Team-level slides

Group 2, 5/7/25

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-6-submission

Milestone/Issues: https://github.com/MrLarryMan/Tensile/milestone/4

The Builders

Anay Gandhi - *Scrum Master & Project Manager*

- UI Issue: Job Specification Page

Hoa La - *QA Lead & Mediator*

- UI Issue: Login page

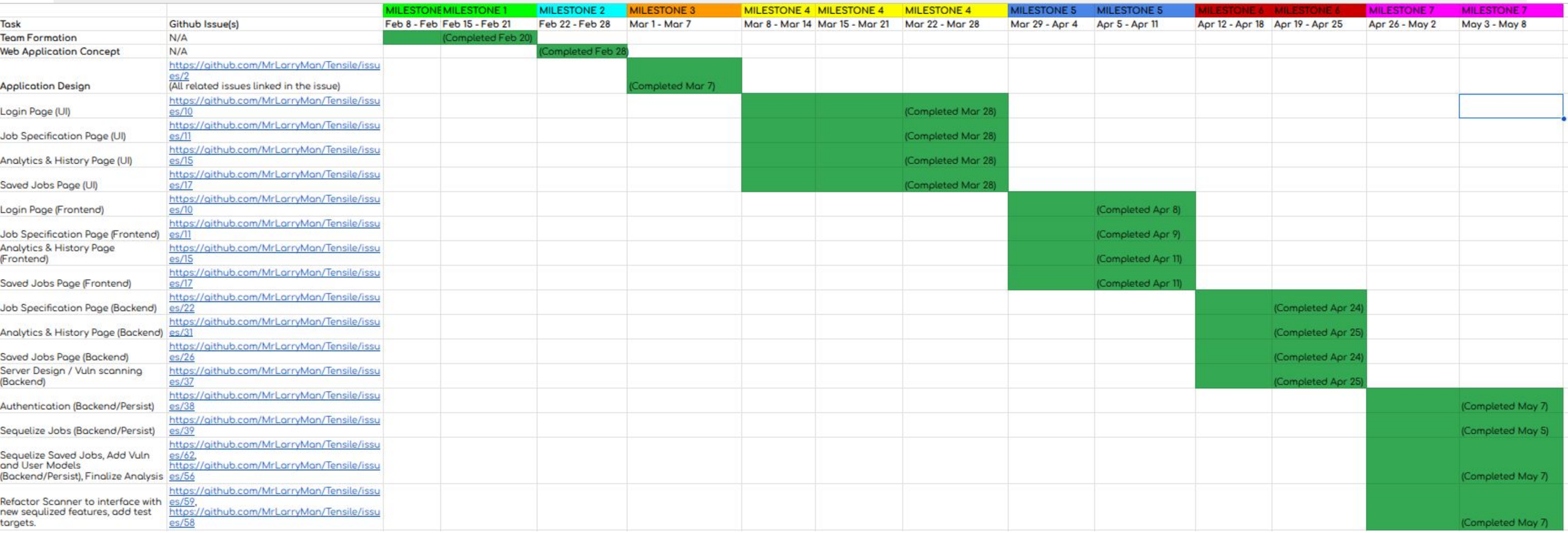
Larry Liu - *Cybersecurity Architect & Timekeeper*

- UI Issue: Saved Job Page

Mason Lemberger - *Systems Architect & Notetaker*

- UI Issue: Analytics and History Page

Historical Timeline



		MILESTONE	MILESTONE 1	MILESTONE 2	MILESTONE 3	MILESTONE 4	MILESTONE 4	MILESTONE 4	MILESTONE 5	MILESTONE 5	MILESTONE 6	MILESTONE 6	MILESTONE 7	MILESTONE 7
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	Apr 12 - Apr 18	Apr 19 - Apr 25	Apr 26 - May 2	May 3 - May 8
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)				
Job Specification Page (Backend)	https://github.com/MrLarryMan/Tensile/issues/22										🟩	(Completed Apr 24)		
Analytics & History Page (Backend)	https://github.com/MrLarryMan/Tensile/issues/31										🟩	(Completed Apr 25)		
Saved Jobs Page (Backend)	https://github.com/MrLarryMan/Tensile/issues/26										🟩	(Completed Apr 24)		
Server Design / Vuln scanning (Backend)	https://github.com/MrLarryMan/Tensile/issues/37										🟩	(Completed Apr 25)		
Authentication (Backend/Persist)	https://github.com/MrLarryMan/Tensile/issues/38												🟩	(Completed May 7)
Sequelize Jobs (Backend/Persist)	https://github.com/MrLarryMan/Tensile/issues/39												🟩	(Completed May 5)
Sequelize Saved Jobs, Add Vuln and User Models (Backend/Persist), Finalize Analysis	https://github.com/MrLarryMan/Tensile/issues/62, https://github.com/MrLarryMan/Tensile/issues/56												🟩	(Completed May 7)
Refactor Scanner to interface with new sequlized features, add test targets.	https://github.com/MrLarryMan/Tensile/issues/59, https://github.com/MrLarryMan/Tensile/issues/58												🟩	(Completed May 7)

Individual Team Member Slides

-Hoa La-

Assigned Work Summary

- Work done this milestone: complete authentication features for project with login and logout
- PR (merged) link: https://github.com/MrLarryMan/Tensile/pull/45

- Status: Completed & closed all issues in this milestone
- Links to all issues you are assigned in the milestone that were not closed: none

Feature Demonstration:

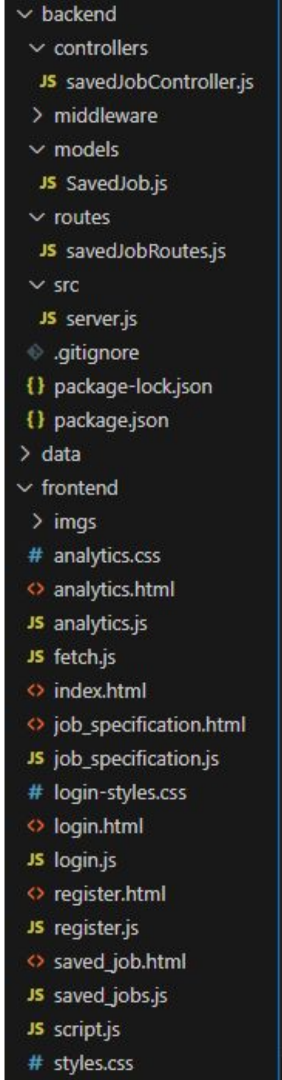
Features implemented:

1. Login: frontend + backend: when click login, make a request to backend for api/login, check logic there, and return to frontend result
2. Logout: frontend + backend. When click logout, make request to backend to process and manage authentication, then, return to frontend logged out.

