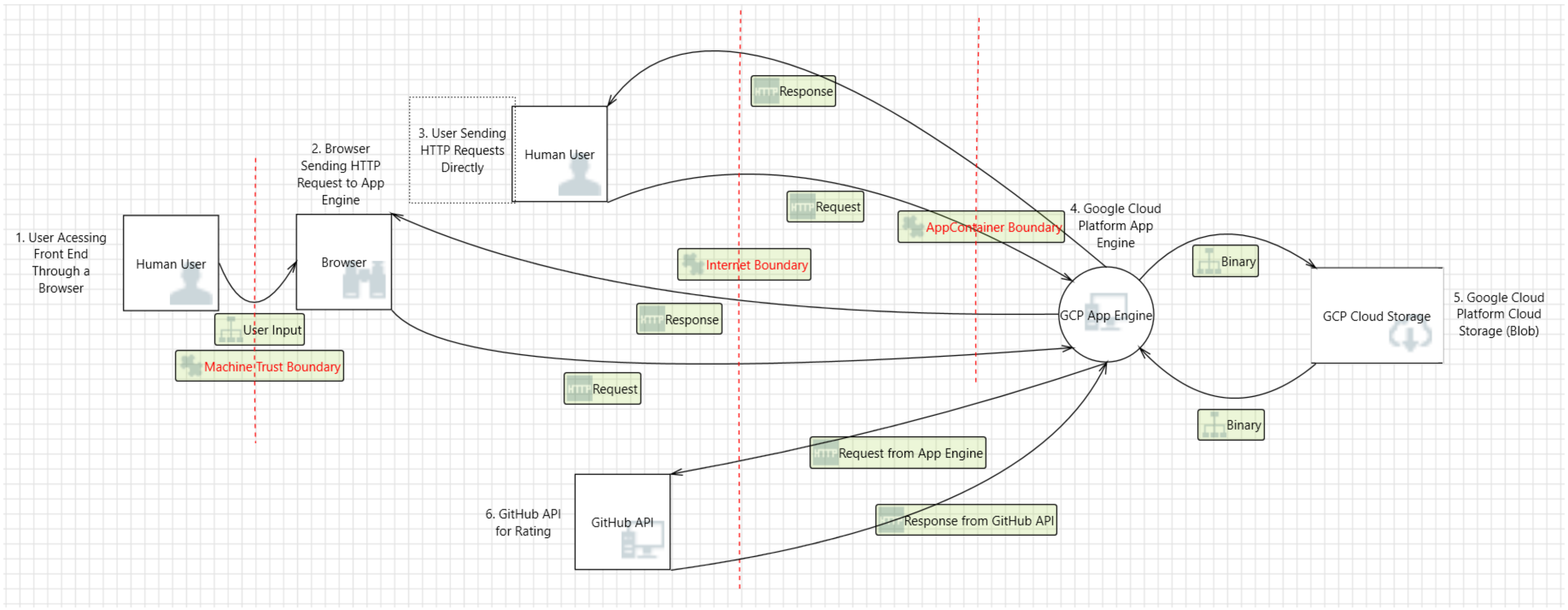


Diagram 1: System Model. Generated using Microsoft Threat Modeling Tool [https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool] (1) *User Accessing Front End Through a Browser,* (2) *Browser Sending HTTP Request to App Engine,* (3) *User Sending HTTP Requests Directly using API,* (4) *Google Cloud Platform App Engine,* (5) *Google Cloud Platform Cloud Storage (Blob Storage),* (6) *Github API for Rating*

For each trust boundary indicated, describe the nature of the untrusted party involved (e.g. “outsider threat [e.g. external hacker]” or “insider threat [e.g. ACME employee with valid credentials]” or “infrastructure provider threat [GCP]”).

Trust boundary #: 1 - Machine Trust Boundary
Untrusted party: Insider threat. Someone who is not authorized to be on the Package Manager could use an unlocked computer within ACME to access the service.

Trust boundary #: 2 - Internet Boundary
Untrusted party: Outsider threat. Information could be intercepted during transmission of data in either direction, leading to possibly sensitive data being gathered by external sources.

