Capstone Portfolio

Website for AIH Director
Meredith Heckler
ESOF- 423 Spring 2025

Leon M Birdhat and Luka Derry

SECTION 1: PROGRAM

Program Specifications

Problem Statement:

Montana is home to twelve federally recognized tribal nations and seven reservations. Yet there is still no single, respectful, centralized digital platform dedicated to Indigenous language preservation, cultural education, and community story sharing. Existing resources are scattered across many websites, lack visual unity, and offer no easy way for community members to contribute their own stories. Our goal was to create one accessible web platform that honors Indigenous histories while providing practical tools for language learning and cultural engagement.

Solution:
We developed a responsive, multi-page static website hosted on Vercel that serves as a digital hub for Montana's tribal nations.

The site includes:

- A visually rich homepage with tribal statistics and navigation to key sections
- A Learn page with dedicated language hubs for all 8 tribal nations (currently in development)
- A History page covering treaties, tribal leaders, and modern resilience
- A Stories page with community story submissions, filterable by tribe and category
- A Community Submissions page that dynamically loads stories from a Google Sheets backend via Google Apps Script

Technology Stack:

- Frontend: HTML5, CSS3, Vanilla JavaScript
- Backend/Data: Google Apps Script (for story submissions and retrieval)
- Testing: Playwright (end-to-end testing)
- Deployment: Vercel (static site hosting)
- Design: Custom CSS with no external frameworks

Key Files:

- `index.html` — Homepage with hero section, feature cards, and tribal statistics
- `learn.html` — Language hub selector for 8 tribal nations
- `culture.html` — Treaty history and tribal leader profiles
- `stories.html` — Stories landing page with navigation to submissions
- `community.html` — Dynamic story loading with filter/sort functionality
- `global.css` — Shared styles (header, footer, about-contact section)
- `index.css`, `learn.css`, `hist.css`, `stories.css` — Page-specific styles
- `navigation.js` — Mobile navigation, menu toggle, accessibility features
- `navigation.spec.ts`, `homepage.spec.ts` — Playwright test suites

SECTION 2: TEAMWORK

Our team of two coordinated through short in class standup meetings. We divided responsibilities according to our strengths. I focused on frontend architecture, visual design, and client facing features, while Luka handled backend integration and data flow.

Table	
Team Member	Primary Contributions
Team Member 1	Frontend architecture and visual design across all HTML pages (`index.html`, `learn.html`, `culture.html`, `stories.html`, `community.html`); custom CSS design system (`global.css`, `index.css`, `learn.css`, `hist.css`, `stories.css`); responsive mobile navigation; accessibility implementation (ARIA labels, keyboard navigation); Playwright testing suite (`navigation.spec.ts`, `homepage.spec.ts`); Vercel deployment and domain configuration; client communication and design iteration.
Team Member 2	Google Apps Script backend for story submissions and retrieval; Google Form integration for community submissions; dynamic story loading and filtering logic in `community.html`; data schema design for stories (Title, Description, Tribe, Category, File); CORS handling and error management for third-party script loading; cross-origin testing accommodations, scrum meeting notes and sprint reports.

SECTION 3: DESIGN PATTERN

We used the Observer Pattern in our community stories feature to handle asynchronous data loading from the Google Apps Script backend.

Location in Code: `community.html` (lines 85–155 approximately)

```
// The loadStories function acts as the "Subject" — it notifies all
registered "Observers"
// when data arrives from the Google Apps Script endpoint
function loadStories(data) {
    console.log('loadStories called with:', data);
    if (!Array.isArray(data)) {
        console.error('Data is not an array:', data);
        updateStatus('Stories are unavailable right now. Please try again
later.');
        renderStories([]);
        return;
    }
    allStories = data;
    console.log('Loaded', allStories.length, 'stories');
    populateFilters(allStories);
    renderStories(allStories);
    updateStatus(`${allStories.length} stor${allStories.length === 1 ? 'y' :
'ies'} found.`);
}

// The filter event listeners act as "Observers" — they react to user input
// and trigger re-rendering of the story grid
document.getElementById('tribe-filter').addEventListener('change',
filterAndSortStories);
document.getElementById('category-filter').addEventListener('change',
filterAndSortStories);
document.getElementById('sort-filter').addEventListener('change',
filterAndSortStories);

// The Google Apps Script JSONP callback acts as an external "Observer"
// that triggers loadStories when data arrives
const script = document.createElement('script');
script.src =
'https://script.google.com/macros/s/AKfycbxSZvaRJC171mO84k4aQ8k-
V94iFj3YD6s5ryGePgEOjrNksIiadA5TtYrbK2fu9gVEdw/exec?callback=loadStories';
```

