Team-level slides

Group 2

An Automated Web Vulnerability Scanner

Problem: Web Application vulnerabilities are the most common form of security vulnerability, composing 25% of all breaches. They are difficult to detect as they require specialized training to understand and most web applications have wide surface areas.

Solution: Create an automated web vulnerability scanner that will catch simple vulnerabilities to reduce the attack surface area of web applications.

Key Features: The application, at its simplest level, would function through being supplied a domain/url to a website. Additionally, subdomains and other endpoints may be specified. Our scanner would then execute a number of simple enumeration procedures commonly seen in pentesting (e.g. directory busting) and display the data to the user (e.g. Commonly named directories). The user can also supply additional information to be able to run more specialized scans (e.g. specify where user data may be reflected in the client to perform a Cross Site Scripting scan).

Why This Project: Web application vulnerabilities, particularly client side ones, often affect not only the web application's provider but also the users. Many smaller web application providers also do not have the resources to contract audits of their applications. Because of this, the potential damage from and attack surface for web application vulnerabilities is extremely high. An automated scanner would hopefully be able to remediate at least the vulnerabilities that automated malicious vulnerability scanners check for.

Tagged Repository: https://github.com/MrLarryMan/Tensile/tree/milestone-5-submission

Milestone/Issues: https://github.com/MrLarryMan/Tensile/milestone/2

The Builders

Anay Gandhi - *Scrum Master & Project Manager*

- UI Issue: Job Specification Page

Hoa La - *QA Lead & Mediator*

- UI Issue: Login page

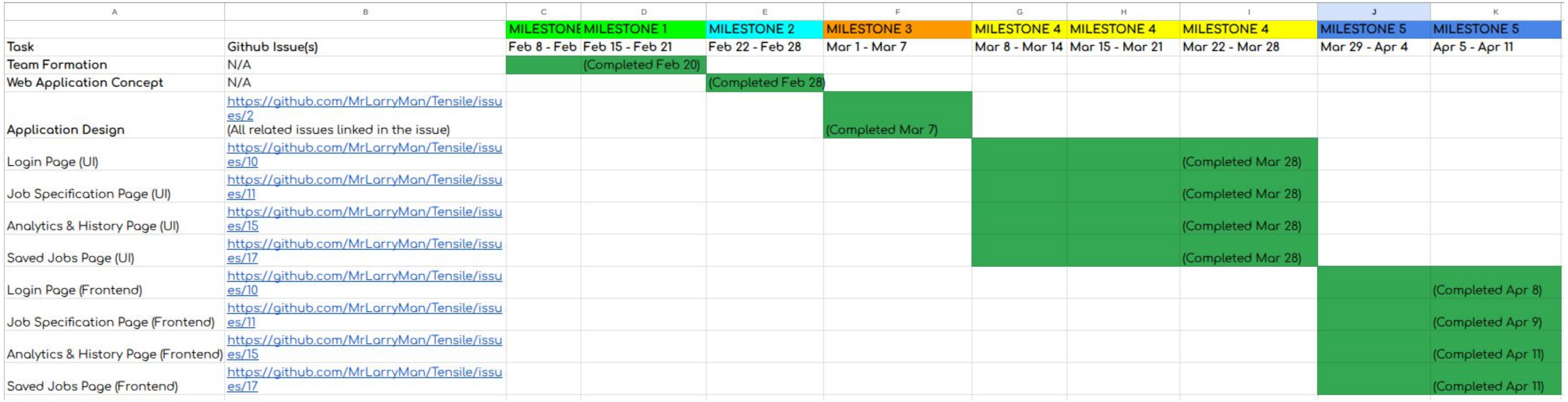
Larry Liu - *Cybersecurity Architect & Timekeeper*

- UI Issue: Saved Job Page

Mason Lemberger - *Systems Architect & Notetaker*

- UI Issue: Analytics and History Page

Historical Timeline



A	B	C	D	E	F	G	H	I	J	K
		MILESTONE	MILESTONE 1	MILESTONE 2	MILESTONE 3	MILESTONE 4	MILESTONE 4	MILESTONE 4	MILESTONE 5	MILESTONE 5
Task	Github Issue(s)	Feb 8 - Feb	Feb 15 - Feb 21	Feb 22 - Feb 28	Mar 1 - Mar 7	Mar 8 - Mar 14	Mar 15 - Mar 21	Mar 22 - Mar 28	Mar 29 - Apr 4	Apr 5 - Apr 11
Team Formation	N/A	🟩	(Completed Feb 20)							
Web Application Concept	N/A			(Completed Feb 28)						
Application Design	https://github.com/MrLarryMan/Tensile/issues/2 (All related issues linked in the issue)				(Completed Mar 7)					
Login Page (UI)	https://github.com/MrLarryMan/Tensile/issues/10					🟩	🟩	(Completed Mar 28)		
Job Specification Page (UI)	https://github.com/MrLarryMan/Tensile/issues/11					🟩	🟩	(Completed Mar 28)		
Analytics & History Page (UI)	https://github.com/MrLarryMan/Tensile/issues/15					🟩	🟩	(Completed Mar 28)		
Saved Jobs Page (UI)	https://github.com/MrLarryMan/Tensile/issues/17					🟩	🟩	(Completed Mar 28)		
Login Page (Frontend)	https://github.com/MrLarryMan/Tensile/issues/10								🟩	(Completed Apr 8)
Job Specification Page (Frontend)	https://github.com/MrLarryMan/Tensile/issues/11								🟩	(Completed Apr 9)
Analytics & History Page (Frontend)	https://github.com/MrLarryMan/Tensile/issues/15								🟩	(Completed Apr 11)
Saved Jobs Page (Frontend)	https://github.com/MrLarryMan/Tensile/issues/17								🟩	(Completed Apr 11)

Quick Clarification

- Our application features a multi-view UI (D) in the form of a navigation bar at the top of each page. It was already implemented by the start of this milestone and as such is not mentioned in our individual sections, but we are in fulfillment of requirement D due to its existence.