- Progress: all are completed and integrated well with frontend js code: https://github.com/MrLarryMan/Tensile/tree/hoa-milestone7

Code Structure & Organization

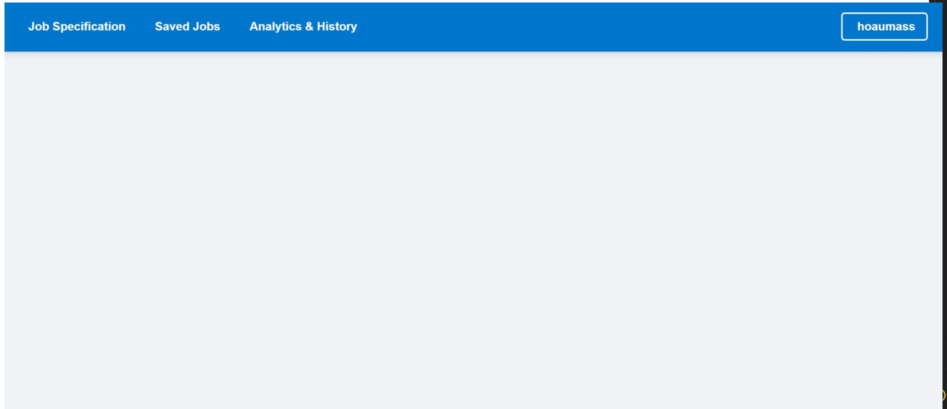
- Code structure: follows template structure of separating frontend and backend folders; in backend, split into models, routes, controllers, and middleware.
- Critical components: routes, models, controllers, and middleware components are organized separately in their folders.

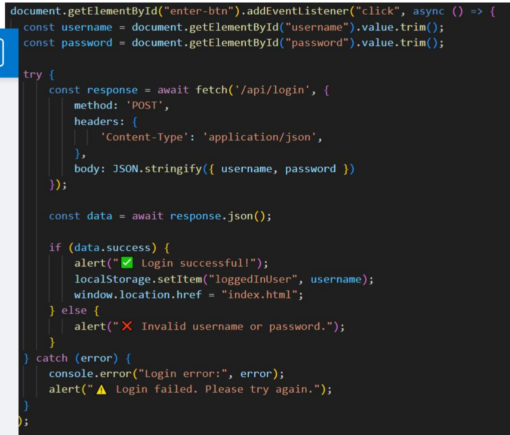


```
backend
  controllers
    savedJobController.js
  middleware
  models
    SavedJob.js
  routes
    savedJobRoutes.js
  src
    server.js
  .gitignore
  package-lock.json
  package.json
data
frontend
  imgs
  analytics.css
  analytics.html
  analytics.js
  fetch.js
  index.html
  job_specification.html
  job_specification.js
  login-styles.css
  login.html
  login.js
  register.html
  register.js
  saved_job.html
  saved_jobs.js
  script.js
  styles.css
```

Front-End Implementation

- Structure: components, styles, and utilities are split into different html, css, js files – clean and nice
- Backend integration: by fetching/ calling the right API endpoints exposed in the backend code
- Challenge: map to the right backend API → Solution: use console.log for debugging on both sides and iterate till success

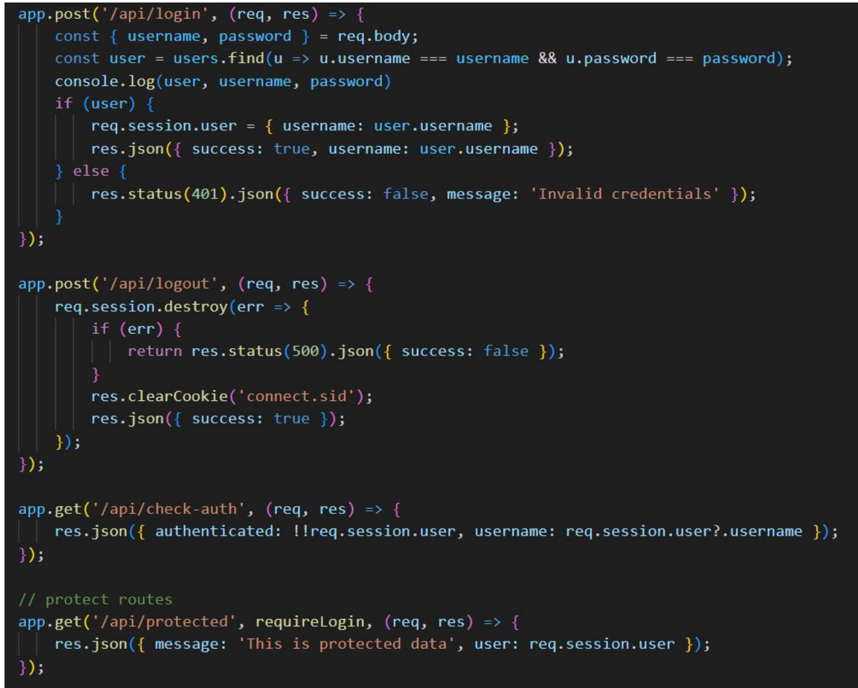




```
document.getElementById("enter-btn").addEventListener("click", async () => {
  const username = document.getElementById("username").value.trim();
  const password = document.getElementById("password").value.trim();

  try {
    const response = await fetch('/api/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ username, password })
    });

    const data = await response.json();

    if (data.success) {
      alert("✅ Login successful!");
      localStorage.setItem("loggedInUser", username);
      window.location.href = "index.html";
    } else {
      alert("❌ Invalid username or password.");
    }
  } catch (error) {
    console.error("Login error:", error);
    alert("⚠️ Login failed. Please try again.");
  }
});
```

Back-End Implementation

- Structure: models, routes, controllers, and middleware components are split into respective folders – ease for collaboration
- Frontend integration: by exposing clearly the right API endpoints, along with clear documentation how to use
- Challenge: deciding and parsing correct data type for request body → Solution: review Express.js request and body types to use the right data type. Make this clear via docs.



```
app.post('/api/login', (req, res) => {
    const { username, password } = req.body;
    const user = users.find(u => u.username === username && u.password === password);
    console.log(user, username, password)
    if (user) {
        req.session.user = { username: user.username };
        res.json({ success: true, username: user.username });
    } else {
        res.status(401).json({ success: false, message: 'Invalid credentials' });
    }
});

app.post('/api/logout', (req, res) => {
    req.session.destroy(err => {
        if (err) {
            return res.status(500).json({ success: false });
        }
        res.clearCookie('connect.sid');
        res.json({ success: true });
    });
});

app.get('/api/check-auth', (req, res) => {
    res.json({ authenticated: !!req.session.user, username: req.session.user?.username });
});

// protect routes
app.get('/api/protected', requireLogin, (req, res) => {
    res.json({ message: 'This is protected data', user: req.session.user });
});
```