Trust boundary #: 3 - App Container Boundary
Untrusted party: Infrastructure provider threat. Container is managed by GCP, and therefore could be attacked by anyone with GCP credentials. Our system would also be affected in any mass GCP attack that affects the platform as a whole.

Security requirements. The project document defines many requirements. Identify the *security* requirements of your system, aligned with the six security properties defined by the STRIDE article. These requirements may vary by system, depending on which features you implemented. It is possible that you will not have a requirement associated with every security property.

Confidentiality
- [All systems]: Observers on the network cannot directly observe client-server interactions.

Integrity
- [All systems] Build must be tested before being deployed.

Authorization
- [All systems] Unauthorized users cannot tamper with source code.
- [All systems] Unauthorized users cannot deploy code.

Nonrepudiation
- [All systems] Changes to source code must be tracked and attributed to specific contributors.

Fill out this table for each STRIDE property (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege):

Stride property: Spoofing
Affected security properties: Authentication
Analysis of components:

- Diagram 1, Component 4 - GCP App Engine
  - Web server may be spoofed by an attacker and this may lead to information disclosure by the user.
    - Mitigations applied: No sensitive information is disclosed by users. This is not a threat to the security of the system.
    - Degree of risk resolution: High Severity, Low frequency. Adversaries would need to hijack DNS for our specific site. Could lead to possibly sensitive content going to the attacker.

- Diagram 1, Component 3 - User Sending HTTP Requests Directly using API
  - User spoofed by attacker.
    - Mitigations applied: We do not send any confidential data to users, thus spoofing a user has no advantage and does not pose a security threat.
    - Degree of risk resolution: Low Severity, Low frequency. Would require attacker to have control of users device, would not gain access to any significantly important data.

Stride property: Tampering
Affected security properties: Integrity
Analysis of components:

- Diagram 1, Component 5 - Google Cloud Platform Cloud Storage (Blob Storage)
  - XSS scripting attack: Attacker injects malicious code into our browser interface.
    - Mitigations applied: Keep secure passwords and ensure that attackers cannot push code to the repository. We only deploy code that is written by ourselves.
    - Degree of risk resolution: Low Severity, Low Frequency. GCP App Engine will not run code put in by the user. Files are only read in as raw bytes.
  - Log Readers: Log readers attacked using log files.
    - Mitigations applied: Users have no ability to run files. Data is passed as a zip file in the form of bytes and rejected if an invalid zip file. No ingested files are run by our program.
    - Degree of risk resolution: Low Severity, Low Frequency. GCP App Engine will not run code put in by the user. Files are only read in as raw bytes.
    - Suggested Mitigation: Ingest any external inputs in a sandboxed environment.
- Diagram 1, Component 4 - Google Cloud Platform App Engine
  - Faulty validation for web server: Not all possible inputs considered.
    - Mitigations applied: Detailed criteria for ingestion. We limit our attack surface by limiting the data flowing between modules.

        - Degree of risk resolution: High Severity, High Frequency. Not all possible use cases have been tested. GCP App Engine, upon encountering any errors, will return to the requestor either an error defined in our code or a default error that has been pre-determined by App Engine.
        - Suggested Mitigation: Fuzz the API with Restler.
    - JavaScript Object Notation Processing: JSON processing and hijacking threats may be exploited.
        - Mitigations applied: We have strict JSON parsing requirements and extract only a small slice of the data in the JSON file to decrease the ability to exploit JSON vulnerabilities. If this element is not present, the request is rejected.
        - Degree of risk resolution: High Severity, High Frequency. Possibility for JSON to be hijacked with the contents of the package. A hijacker could plant malicious code into the uploading or downloading package, without the user being aware.
        - Suggested Mitigation: Fuzz the API with Restler to find additional unconventional vulnerabilities.