Individual Team Member Slides

-Hoa La-

Assigned Work Summary

- Work done this milestone: Register fetch, login fetch, and username top right dynamic content change

+ PR (merged) link: https://github.com/MrLarryMan/Tensile/pull/28

- Status: this milestone just involves JS implementation of some advanced features, which I have completed. → Completed & closed all issues in this milestone
- Links to all issues you are assigned in the milestone that were not closed: none

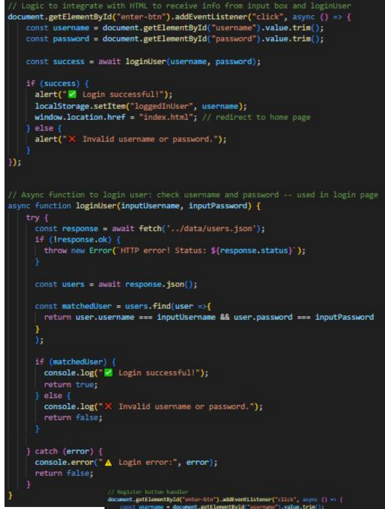
Feature Demonstration:

1. [F–2 points] Register page async fetch: check if username is taken and registerUser
   - Logic to receive and verify inputs (username, password).
   - Logic to check if username is taken
   - Logic to register user (username and password) by assuming a backend API to update database
2. [F–1 points] Login page async fetch: check if username exists and password matches
   - Logic to check if username exists in the users.json fake database and if the password matches
   - Logic to mark as logged in
3. [A–1 points] All pages display username on top right corner
   - Logic to display username on top right corner of every page, when the user is logged in
   - Used innerHTML in common script.js file

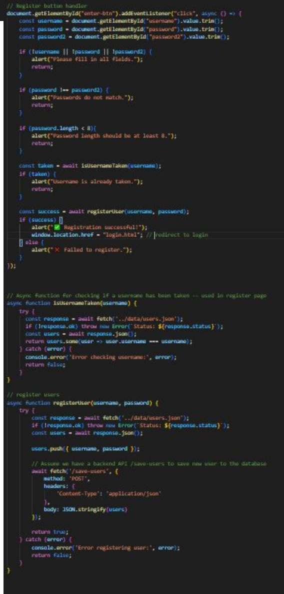
- Progress: all are completed: https://github.com/MrLarryMan/Tensile/tree/milestone5-hoa

Code Explanation

- My code contribution: functions to fetch and compare logic, functions to register user, logic to use localStorage to display username
- How the code integrates into the larger UI architecture: displaying username top right works across all pages
- Discussion of challenges faced and solutions implemented: using localStorage to store data and retrieve data to display in every page



```javascript
// Logic to integrate with HTML to receive info from input box and loginUser
document.getElementById("enter-btn").addEventListener("click", async () => {
    const username = document.getElementById("username").value.trim();
    const password = document.getElementById("password").value.trim();

    const success = await loginUser(username, password);

    if (success) {
      alert("✅ Login successful!");
      localStorage.setItem("loggedInUser", username);
      window.location.href = "index.html"; // redirect to home page
    } else {
      alert("❌ Invalid username or password.");
    }
});


// Async function to login user: check username and password -- used in login page
async function loginUser(inputUsername, inputPassword) {
    try {
      const response = await fetch('../data/users.json');
      if (!response.ok) {
        throw new Error(`HTTP error! Status: ${response.status}`);
      }

      const users = await response.json();

      const matchedUser = users.find(user =>{
        return user.username === inputUsername && user.password === inputPassword
      }
      );

      if (matchedUser) {
        console.log("✅ Login successful!");
        return true;
      } else {
        console.log("❌ Invalid username or password.");
        return false;
      }

    } catch (error) {
      console.error("⚠ Login error:", error);
      return false;
    }
}
```



```javascript
// Register button handler
document.getElementById("enter-btn").addEventListener("click", async () => {
    const username = document.getElementById("username").value.trim();
    const password = document.getElementById("password").value.trim();
    const password2 = document.getElementById("password2").value.trim();

    if (!username || !password || !password2) {
        alert("Please fill in all fields.");
        return;
    }

    if (password !== password2) {
        alert("Passwords do not match.");
        return;
    }

    if (password.length < 8){
        alert("Password length should be at least 8.");
        return;
    }

    const taken = await isUsernameTaken(username);
    if (taken) {
        alert("Username is already taken.");
        return;
    }

    const success = await registerUser(username, password);
    if (success) {
        alert("✅ Registration successful!");
        window.location.href = "login.html"; // redirect to login
    } else {
        alert("❌ Failed to register.");
    }
});


// Async function for checking if a username has been taken -- used in register page
async function isUsernameTaken(username) {
    try {
      const response = await fetch('../data/users.json');
      if (!response.ok) throw new Error(`Status: ${response.status}`);
      const users = await response.json();
      return users.some(user => user.username === username);
    } catch (error) {
      console.error('Error checking username:', error);
      return false;
    }
}

// register users
async function registerUser(username, password) {
    try {
        const response = await fetch('../data/users.json');
        if (!response.ok) throw new Error(`Status: ${response.status}`);
        const users = await response.json();

        users.push({ username, password });

        // Assume we have a backend API /save-users to save new user to the database
        await fetch('/save-users', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(users)
        });

        return true;
    } catch (error) {
        console.error('Error registering user:', error);
        return false;
    }
}
```