Challenges & Insights

- Obstacles faced and lessons acquired: how to manage authentication sessions and integrate with current login-logout flow
- 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: allow Login with Google/Facebook/Microsoft…

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

- Issue link: https://github.com/MrLarryMan/Tensile/issues/46

Individual Team Member Slides

-Anay Gandhi-

Assigned Work Summary

+ Work done this milestone: Switched /api/jobs backend to use SQLite with Sequelize, Added Sequelize Job model and DB sync logic, Updated controllers/routes for Sequelize CRUD
+ PR (merged) link: https://github.com/MrLarryMan/Tensile/pull/44

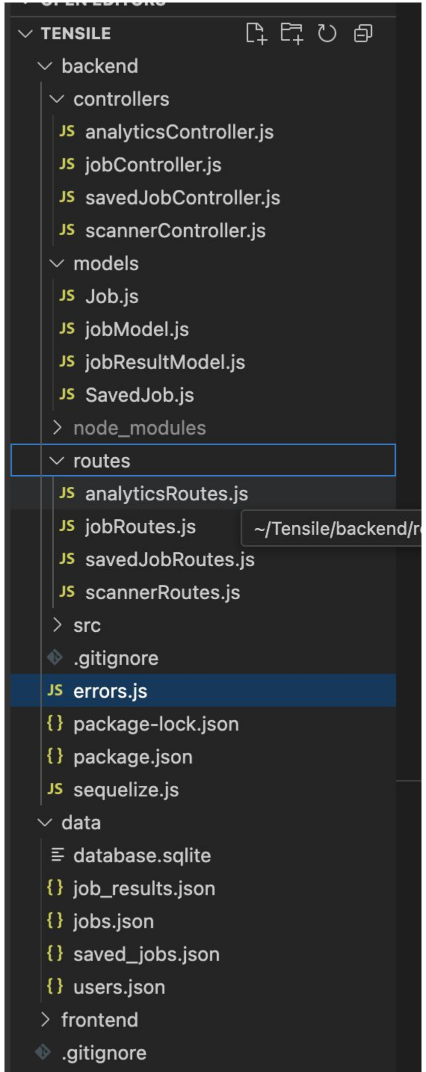
- Status: this milestone completed the SQLite and sequelize integration rather than static JSON data → Completed & closed all issues in this milestone
- Links to all issues you are assigned in the milestone that were not closed: no remaining open issues for Milestone #7

Feature Demonstration

- Migrated the /api/jobs endpoint to use SQLite as the new backend database, powered by Sequelize
- Defined a new Job model with schema validation
- All CRUD operations (GET, POST, PUT, DELETE) now use Sequelize methods
- https://github.com/MrLarryMan/Tensile/tree/anay-milestone7

Code Structure & Organization

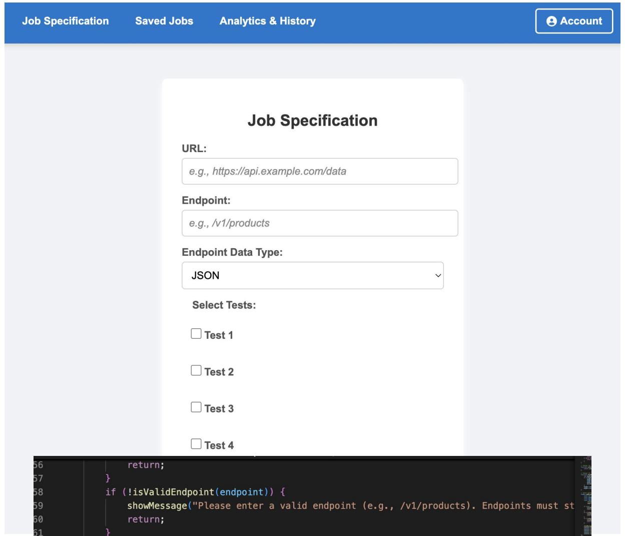
- Code structure: clear separation dividing frontend and backend. Backend is divided into dedicated folders for each of the different parts – models, routes, controllers, src
- Critical components: as previously mentioned, models, routes, controllers, src all have their independent attributes



```
TENSILE
  backend
    controllers
      analyticsController.js
      jobController.js
      savedJobController.js
      scannerController.js
    models
      Job.js
      jobModel.js
      jobResultModel.js
      SavedJob.js
    node_modules
    routes
      analyticsRoutes.js
      jobRoutes.js          ~/Tensile/backend/r
      savedJobRoutes.js
      scannerRoutes.js
    src
    .gitignore
    errors.js
    package-lock.json
    package.json
    sequelize.js
  data
    database.sqlite
    job_results.json
    jobs.json
    saved_jobs.json
    users.json
  frontend
  .gitignore
```

Front-End

- All UI logic for job spec page is within dedicated JS file, with clear separation from HTML/CSS
- Interacts with backend by sending fetch requests to appropriate /api/jobs endpoints for saving/entering job data, ensuring seamless data flow
- Challenge: ensuring frontend matched backend data expectations, and so that the API calls reached the intended endpoints
    - Log statements and console debugging to manually track it



Job Specification

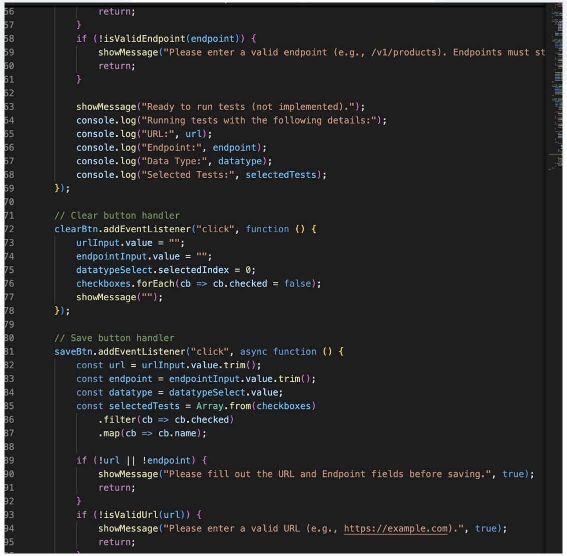
URL: e.g., https://api.example.com/data

Endpoint: e.g., /v1/products

Endpoint Data Type: JSON

Select Tests:

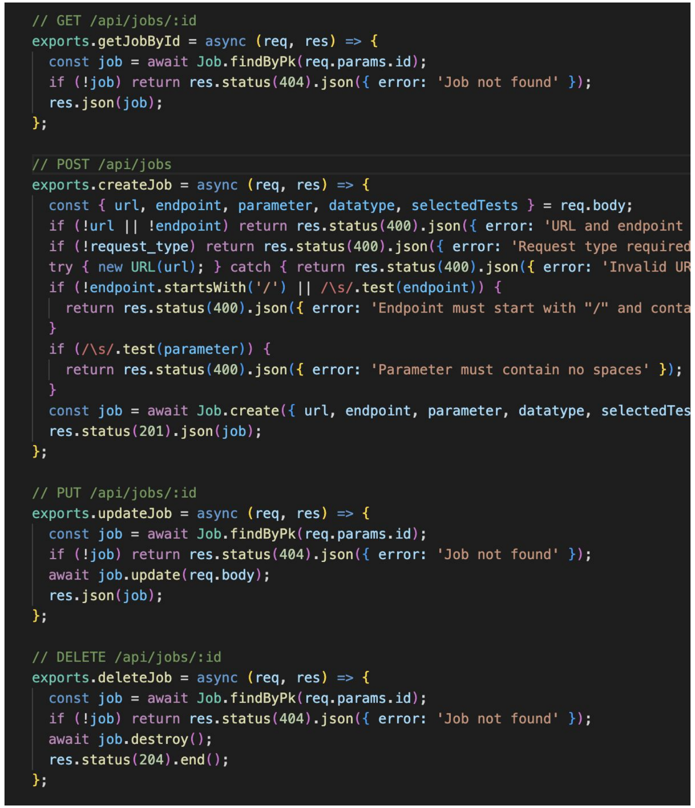
- [ ] Test 1
- [ ] Test 2
- [ ] Test 3
- [ ] Test 4



```
            return;
        }
        if (!isValidEndpoint(endpoint)) {
            showMessage("Please enter a valid endpoint (e.g., /v1/products). Endpoints must st
            return;
        }

        showMessage("Ready to run tests (not implemented).");
        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);
        showMessage("");
    });

    // Save button handler
    saveBtn.addEventListener("click", async 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) {
            showMessage("Please fill out the URL and Endpoint fields before saving.", true);
            return;
        }
        if (!isValidUrl(url)) {
            showMessage("Please enter a valid URL (e.g., https://example.com).", true);
            return;
```

Back-End

- Structure: Sequelize models in models/, logic in controllers/, route definitions in routes/, and integration-ready endpoints under src/
- Frontend integration: the /api/jobs endpoints cover all job operations (create, read, update, delete) with consistent response formats, enabling smooth integration with the frontend
- Challenge: validation and user feedback for each of the job spec fields on the front end, especially for back end error handling
    - Cleary validation logic and user messages
- Takeaway:clear validation logic and precise error messages to improve user guidance and maintain a strong contract between frontend and backend



```
// GET /api/jobs/:id
exports.getJobById = async (req, res) => {
  const job = await Job.findByPk(req.params.id);
  if (!job) return res.status(404).json({ error: 'Job not found' });
  res.json(job);
};

// POST /api/jobs
exports.createJob = async (req, res) => {
  const { url, endpoint, parameter, datatype, selectedTests } = req.body;
  if (!url || !endpoint) return res.status(400).json({ error: 'URL and endpoint
  if (!request_type) return res.status(400).json({ error: 'Request type required
  try { new URL(url); } catch { return res.status(400).json({ error: 'Invalid UR
  if (!endpoint.startsWith('/') || /\s/.test(endpoint)) {
    return res.status(400).json({ error: 'Endpoint must start with "/" and conta
  }
  if (/\s/.test(parameter)) {
    return res.status(400).json({ error: 'Parameter must contain no spaces' });
  }
  const job = await Job.create({ url, endpoint, parameter, datatype, selectedTes
  res.status(201).json(job);
};

// PUT /api/jobs/:id
exports.updateJob = async (req, res) => {
  const job = await Job.findByPk(req.params.id);
  if (!job) return res.status(404).json({ error: 'Job not found' });
  await job.update(req.body);
  res.json(job);
};

// DELETE /api/jobs/:id
exports.deleteJob = async (req, res) => {
  const job = await Job.findByPk(req.params.id);
  if (!job) return res.status(404).json({ error: 'Job not found' });
  await job.destroy();
  res.status(204).end();
};
```

Challenges & Insights

- Faced a learning curve with Sequelize and had to refactor existing logic without breaking endpoints
- Syncing local environments and aligning on DB setup caused minor setup delays

Collaborative Takeaway

- Helpful to ensure everyone is on the same page before beginning to work with data structure setup and DB setup
- Clear communication and collaborative code reviews ensured smooth PRs – adding detail to PR specifics is really helpful

Future Improvements & Next Steps

- Adding more vulnerability features?
- Cleaning and more aesthetically pleasing UI?
- More secure handling?
- Demo Day!!

Individual Team Member Slides

-Mason Lemberger-

Assigned Work Summary

- **Work done this milestone:** Implemented sequelize model for storage:
    - Set up savedJob, Users, jobResults, and Vulnerability models
    - Set respective tables inside of database.sqlite file for models
    - Updated front-end logic to work with the newly created api responses
+ Issue (closed) link:
    + https://github.com/MrLarryMan/Tensile/issues/47
    + https://github.com/MrLarryMan/Tensile/issues/51
+ PR (merged) link: https://github.com/MrLarryMan/Tensile/pull/50

- Status: this milestone dealt with the frontend to backend 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

Frontend:

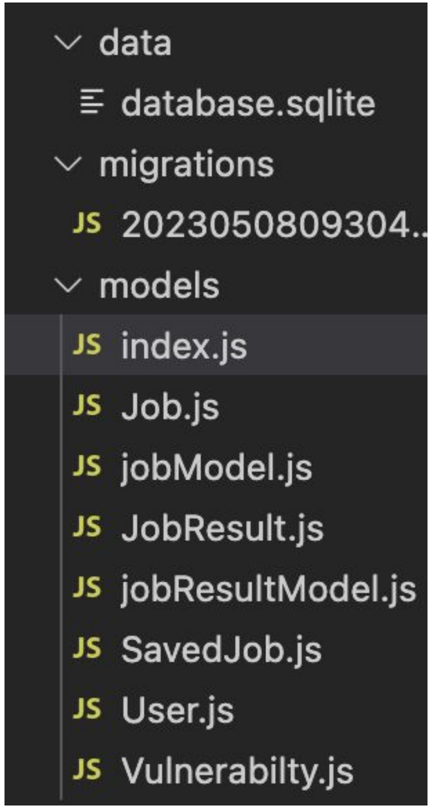
- Changed all necessary logic to account for database change
- Logic that utilizes sessionStorage to reduce api calls