Stride property: Repudiation
Affected security properties: Non-repudiation
Analysis of components:

- Diagram 1, Component 4 - Google Cloud Platform App Engine
    - Data logs from unknown source: Unknown users able to to change log file.
        - Mitigations applied: Unknown users are unable to access the log file, log file access is limited to collaborators on Google Cloud Platform. Any executable, however, can log messages to the log file.
        - Degree of risk resolution: Low Severity, Low Frequency. Logging is carried out entirely by GCP App Engine.
    - Repudiation by web server: Did not receive data from source outside trust boundary.
        - Mitigation: Detailed logging of the event and error messages.
        - Degree of risk resolution: Low Severity, Low Frequency. Logging implemented by GCP App Engine will be capable of logging any data coming in our out of system.
        - Suggested mitigation: Implement handshake protocol between sources to confirm success and retry if failure.
- Diagram 1, Component 3 - User Sending HTTP Requests Directly using API
    - Repudiation for attack on audit mechanism: Person denies upload of (possibly malicious) package.
        - Mitigation: There are no current features in place to avoid this from happening.
        - Degree of risk resolution: High Severity, Low Frequency. Since traceability is not thoroughly supported by our system, an attacker could

upload malicious content without being traced outside of standard GCP App Engine Logging.
        - Suggested mitigation: Tracing on package ingestion.

Stride property: Information Disclosure
Affected security properties:Confidentiality
Analysis of components:

- Diagram 1, Component 5 - Google Cloud Platform Cloud Storage (Blob Storage)
    - Weak access control: Weak data protection in GCP cloud storage.
        - Mitigations applied: We do not have any security features, but we also do not store any sort of sensitive information. Anything in the bucket is theoretically free for anyone to access.
        - Degree of risk resolution: High Severity, High Frequency. Anyone with access to our site can access and upload packages. Anything without a package.json and that is not a .zip will not be accepted, but requests with those will.
        - Suggestions for additional mitigations: Implement authentication to ensure only those with the correct permissions may access information stored by the system.
- Diagram 1, Component 2- Browser Sending HTTP Request to App Engine
    - Data Flow Sniffing: Data flowing across Requests may be sniffed by an attacker.
        - Mitigations applied: We do not store any sort of sensitive information in requests. Anything in the bucket is theoretically free for anyone to access.
        - Degree of risk resolution: Low Severity, High Frequency. Anyone could potentially view the requests and responses if they are intercepted. No sensitive data is included in these requests or responses.
        - Suggestions for additional mitigations: Once authentication is implemented, ensure credentials are sent and received using only the most recent recommended practices.

Stride property: Denial of Service
Affected security properties: Availability
Analysis of components:

- Diagram 1, Component 4 - Google Cloud Platform App Engine
    - Excessive Resource Consumption: Use an excessive amount of computational power or storage.
        - Mitigations applied: Google App Engine caps runtime of process at 60 seconds. We have an automatically scaling bucket that avoids crashing when out of memory. Furthermore, there is a maximum size of 32 MB per http request. In addition, when getting a list of packages, only a maximum of 10 are returned at one time in a paginated format.

        - Degree of risk resolution: Low Severity, Low Frequency. App Engine will automatically scale to handle heavy loads, and will automatically reject requests when it is overloaded.
        - Suggestions for additional mitigations: Limit the amount of data that can come from any one user during a set time period. Flag any peculiar behavior for manual review.
    - Potential Process Crash or Stop for Web Server: Access to Google Cloud Platform is temporarily unavailable.
        - Mitigations applied: There is no sensitive local data being stored on the device. There is no requirement for a web server for authentication.
        - Degree of risk resolution: High Severity, Low Frequency. In the event that App Engine goes down, the tool will be unavailable for use.
        - Suggestions for additional mitigations: ACME should hold local backups in a secure location.

- Diagram 1, Component 3 - User Sending HTTP Requests Directly using API
    - Data Flow Request interrupted: An attacker interrupts data flowing between the user and the browser.
        - Mitigations applied: Users must take steps to protect their computers from viruses that might interrupt service.
        - Degree of risk resolution: Possibility for JSON to be hijacked with the contents of the package. A hijacker could plant malicious code into the uploading or downloading package, without the user being aware.
        - Suggestions for additional mitigations: Use end-to-end encryption to protect sensitive data.