Challenges & Insights

- Obstacles faced and lessons acquired: learnt how to integrate a full user flow for register and login. Learnt to draw the user flow to design code product from scratch
- Key takeaways from working within a collaborative team environment: communicate well to avoid overlap and work towards a common goal for the team

Future Improvements & Next Steps

- Future improvements or features: backend integration in a real database
- Technical debt or aspects that could be optimized further: none for now.
- Issue link: https://github.com/MrLarryMan/Tensile/issues/29

Individual Team Member Slides

-Anay Gandhi-

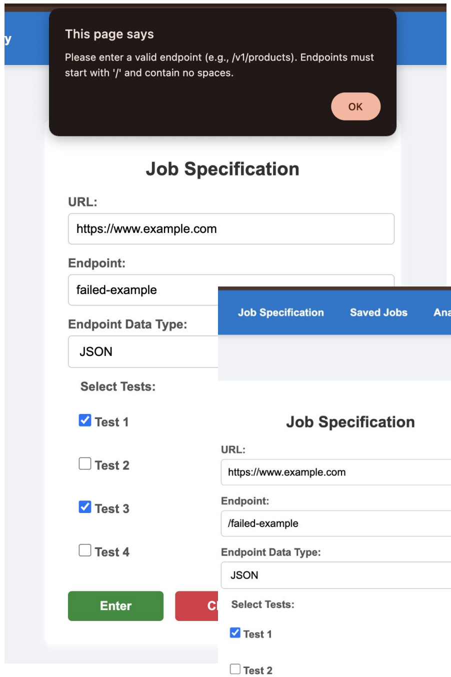
Assigned Work Summary

- Work done this milestone: Implemented data handling for job specification page: url input box, endpoint input box, endpoint data type dropdown, 4 test selection checkboxes, and 3 buttons – enter, clear, and save
- IndexedDB for saving jobs
+ Issue (closed) link: https://github.com/MrLarryMan/Tensile/issues/13
+ PR (merged) link: https://github.com/MrLarryMan/Tensile/pull/30

- Status: this milestone just touches the handling of data on the job spec page, which I have completed. → Completed & closed all issues in this milestone
- Links to all issues you are assigned in the milestone that were not closed: no remaining open issues for Milestone #4

Feature Demonstration

- Advanced feature - 3 pts
- Dynamic content updates (A), Form validation and feedback (B), Event handling (C), IndexedDB for persistence (E)
- Saves jobs to browser, validates user input, allows saving/clearing entries, retrieves saved data from IndexedDB
- ABCE all fully functional
- https://github.com/MrLarryMan/Tensile/tree/job-spec-page-js



This page says

Please enter a valid endpoint (e.g., /v1/products). Endpoints must start with '/' and contain no spaces.

Job Specification

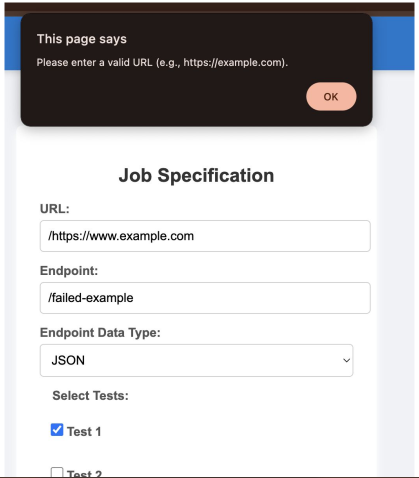
URL: https://www.example.com

Endpoint: failed-example

Endpoint Data Type: JSON

Select Tests:

☑ Test 1
☐ Test 2
☑ Test 3
☐ Test 4



This page says

Please enter a valid URL (e.g., https://example.com).

Job Specification

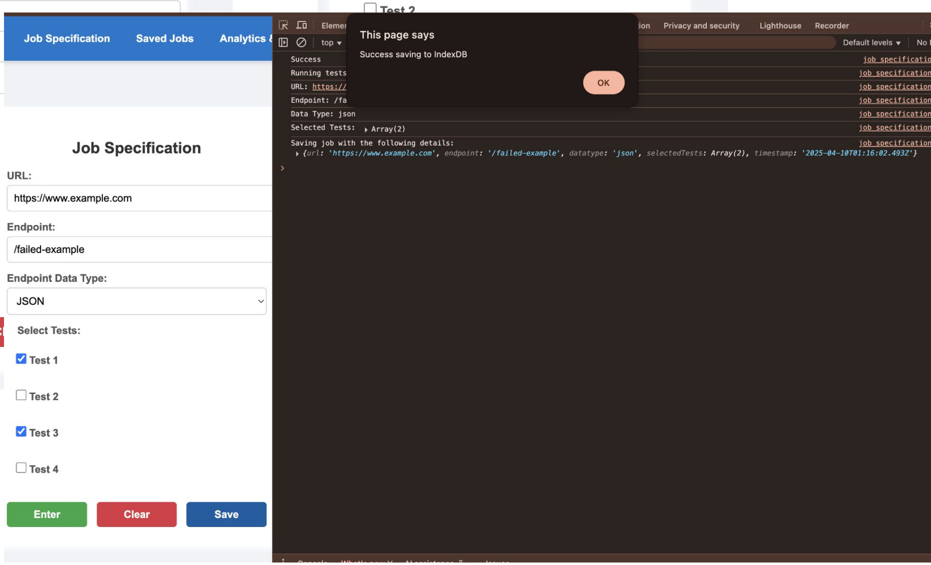
URL: /https://www.example.com

Endpoint: /failed-example

Endpoint Data Type: JSON

Select Tests:

☑ Test 1
☐ Test 2



This page says

Success saving to IndexDB

Job Specification

URL: https://www.example.com

Endpoint: /failed-example

Endpoint Data Type: JSON

Select Tests:

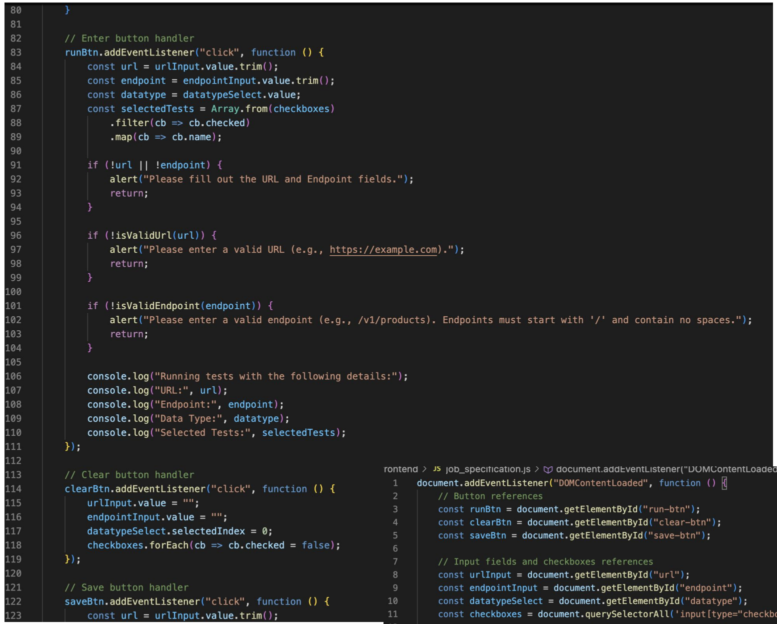
☑ Test 1
☐ Test 2
☑ Test 3
☐ Test 4

```
Success                                   job_specificatio
Running tests                             job_specification
URL: https://                             job_specification
Endpoint: /fa                             job_specification
Data Type: json                           job_specification
Selected Tests:  ▸ Array(2)               job_specification
Saving job with the following details:    job_specification
 ▸ {url: 'https://www.example.com', endpoint: '/failed-example', datatype: 'json', selectedTests: Array(2), timestamp: '2025-04-10T01:16:02.493Z'}
>
```

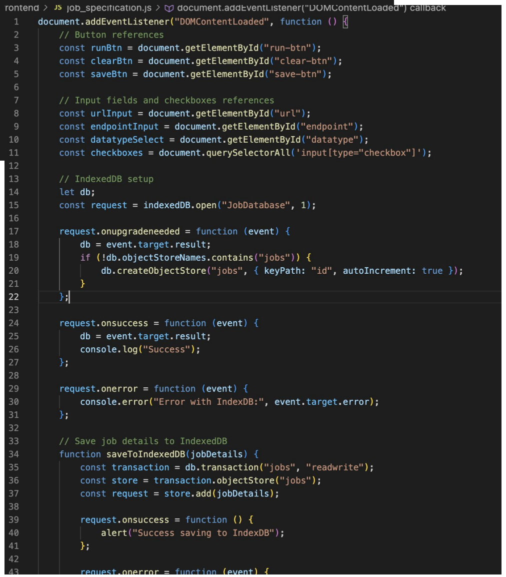
Code & Explanation

- Event listeners to handle button presses
- IndexedDB to set up user side persistence
- Clean separation between UI interaction, validations, and storage

- Challenge: initial setup of IndexedDB



```javascript
    }

    // Enter button handler
    runBtn.addEventListener("click", function () {
        const url = urlInput.value.trim();
        const endpoint = endpointInput.value.trim();
        const datatype = datatypeSelect.value;
        const selectedTests = Array.from(checkboxes)
            .filter(cb => cb.checked)
            .map(cb => cb.name);

        if (!url || !endpoint) {
            alert("Please fill out the URL and Endpoint fields.");
            return;
        }

        if (!isValidUrl(url)) {
            alert("Please enter a valid URL (e.g., https://example.com).");
            return;
        }

        if (!isValidEndpoint(endpoint)) {
            alert("Please enter a valid endpoint (e.g., /v1/products). Endpoints must start with '/' and contain no spaces.");
            return;
        }

        console.log("Running tests with the following details:");
        console.log("URL:", url);
        console.log("Endpoint:", endpoint);
        console.log("Data Type:", datatype);
        console.log("Selected Tests:", selectedTests);
    });

    // Clear button handler
    clearBtn.addEventListener("click", function () {
        urlInput.value = "";
        endpointInput.value = "";
        datatypeSelect.selectedIndex = 0;
        checkboxes.forEach(cb => cb.checked = false);
    });

    // Save button handler
    saveBtn.addEventListener("click", function () {
        const url = urlInput.value.trim();
```