Backend:

- Created models for Users, jobResults, savedJobs, and Vulnerabilities
- Created tables for the models
- Created the proper associations and changed the controller for the analytics page to account for new setup

Code Structure & Organization

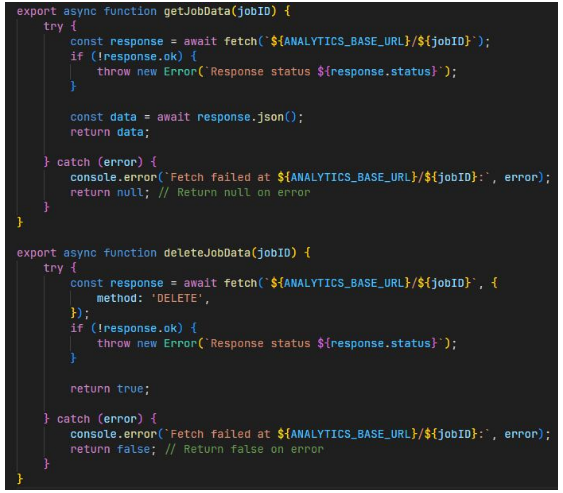
- Code structure:
  - this follows the class structure of controller, models, and routes
  - Also contains the migrations and config folder to easily change parts
- The front-end also have clearly labeled files to help understand the path the data takes
- Critical components:
  - All models are clearly labeled
  - Relationships between models are set in the index file
  - The database file is stored under data for clarity



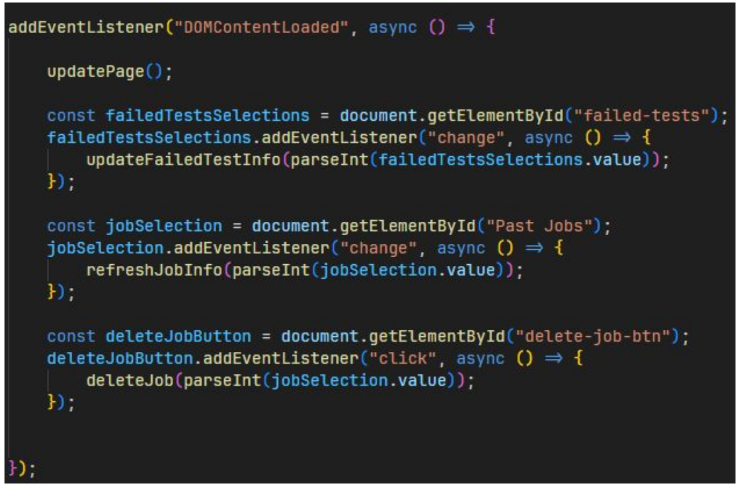
```
∨ data
    database.sqlite
∨ migrations
  JS 2023050809304..
∨ models
  JS index.js
  JS Job.js
  JS jobModel.js
  JS JobResult.js
  JS jobResultModel.js
  JS SavedJob.js
  JS User.js
  JS Vulnerabilty.js
```

Front-end

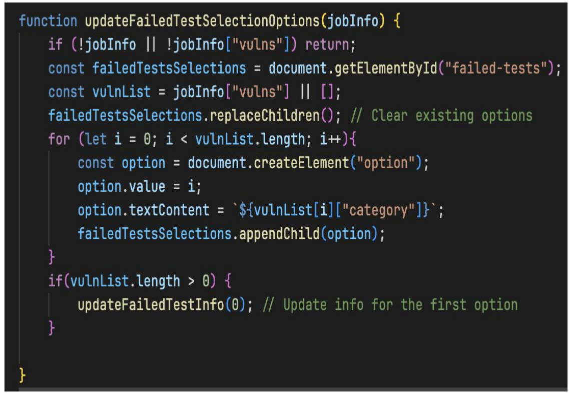
- Have multiple functions that ask the backend api for information (left picture)
- This data is then used to update the page (middle picture)
- For milestone 7 functions that used the old models were to updated to implement the new models (right picture)



```javascript
export async function getJobData(jobID) {
    try {
        const response = await fetch(`${ANALYTICS_BASE_URL}/${jobID}`);
        if (!response.ok) {
            throw new Error(`Response status ${response.status}`);
        }

        const data = await response.json();
        return data;

    } catch (error) {
        console.error(`Fetch failed at ${ANALYTICS_BASE_URL}/${jobID}:`, error);
        return null; // Return null on error
    }
}

export async function deleteJobData(jobID) {
    try {
        const response = await fetch(`${ANALYTICS_BASE_URL}/${jobID}`, {
            method: 'DELETE',
        });
        if (!response.ok) {
            throw new Error(`Response status ${response.status}`);
        }

        return true;

    } catch (error) {
        console.error(`Fetch failed at ${ANALYTICS_BASE_URL}/${jobID}:`, error);
        return false; // Return false on error
    }
}
```



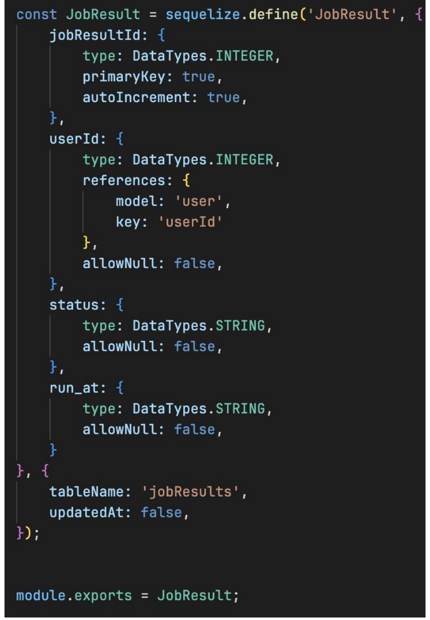
```javascript
addEventListener("DOMContentLoaded", async () => {

    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));
    });

    const deleteJobButton = document.getElementById("delete-job-btn");
    deleteJobButton.addEventListener("click", async () => {
        deleteJob(parseInt(jobSelection.value));
    });


});
```