Stride property: Elevation of privilege
Affected security properties: Authorization
Analysis of components:

- Diagram 1, Component 1 - User Accessing Front End Through a Browser
    - Elevation Using Impersonation: Impersonate a user with higher permissions and exploit it. Since we do not have permission tiers in our service, there are no users with higher permission levels to impersonate.
        - Mitigations applied: Keep Github tokens private.
        - Degree of risk resolution: High Severity, Low Frequency. Having adversaries have access to an employee's computer can be a very severe breach that would allow the attacker to access all files.
        - Suggestions for additional mitigations: Implement authorization and an authorized users list.

- Diagram 1, Component 4 - Google Cloud Platform App Engine
    - Elevation by Changing the Execution Flow in Web Server: Change the flow of program execution within API or trustworthiness module to serve attackers. Ingest packages that are not trustworthy, change functionality of API requests.

- Mitigations applied: Google Cloud or Github token credentials need to be compromised in order for this to happen. Users need to set secure Github and Google Cloud Platform passwords.
- Degree of risk resolution: High Severity, Low Frequency. This may come from a bad actor within the client. Traffic could be redirected covertly by someone who has permissions to push code.
- Suggestions for additional mitigations: Increase the standards for code review before deploying.

Deployment: GCP

Complete the following table. It should include all GCP components that you used.

Purpose	**Selected GCP component(s)**	**Other GCP components considered**	**Justification for selected component**
Compute/Auto-Scaling	Google App Engine	Google Compute Engine	● More scalable than Compute Engine. Automatically manages resources ● Serverless ● Containerized ● Well Documented
Storage	Google Cloud Storage	PostgreSQL mySQL firestore	● Blob storage ● Cheap ● Scalable ● Well Documented
Secure storage	Google Secrets Manager	none	● Allows us to store our GitHub tokens securely ● Directly interfaces with other GCP tools

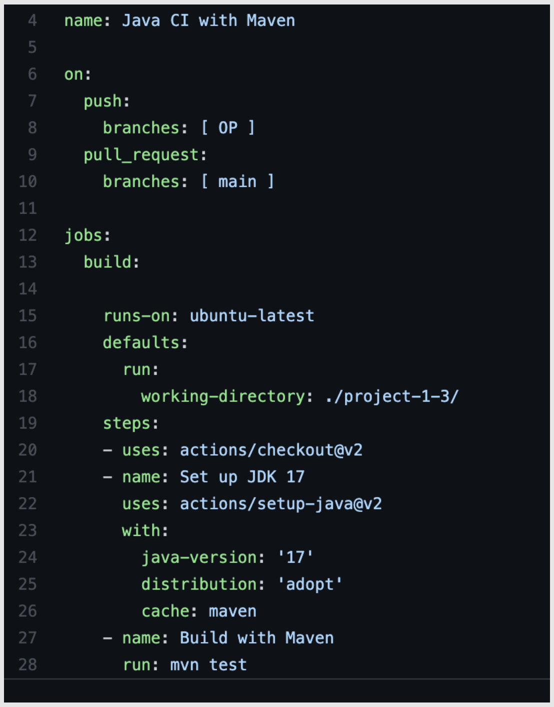
Deployment: CI/CD

CI/CD: Provide prose and screenshot(s) demonstrating that your team is using GitHub actions to facilitate continuous integration (e.g. by running a *linter* and a *test suite* on every pull request) and continuous deployment.

You may re-use answers from Deliverable #1 as appropriate.

What steps is your team following prior to accepting a code change? (e.g. git-hooks, code review, linting, test suite, etc.)

We have 3 GitHub actions, shown below, that run on pushes and pull requests, as well as in-person code reviews. In addition, any time we push a code change to the main branch after merging our code, the program is automatically deployed to GCP App Engine.