rontend > JS job_specification.js > document.addEventListener("DOMContentLoaded") callback

```javascript
document.addEventListener("DOMContentLoaded", function () {
    // Button references
    const runBtn = document.getElementById("run-btn");
    const clearBtn = document.getElementById("clear-btn");
    const saveBtn = document.getElementById("save-btn");

    // Input fields and checkboxes references
    const urlInput = document.getElementById("url");
    const endpointInput = document.getElementById("endpoint");
    const datatypeSelect = document.getElementById("datatype");
    const checkboxes = document.querySelectorAll('input[type="checkbox"]');

    // IndexedDB setup
    let db;
    const request = indexedDB.open("JobDatabase", 1);

    request.onupgradeneeded = function (event) {
        db = event.target.result;
        if (!db.objectStoreNames.contains("jobs")) {
            db.createObjectStore("jobs", { keyPath: "id", autoIncrement: true });
        }
    };

    request.onsuccess = function (event) {
        db = event.target.result;
        console.log("Success");
    };

    request.onerror = function (event) {
        console.error("Error with IndexDB:", event.target.error);
    };

    // Save job details to IndexedDB
    function saveToIndexedDB(jobDetails) {
        const transaction = db.transaction("jobs", "readwrite");
        const store = transaction.objectStore("jobs");
        const request = store.add(jobDetails);

        request.onsuccess = function () {
            alert("Success saving to IndexDB");
        };

        request.onerror = function (event) {
```

Challenges & Insights

- Obstacles faced and lessons acquired
    - IndexedDB caused some confusion with timing, but resolved quickly
    - Making validation useful, without being over validating

- Key takeaways:
    - Comfortable integration of IndexedDB
    - Thinking of all possible scenarios the user could run into that would cause error messages that need to be displayed

Future Improvements & Next Steps

- Future improvements or features: deletion/edit of jobs, form auto-populate when selecting a past job, displaying past jobs

- Technical debt or aspects that could be optimized further: none for now.

- Issue link: https://github.com/MrLarryMan/Tensile/issues/22

Individual Team Member Slides

-Mason Lemberger-

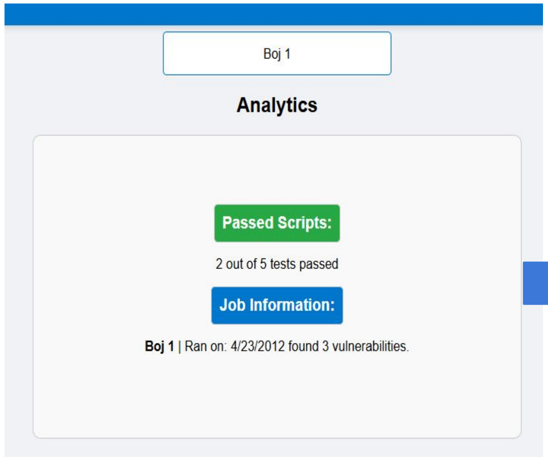
Assigned Work Summary

- **Work done this milestone:** Implemented js for history and analytics page:
  - Dynamically update multiple information boxes based on dropdown menus
  - Created a graph to understand historical data
  - Made fetch functions as well as mock server to simulate live data
+ Issue (closed) link: https://github.com/MrLarryMan/Tensile/issues/24
+ PR (merged) link: https://github.com/MrLarryMan/Tensile/pull/32

- Status: this milestone just touches the frontend js part, which I have completed. → Completed & closed all issues in this milestone
- Links to all issues you are assigned in the milestone that were not closed: none

Feature Demonstration

- Advanced feature - 3 pts
- Dynamic content updates (A), Event handling (C), Asynchronous Data Handling (F)
- Changes information based off selection in dropdown menus (no page reload), stores to sessionStorage to reduce fetch calls, has a graph for historical data that updates on reload(exact data showed subject to change), retrieves saved data from mock server or json file
- ACF all fully functional
- https://github.com/MrLarryMan/Tensile/tree/analytics-js


Boj 1

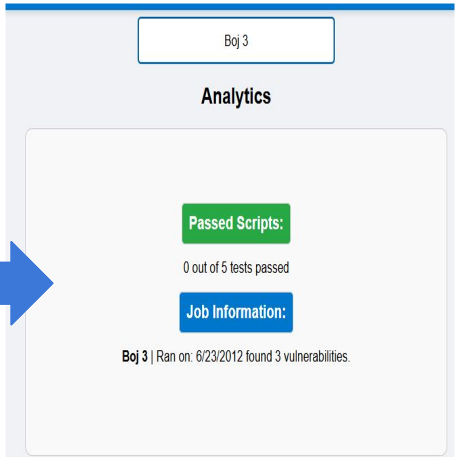
Analytics

Passed Scripts:

2 out of 5 tests passed

Job Information:

Boj 1 | Ran on: 4/23/2012 found 3 vulnerabilities.