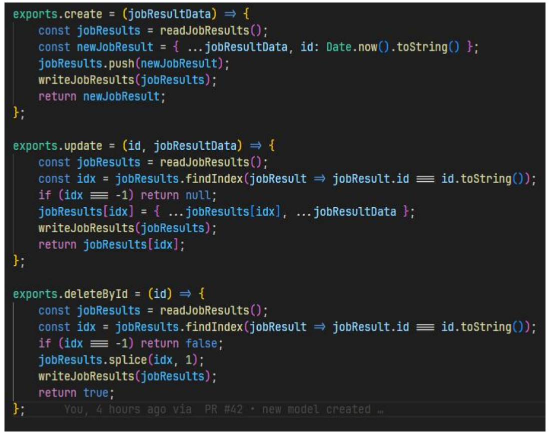
```javascript
function updateFailedTestSelectionOptions(jobInfo) {
    if (!jobInfo || !jobInfo["vulns"]) return;
    const failedTestsSelections = document.getElementById("failed-tests");
    const vulnList = jobInfo["vulns"] || [];
    failedTestsSelections.replaceChildren(); // Clear existing options
    for (let i = 0; i < vulnList.length; i++){
        const option = document.createElement("option");
        option.value = i;
        option.textContent = `${vulnList[i]["category"]}`;
        failedTestsSelections.appendChild(option);
    }
    if(vulnList.length > 0) {
        updateFailedTestInfo(0); // Update info for the first option
    }

}
```

Back-end

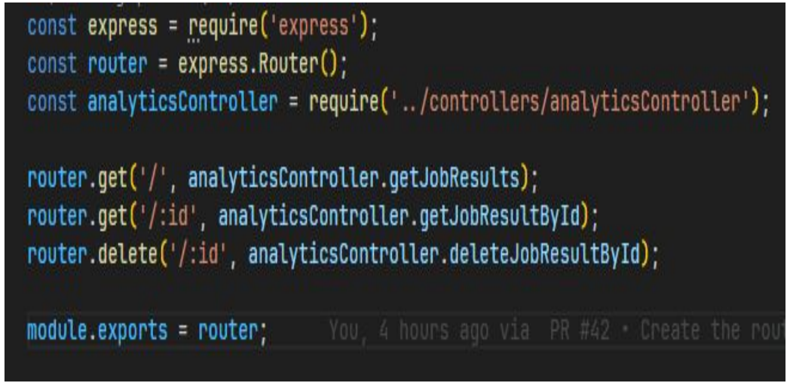
- Created the controller, routes, and model for the backend (bottom pictures)
- Created models like JobResult to utilize the sqlite storage (right picture)



```
const JobResult = sequelize.define('JobResult', {
    jobResultId: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true,
    },
    userId: {
        type: DataTypes.INTEGER,
        references: {
            model: 'user',
            key: 'userId'
        },
        allowNull: false,
    },
    status: {
        type: DataTypes.STRING,
        allowNull: false,
    },
    run_at: {
        type: DataTypes.STRING,
        allowNull: false,
    }
}, {
    tableName: 'jobResults',
    updatedAt: false,
});


module.exports = JobResult;
```



```
exports.create = (jobResultData) => {
    const jobResults = readJobResults();
    const newJobResult = { ...jobResultData, id: Date.now().toString() };
    jobResults.push(newJobResult);
    writeJobResults(jobResults);
    return newJobResult;
};

exports.update = (id, jobResultData) => {
    const jobResults = readJobResults();
    const idx = jobResults.findIndex(jobResult => jobResult.id === id.toString());
    if (idx === -1) return null;
    jobResults[idx] = { ...jobResults[idx], ...jobResultData };
    writeJobResults(jobResults);
    return jobResults[idx];
};

exports.deleteById = (id) => {
    const jobResults = readJobResults();
    const idx = jobResults.findIndex(jobResult => jobResult.id === id.toString());
    if (idx === -1) return false;
    jobResults.splice(idx, 1);
    writeJobResults(jobResults);
    return true;
};      You, 4 hours ago via  PR #42 • new model created …
```



```
const express = require('express');
const router = express.Router();
const analyticsController = require('../controllers/analyticsController');

router.get('/', analyticsController.getJobResults);
router.get('/:id', analyticsController.getJobResultById);
router.delete('/:id', analyticsController.deleteJobResultById);

module.exports = router;      You, 4 hours ago via  PR #42 • Create the rou
```

Challenges & Insights

- Obstacles faced and lessons acquired: One challenge I faced was figuring out was how to implement code that heavily relies on other team members. This expanded the way I thought about coding and how to make my code more approachable to other members of the project. **This became even more apparent for milestone 7, this is still the main lesson from this milestone.**

- Key takeaways from working within a collaborative team environment: Once your code starts communicating across the different sections, it becomes very important to be very detailed oriented when discussing future plans and how work needs to be delegated. If this doesn't happen it can quickly become confusing. **Again, this became more apparent for this milestone. Most of my time was spent communicating with my team members rather coding.**

Future Improvements & Next Steps

- Future improvements or features: A new feature that could be implemented is allowing users to click on nodes of the history graph and running that test. This could allow users to get better insight into how much they have improved their website.

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

- Future Issue link: https://github.com/MrLarryMan/Tensile/issues/48

Individual Team Member Slides

-Larry Liu-

Assigned Work Summary

- **Work done this milestone:** Did extensive work refactoring the backend code to be compatible with the sequelize databases, and fixed miscellaneous bugs throughout all aspects of the code to prep the website for demo day (including router, controller, model, and frontend html/js code). Also created vulnerable target websites with node + docker to test our scanner. Added a "parameter" parameter to job details as well integrating it with existing systems and adding functionalities to the scanner.
+ Issue (closed) link: https://github.com/MrLarryMan/Tensile/issues/59 https://github.com/MrLarryMan/Tensile/issues/58
+ PR (merged) link: https://github.com/MrLarryMan/Tensile/pull/64 , https://github.com/MrLarryMan/Tensile/pull/57 , https://github.com/MrLarryMan/Tensile/pull/54 , https://github.com/MrLarryMan/Tensile/pull/52 https://github.com/MrLarryMan/Tensile/pull/50

- Status: The milestone covers backend/persistence features, as well as a requirement to finalize the project which have been completed → Completed & closed all issues in this milestone
- Non-closed Issues: N/A

Feature Demonstration

Test Website Targets:

- Created two dockerized target websites vulnerable to XSS and Path Traversal, respectively.

Modified Models

- Modified several of the models & refactored the code to conform to a new standard.