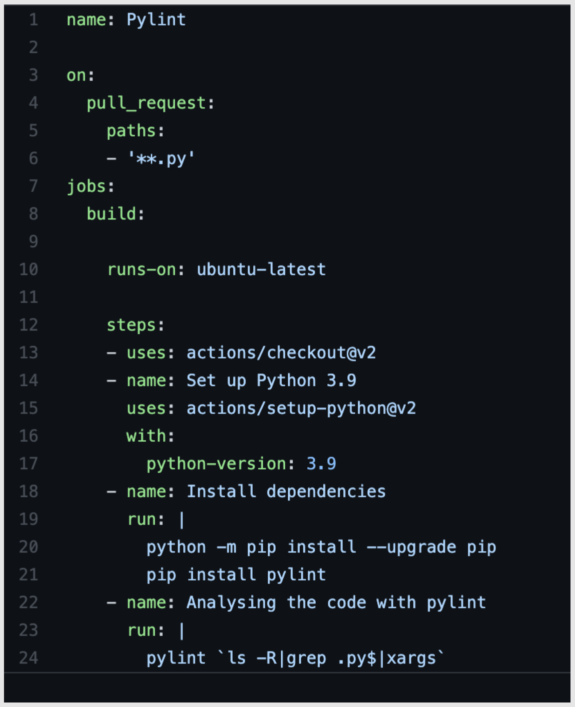


```yaml
name: Java CI with Maven

on:
  push:
    branches: [ OP ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./project-1-3/
    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 17
      uses: actions/setup-java@v2
      with:
        java-version: '17'
        distribution: 'adopt'
        cache: maven
    - name: Build with Maven
      run: mvn test
```



```yaml
name: Pylint

on:
  pull_request:
    paths:
    - '**.py'
jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.9
      uses: actions/setup-python@v2
      with:
        python-version: 3.9
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install pylint
    - name: Analysing the code with pylint
      run: |
        pylint `ls -R|grep .py$|xargs`
```

Provide a link to an example in your GitHub repo where your team followed this CI process (e.g. a pull request):
https://github.com/Purdue-ECE-461/project-2-project-2-4/pull/3
https://github.com/Purdue-ECE-461/project-2-project-2-4/actions/workflows/GAE_CD.yml

How consistent have you been with this process? What is keeping you from full consistency?

We spent more time doing this during the first few weeks of the project than the last three. We ran into some issues with the CD process. App Engine does not have the JDK installed by default, and we require it. It is too large to store in our repository without having issues, so we have to manually deploy this part of the system. Future work would include finding a platform that can support both Python and Java.
We also prioritized the functionality of the API over the CI/CD stuff during the last few weeks of the project. This meant that we did not flesh out some of the CI materials as much as we could have.

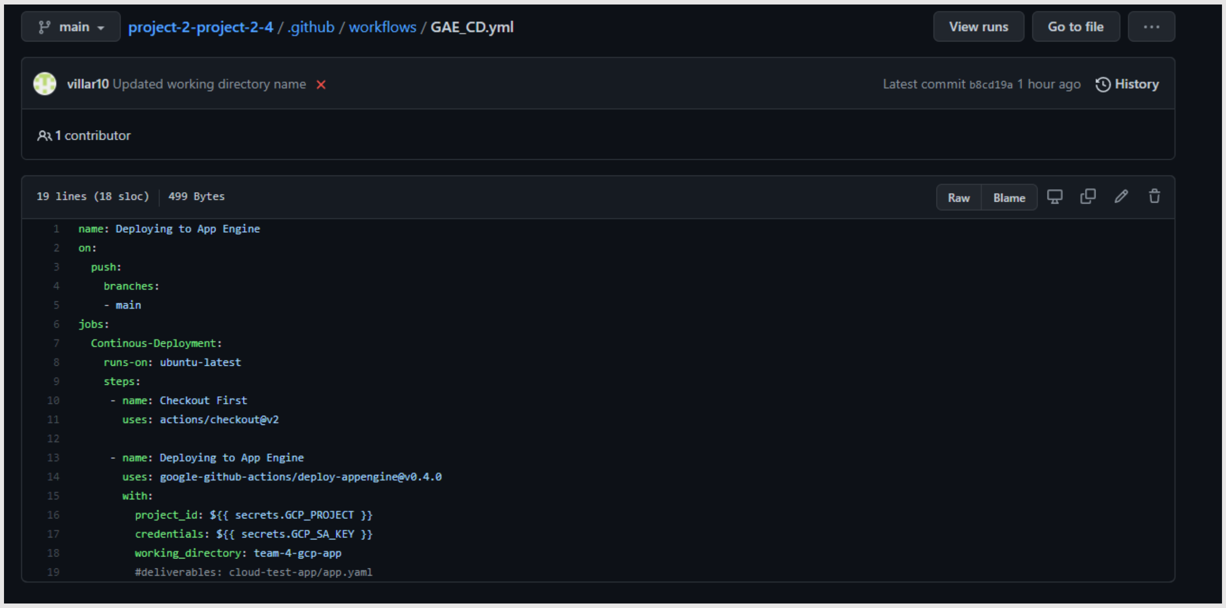
What aspects of your system are being tested automatically by your CI scheme?