Boj 3

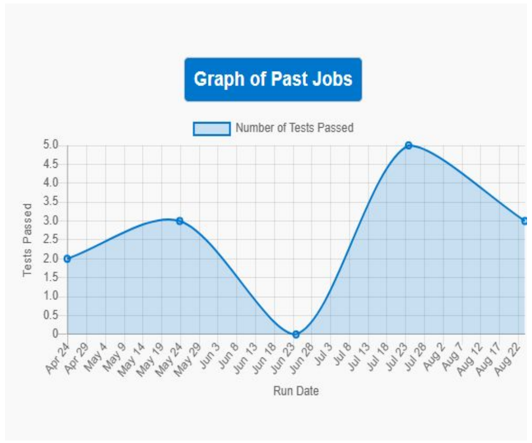
Analytics

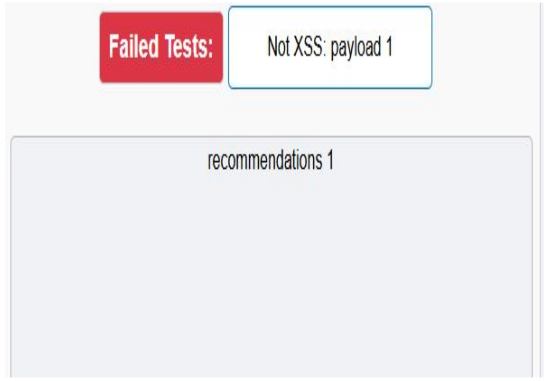
Passed Scripts:

0 out of 5 tests passed

Job Information:

Boj 3 | Ran on: 6/23/2012 found 3 vulnerabilities.

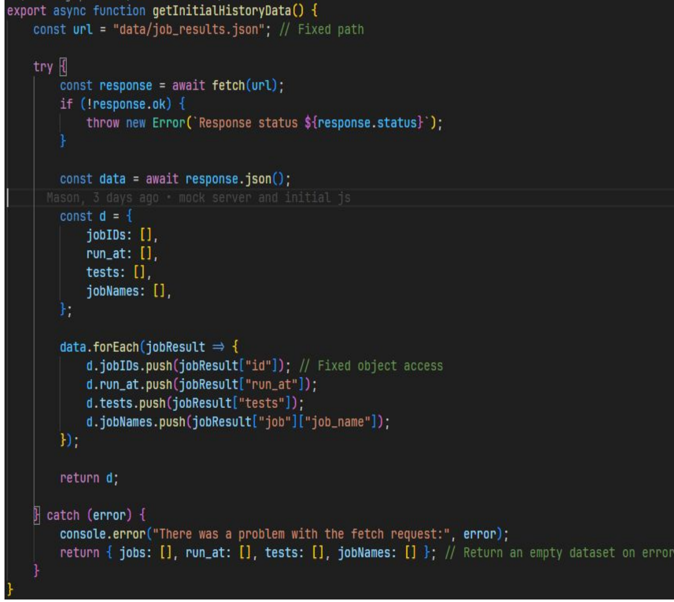



Failed Tests: Not XSS: payload 1

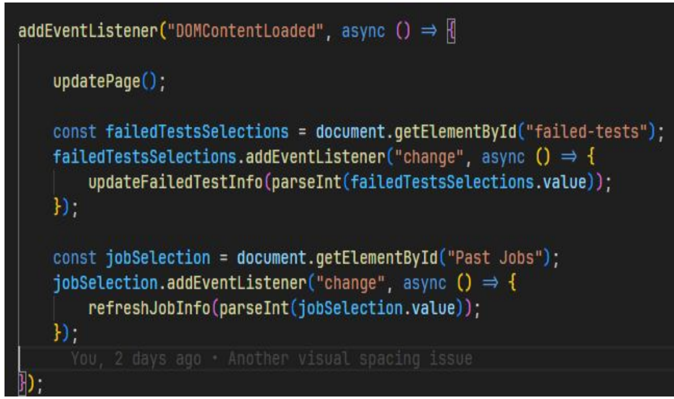
recommendations 1

Code Explanation

- **Dynamic Content Updates:** with the way I wrote my code, it will only fetch if the data is missing from the session storage. This reduces fetch requests but still allows for the data to be updated live if necessary.
- **Event Handling/User Interactions:** there are eventListeners tied to dropdown menus. When an option is selected it will set the relevant information boxes to the correct data using .innerHTML =.
- **Asynchronous Data Handling:** to access this data, I am using a local JSON file to simulate a server and using two fetch functions to access this file



```javascript
export async function getInitialHistoryData() {
    const url = "data/job_results.json"; // Fixed path

    try {
        const response = await fetch(url);
        if (!response.ok) {
            throw new Error(`Response status ${response.status}`);
        }

        const data = await response.json();
      Mason, 3 days ago • mock server and initial js
        const d = {
            jobIDs: [],
            run_at: [],
            tests: [],
            jobNames: [],
        };

        data.forEach(jobResult => {
            d.jobIDs.push(jobResult["id"]); // Fixed object access
            d.run_at.push(jobResult["run_at"]);
            d.tests.push(jobResult["tests"]);
            d.jobNames.push(jobResult["job"]["job_name"]);
        });

        return d;

    } catch (error) {
        console.error("There was a problem with the fetch request:", error);
        return { jobs: [], run_at: [], tests: [], jobNames: [] }; // Return an empty dataset on error
    }
}
```



```javascript
addEventListener("DOMContentLoaded", async () => {

    updatePage();

    const failedTestsSelections = document.getElementById("failed-tests");
    failedTestsSelections.addEventListener("change", async () => {
        updateFailedTestInfo(parseInt(failedTestsSelections.value));
    });

    const jobSelection = document.getElementById("Past Jobs");
    jobSelection.addEventListener("change", async () => {
        refreshJobInfo(parseInt(jobSelection.value));
    });
      You, 2 days ago • Another visual spacing issue
});
```

Challenges & Insights

- Obstacles faced and lessons acquired: One challenge I faced was figuring out when to use fetch and when to use sessionStorage to reduce server calls. This improved the way I thought about collecting data from a server and how the type of data used in a website might affect this.