Parameter parameter

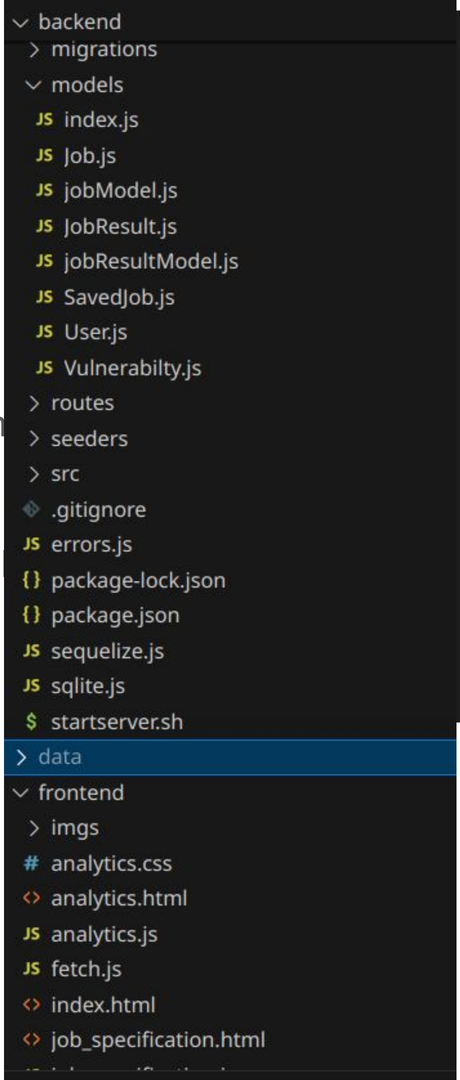
- Create a new job parameter, “Parameter”, representing a query parameter.

Specification and Saved Jobs Functionality

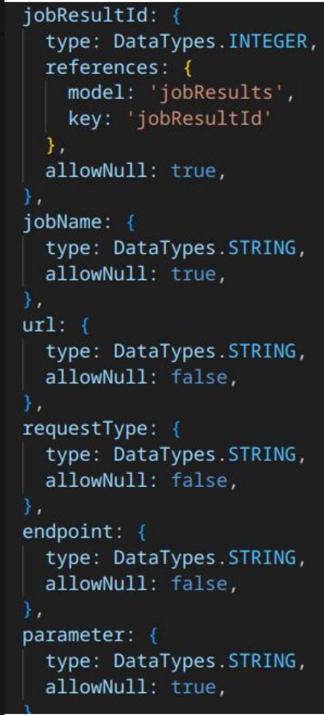
- Fixed functionality on the job spec and saved jobs pages to save/run jobs.

Code Explanation:

- Code Structure: My code modifications follow the class structure of sequelized models. Although I did not make any persistent features in their entirety, I made extensive use of them in the backend code and modified/patched many of them as well.
- All of the backend files are under the backend folder and the frontend field are under the frontend folder to help with clarity. I have contributions in all folders and subfolders for this milestone.
- Critical components: all code containing functionality relating to the scanning functionality is denoted with scanner in the name.
- The scanning is primarily handled through the scannerMain function, which imports other javascript files that contain the code for scans of specific vulnerability types. Tools used were node puppeteer for XSS and fetch for Path Traversal.



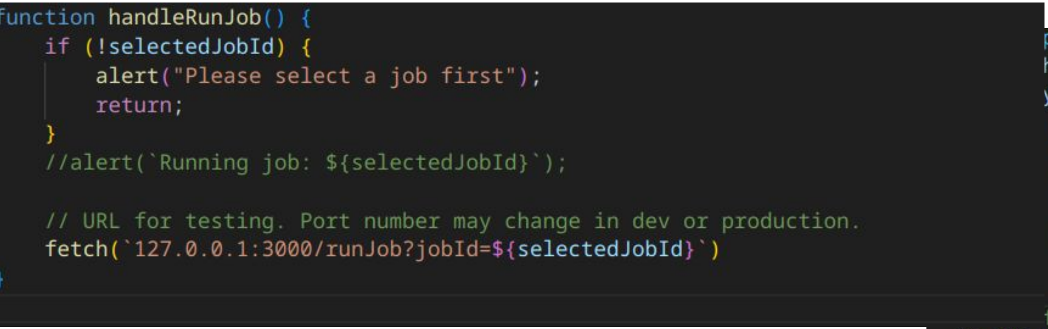
```
v backend
  > migrations
  v models
    JS index.js
    JS Job.js
    JS jobModel.js
    JS JobResult.js
    JS jobResultModel.js
    JS SavedJob.js
    JS User.js
    JS Vulnerabilty.js
  > routes
  > seeders
  > src
  .gitignore
  JS errors.js
  {} package-lock.json
  {} package.json
  JS sequelize.js
  JS sqlite.js
  $ startserver.sh
> data
v frontend
  > imgs
  # analytics.css
  <> analytics.html
  JS analytics.js
  JS fetch.js
  <> index.html
  <> job_specification.html
```



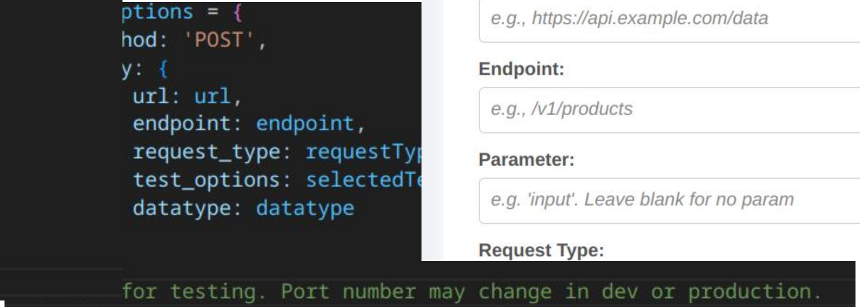
```
jobResultId: {
  type: DataTypes.INTEGER,
  references: {
    model: 'jobResults',
    key: 'jobResultId'
  },
  allowNull: true,
},
jobName: {
  type: DataTypes.STRING,
  allowNull: true,
},
url: {
  type: DataTypes.STRING,
  allowNull: false,
},
requestType: {
  type: DataTypes.STRING,
  allowNull: false,
},
endpoint: {
  type: DataTypes.STRING,
  allowNull: false,
},
parameter: {
  type: DataTypes.STRING,
  allowNull: true,
```