We currently test Python code quality using PyLint and the correctness of our trustworthiness module using a JUnit test suite that is executed through a Maven Github action.

What kinds of defects might go uncaught, and how are you mitigating this risk?

We have a large testing suite set up in Postman, but this does not run automatically. This testing suite tests covers all of the API functionality and should detect a wide range of API issues. Issues with the base trustworthiness module (the one we were provided with) could go uncaught because we do not fully test this module.

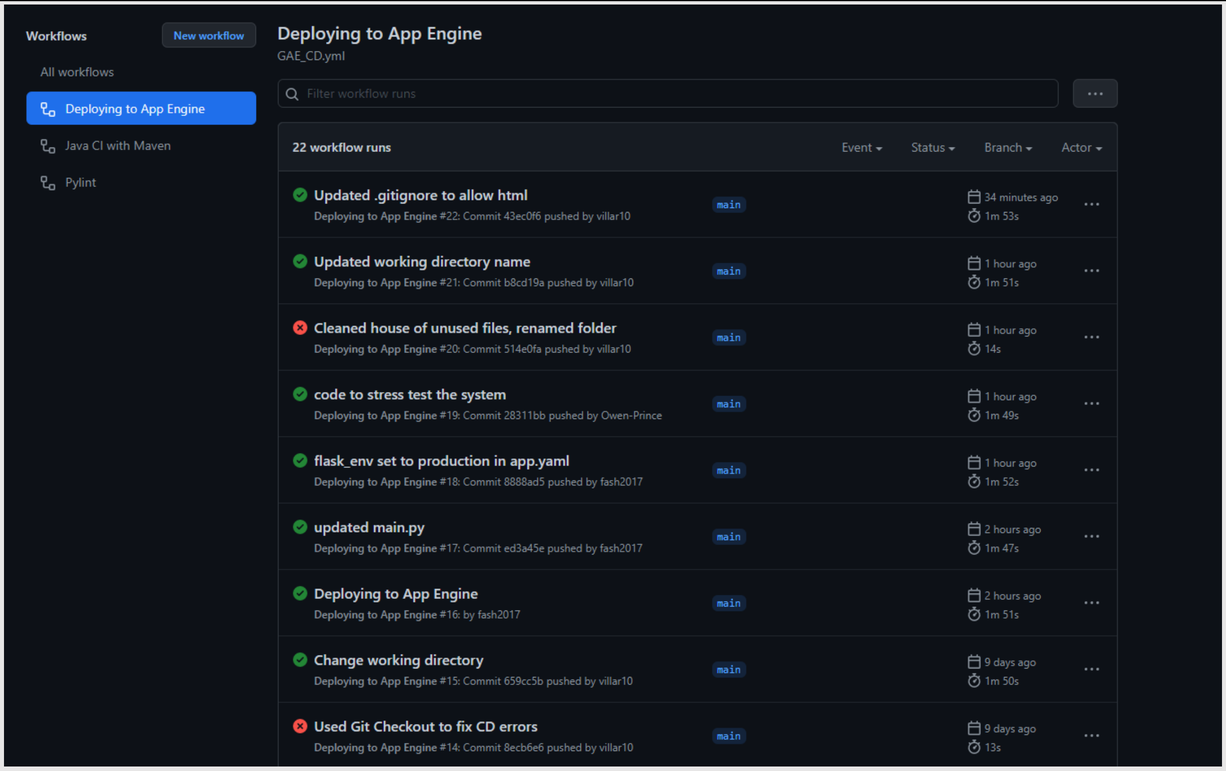
Provide screenshots of the GitHub action file (e.g. YAML) that defines the CI stages