- Key takeaways from working within a collaborative team environment: Communicating what works needs to be and who is to do it makes it easy to finish programming projects by dividing the work evenly

Future Improvements & Next Steps

- Future improvements or features: Making the data on the local JSON file update when the website is run instead of hardcoding in test data. Making sure that onces the backend code is implemented that the frontend code is still correct.

- Technical debt or aspects that could be optimized further: none for now.

- Future Issue link: https://github.com/MrLarryMan/Tensile/issues/31

Individual Team Member Slides

-Larry Liu-

Assigned Work Summary

- **Work done this milestone:** Implemented Javascript for the Saved Jobs UI page page
    - Asynchronously fetch from a Server (for now implemented as a static JSON file) in order to retrieve job details
    - Dynamically update a dropdown selection dialog showing the currently available saved jobs
    - Dynamically update a dialog with details for the currently selected jobs.
    - Added a multi-view UI element in the form of a confirmation modal dialogue for deletion.

+ Issue (closed) link: https://github.com/MrLarryMan/Tensile/issues/17
+ PR (merged) link: https://github.com/MrLarryMan/Tensile/pull/33

- Status: The milestone covers frontend features, which have been completed → Completed & closed all issues in this milestone
- Non-closed Issues: N/A

Feature Demonstration

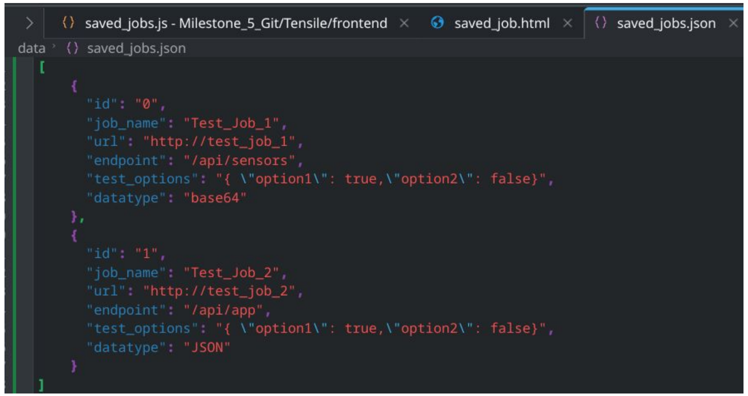
- A: One 3 Point Advanced feature
  - Dynamic content updates (A), Event handling (C), Asynchronous Data Handling (F)
- B: One 1 point Basic Feature
  - Multi-View UI (D), Event handling (C)

A: A dropdown selection to select a job to view & display its details. Makes use of Asynchronous Data Handling using a JSON file as a simulated server, Dynamic content updates to fill out the dialog using the previously fetched data by modifying the HTML using JS, and event handling in the selection element itself to trigger the dynamic content updates.

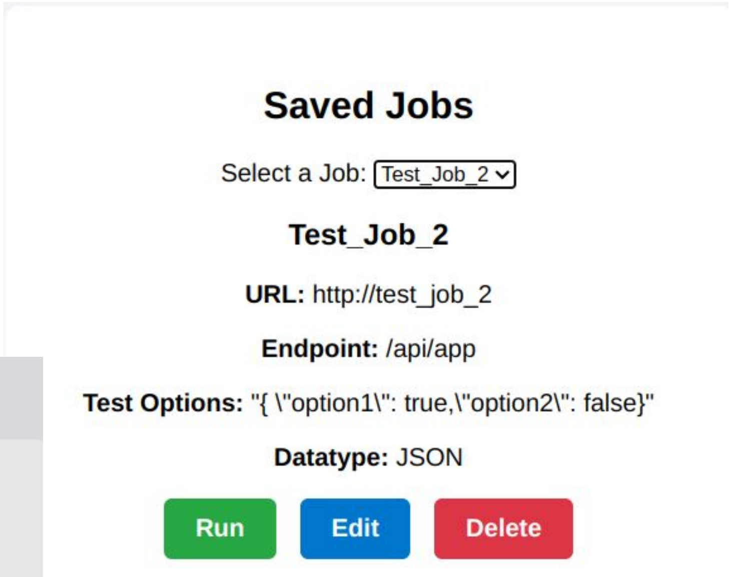
B: A confirmation pop up on the deletion button that shows a modal dialog when the delete button is pressed.

All fully functional when it comes to user visible information. Backend functions not implemented.

https://github.com/MrLarryMan/Tensile/tree/job_select_page_frontend_changes



```
[
    {
      "id": "0",
      "job_name": "Test_Job_1",
      "url": "http://test_job_1",
      "endpoint": "/api/sensors",
      "test_options": "{ \"option1\": true,\"option2\": false}",
      "datatype": "base64"
    },
    {
      "id": "1",
      "job_name": "Test_Job_2",
      "url": "http://test_job_2",
      "endpoint": "/api/app",
      "test_options": "{ \"option1\": true,\"option2\": false}",
      "datatype": "JSON"
    }
]
```



Saved Jobs

Select a Job: Test_Job_2

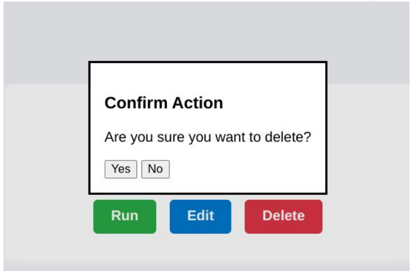
Test_Job_2

URL: http://test_job_2

Endpoint: /api/app

Test Options: "{ \"option1\": true,\"option2\": false}"