Front-end

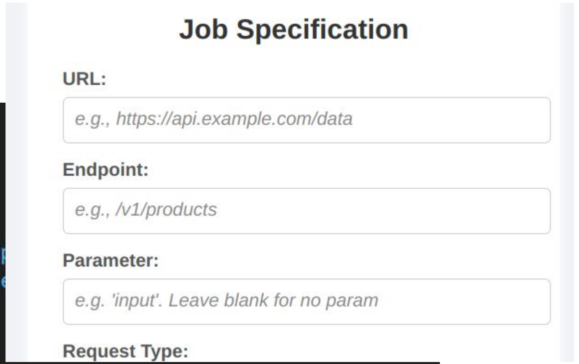
- Fixed any errors in the frontend code, mostly in job specification and saved jobs, that interfaces with the database or scanner functionality (specifically standardized & fixed the fetch request system they used).
- Added fields to display the “parameter” parameter.
- After jobs are run, the user is notified of the progress and can navigate to the results/analysis page to see the results of the job.
- Inconsistencies in the routes and models made finding the right input for fetch requests difficult



```
unction handleRunJob() {
    if (!selectedJobId) {
        alert("Please select a job first");
        return;
    }
    //alert(`Running job: ${selectedJobId}`);

    // URL for testing. Port number may change in dev or production.
    fetch(`127.0.0.1:3000/runJob?jobId=${selectedJobId}`)
}
```



```
ptions = {
hod: 'POST',
y: {
 url: url,
 endpoint: endpoint,
 request_type: requestTy
 test_options: selectedT
 datatype: datatype

for testing. Port number may change in dev or production.
```



Job Specification

URL:
e.g., https://api.example.com/data

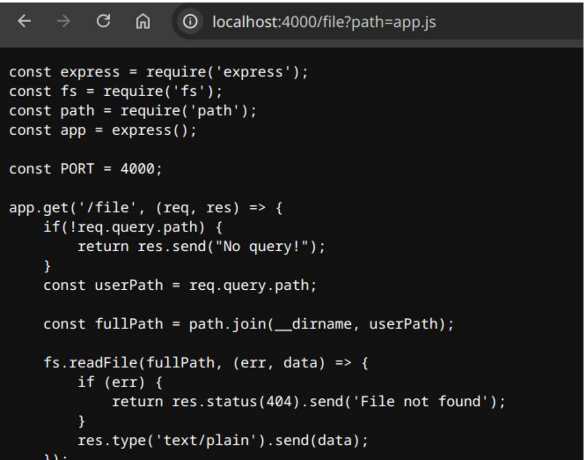
Endpoint:
e.g., /v1/products

Parameter:
e.g. 'input'. Leave blank for no param

Request Type:

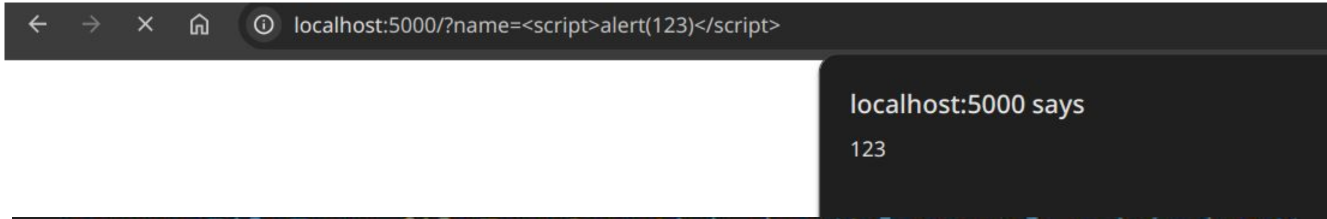
Back-end

- Modified the scanners to use methods for the new database system, and rebased the output to conform to changes to the models.
- Created dockerized target websites vulnerable to XSS and Path Traversal. (Technically not backend but best fit)
- Inconsistencies in the routes and models made returning the right format difficult.



localhost:4000/file?path=app.js

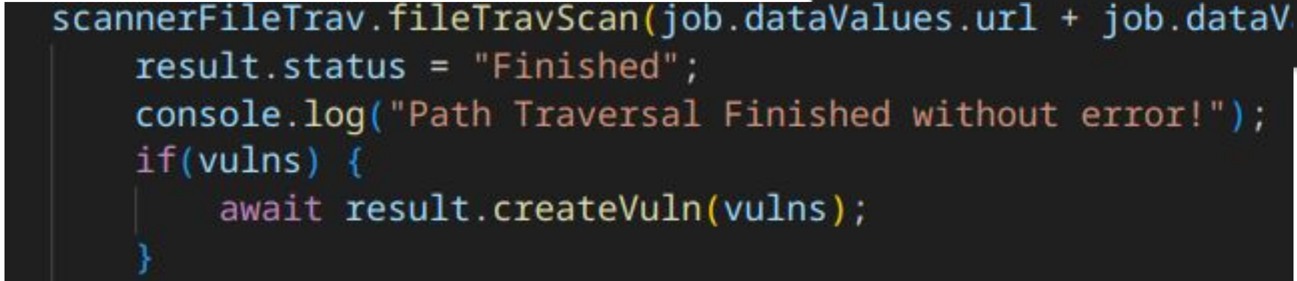
```
const express = require('express');
const fs = require('fs');
const path = require('path');
const app = express();

const PORT = 4000;

app.get('/file', (req, res) => {
    if(!req.query.path) {
        return res.send("No query!");
    }
    const userPath = req.query.path;

    const fullPath = path.join(__dirname, userPath);

    fs.readFile(fullPath, (err, data) => {
        if (err) {
            return res.status(404).send('File not found');
        }
        res.type('text/plain').send(data);
```



localhost:5000/?name=<script>alert(123)</script>

localhost:5000 says

123



```
scannerFileTrav.fileTravScan(job.dataValues.url + job.dataV
    result.status = "Finished";
    console.log("Path Traversal Finished without error!");
    if(vulns) {
        await result.createVuln(vulns);
    }
```

Challenges & Insights

- Obstacles faced and lessons acquired: As we neared the end of the project, we realized that our project was inadequately tested when it came to how all the individual parts fit together, which made the last stretch longer and more stressful than needed. We also found that we didn’t coordinate enough when it came to the data structures required for our code, as we only discussed them at the very start, which resulted in many struggles as certain data was self-referential and could not be defined until other data, which was also dependent on it, was made.
- Key takeaways from working within a collaborative team environment: Although it’s easy to brush off, having a solid testing strategy and accountability for testing is important for making work on a project run smoothly. It’s also important to coordinate with your teammates should you make a feature/structure that is shared between all developers of the program.

Future Improvements & Next Steps

- Future improvements or features: Implement additional tests, test types, and customizability for tests. Perform additional error catching, and add back support for jobs containing multiple tests as well as the terminate job feature.
- Technical debt or aspects that could be optimized further: See above. We also have some redundant code that we have deprecated. We will need to discuss what to do with these features.

- Issue link: https://github.com/MrLarryMan/Tensile/issues/65