main
project-2-project-2-4 / .github / workflows / GAE_CD.yml
View runs
Go to file
villar10 Updated working directory name
Latest commit b8cd19a 1 hour ago
History
1 contributor
19 lines (18 sloc) | 499 Bytes
Raw
Blame

```yaml
name: Deploying to App Engine
on:
  push:
    branches:
    - main
jobs:
  Continous-Deployment:
    runs-on: ubuntu-latest
    steps:
     - name: Checkout First
       uses: actions/checkout@v2

     - name: Deploying to App Engine
       uses: google-github-actions/deploy-appengine@v0.4.0
       with:
         project_id: ${{ secrets.GCP_PROJECT }}
         credentials: ${{ secrets.GCP_SA_KEY }}
         working_directory: team-4-gcp-app
         #deliverables: cloud-test-app/app.yaml
```

Provide screenshot(s) of the test suite in action, e.g. the reports from the various tools you have configured, as run on one of your team's code changes.



Deploying to App Engine
GAE_CD.yml

22 workflow runs

✓	Updated .gitignore to allow html Deploying to App Engine #22: Commit 43ec0f6 pushed by villar10	main	34 minutes ago 1m 53s
✗	…		

Describe the extent to which you are able to "continuously deploy". What is your team's process to get your current prototype into a deployment on GCP?

When we push to main, the new code automatically deploys. Due to the codebase being split between Java and Python, we need to add the JDK folder into our deployed code, but since this is such a large file, we cannot do it automatically since GitHub won't allow us to store JDK with the rest of our files. As a result, we must push to main and re-deploy the JDK folder to App Engine.

Non-baseline

Performance

If you considered the performance requirements:

- Provide latency details for mean, median, and 99%ile clients for the "many clients download lodash" scenario described in the project spec (or as close as you could get to that scenario).

We used the repository "jonschlinkert/even" instead of lodash for testing performance. This was done because lodash's package.json file does not contain the necessary information for our system to perform a rate on. A link to the GitHub repository is required in the package.json for our system. 50,000 "jonschlinkert/even" were uploaded using the "is_even_spam.py" script in our repository. 10,000 requests were sent simultaneously using multithreading on Google Colab, as shown in the "generate_stress_test.ipynb" in our repository. From this test, response times were recorded and then analyzed in Microsoft Excel to get the following metrics:

Mean response time: 4.145s
Median response time: 2.281s
99%tile response time: 12.159s

- Describe any design choices you made specific to performance (e.g. component selection; optimized paths such as caches; etc.)

GCP App Engine is an automatically scaling system. This means that during a mass influx of users all attempting to interact with the same data, App Engine will use as many resources as is necessary to fulfill all requests. As shown in our response times, even during an extremely high load spike the system would react in a timely manner with the expected output.

Notes for the auto-grader

If your submission cannot be automatically parsed by the auto-grader described in the project specification, provide explanatory notes that the course staff can consider while scoring your submission. Be specific. Since this spec was provided well in advance, accommodating any deviations is at the discretion of the staff.

Deviation	Details
package.json does not exist	If the package.json file does not exist, the package will throw an error and fail to upload.
No github link in the package.json	If there is no github URL field in the package.json file, the package will throw an error and fail to upload.
PUT request for update	returns "" instead of nothing

References

Bureau of Internet Accessibility. (2017, May 20). *Best Fonts To Use for Website Accessibility*. Bureau of Internet Accessibility. Retrieved December 8, 2021, from https://www.boia.org/blog/best-fonts-to-use-for-website-accessibility