Datatype: JSON

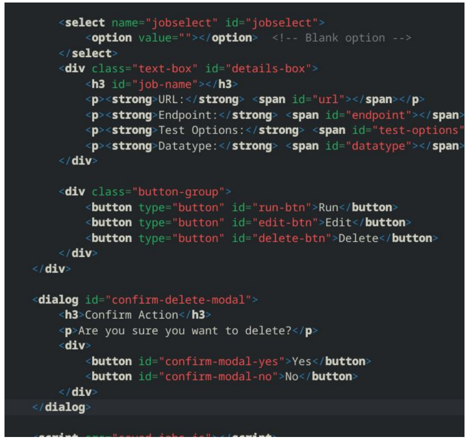


Confirm Action

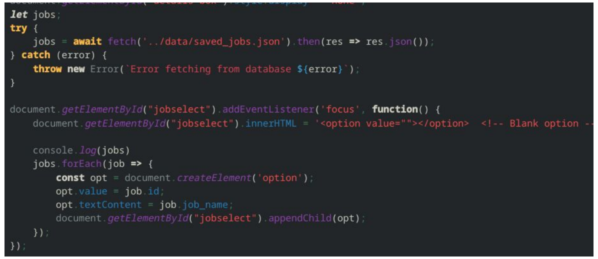
Are you sure you want to delete?

Code Explanation:

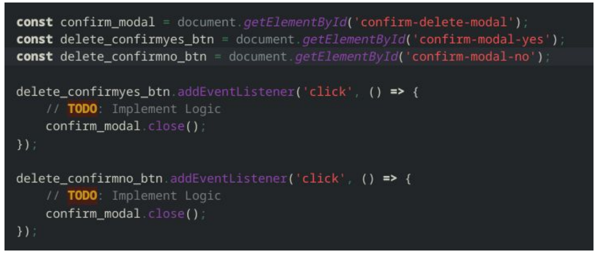
- My code contribution: Refined saved jobs menu to incorporate asynchronous data handling. Added code for a multi-view UI modal on the delete button.
- How the code integrates into the larger UI architecture: This is where users can select previously made jobs to run or edit.
- Discussion of challenges faced and solutions implemented: Setting up a testing environment that worked to bypass CORs security restrictions implemented by the browser. Reformatting the JSON data to be stored as a file instead of a JavaScript object as I was using previously.



```html
        <select name="jobselect" id="jobselect">
            <option value=""></option>  <!-- Blank option -->
        </select>
        <div class="text-box" id="details-box">
            <h3 id="job-name"></h3>
            <p><strong>URL:</strong> <span id="url"></span></p>
            <p><strong>Endpoint:</strong> <span id="endpoint"></span>
            <p><strong>Test Options:</strong> <span id="test-options"
            <p><strong>Datatype:</strong> <span id="datatype"></span>
        </div>

        <div class="button-group">
            <button type="button" id="run-btn">Run</button>
            <button type="button" id="edit-btn">Edit</button>
            <button type="button" id="delete-btn">Delete</button>
        </div>
    </div>

    <dialog id="confirm-delete-modal">
        <h3>Confirm Action</h3>
        <p>Are you sure you want to delete?</p>
        <div>
            <button id="confirm-modal-yes">Yes</button>
            <button id="confirm-modal-no">No</button>
        </div>
    </dialog>
```



```javascript
let jobs;
try {
    jobs = await fetch('../data/saved_jobs.json').then(res => res.json());
} catch (error) {
    throw new Error(`Error fetching from database ${error}`);
}

document.getElementById("jobselect").addEventListener('focus', function() {
    document.getElementById("jobselect").innerHTML = '<option value=""></option>  <!-- Blank option --

    console.log(jobs)
    jobs.forEach(job => {
        const opt = document.createElement('option');
        opt.value = job.id;
        opt.textContent = job.job_name;
        document.getElementById("jobselect").appendChild(opt);
    });
});
```



```javascript
const confirm_modal = document.getElementById('confirm-delete-modal');
const delete_confirmyes_btn = document.getElementById('confirm-modal-yes');
const delete_confirmno_btn = document.getElementById('confirm-modal-no');

delete_confirmyes_btn.addEventListener('click', () => {
    // TODO: Implement Logic
    confirm_modal.close();
});

delete_confirmno_btn.addEventListener('click', () => {
    // TODO: Implement Logic
    confirm_modal.close();
});
```

Challenges & Insights

- Obstacles faced and lessons acquired: One challenge I faced was setting up a testing environment that worked to bypass the CORs security restrictions implemented by my browser, as they prevented simple testing through rendering the HTML page directly on the browser without a server. Another challenge I faced was reformatting the JSON data to be stored as a file instead of a JavaScript object as I was using previously.
- Key takeaways from working within a collaborative team environment: Ensuring that our schedules lined up so that we could properly communicate and review each other’s work was essential. We the future we will coordinate times to check in virtually more efficiently.

Future Improvements & Next Steps

- Future improvements or features: Possibly implementing scrolling functionality so as to make the dropdown more usable in larger save job lists. Implementing backend functionality in a future milestone.

- Technical debt or aspects that could be optimized further: None for now outside the unimplemented features above. We may need to standardize the way we pull from our “servers” as well, although they are temporary features anyways and will be changed in the backend implementation.

- Issue link: https://github.com/MrLarryMan/Tensile/issues/26