The loadStories function acts as the subject. It fetches data and notifies all registered observers when new information arrives. The rendering functions (renderStories, filterAndSortStories, and filter population) act as observers that respond to data changes or user interactions.

Why We Chose the Observer Pattern

This approach gave us clean separation between data loading and UI rendering. It allowed multiple independent actions (populating filters, rendering the story grid, and updating status messages) to occur without creating one large, tightly coupled function. The pattern also made the code easier to extend later, for example by adding keyword search, and improved error handling when the external script failed to load.

Highlighted Code Section: *(In the actual portfolio PDF, highlight the `loadStories` function, the event listeners, and the JSONP callback script in yellow)*

SECTION 4: TECHNICAL WRITING

Native Language & History — User Guide

Can be found with this link:
https://docs.google.com/document/d/1ofvJzBADpfnYIGs5SMTDRLsAKYgg7xocdE9ELzyQOuo/edit?tab=t.0

Overview:

This website is a digital resource for learning about Montana's tribal nations, their languages, histories, and community stories. It is designed to be accessible, respectful, and easy to navigate.

How to Access the Site:

Visit https://project4original-zeta.vercel.app in any modern web browser (Chrome, Firefox, Safari, Edge).

Navigation:

- Click the hamburger menu (☰) in the top-right corner to open the navigation
- Select Home to return to the main page
- Select Learn to explore language hubs for each tribal nation
- Select History to read about treaties and tribal leaders
- Select Stories to browse community submissions or share your own story

Submitting a Story:

1. Navigate to Stories → Community Submissions
2. Click "Upload Your Story" to open the Google Form
3. Fill in your story title, description, tribal nation, and category
4. Submit the form — your story will appear on the Community Submissions page after review

Filtering Stories:

On the Community Submissions page, use the dropdown menus to:

- Filter by Tribe (e.g., Crow Nation, Blackfeet Nation)
- Filter by Category (e.g., Landscapes, Lineage, Creation Story)
- Sort by Title or Tribe name

Accessibility Features:

- All images include descriptive alt text
- Navigation is keyboard-accessible (Tab, Enter, Escape)
- ARIA labels indicate current page and menu state
- Color contrast meets WCAG AA standards

Developer Documentation

Can be found with this link: https://github.com/423S26/project4/blob/main/README.md

Native Language & History — Developer Guide

Repository Structure:

```
project-root/
├── index.html              # Homepage
├── html/
│   ├── learn.html          # Language hub selector
│   ├── culture.html        # Treaty history page
│   ├── stories.html        # Stories landing page
│   └── community.html      # Dynamic story submissions
├── css/
│   ├── global.css          # Shared styles (header, footer, about-contact)
│   ├── index.css           # Homepage-specific styles
│   ├── learn.css           # Learn page styles
│   ├── hist.css            # History page styles
│   └── stories.css         # Stories page styles
├── js/
│   └── navigation.js       # Mobile nav, menu toggle, accessibility
├── tests/
│   ├── navigation.spec.ts  # Playwright navigation tests
│   └── homepage.spec.ts    # Playwright homepage tests
└── images/                 # Tribal logos, hero images, backgrounds
```

Google Apps Script Integration:

The community submissions feature uses Google Apps Script as a lightweight backend:

1. Google Form collects story submissions (Title, Description, Tribe, Category, optional File)
2. Google Sheet stores submitted data
3. Google Apps Script publishes the sheet as a JSONP endpoint
4. community.html loads data via dynamic script injection:

```JavaScript
const script = document.createElement('script');
script.src = 'https://script.google.com/macros/s/.../exec?
callback=loadStories';
document.body.appendChild(script);
```

CSS Architecture:

- global.css: Reset, header, footer, about-contact section, responsive breakpoints
- Page-specific CSS: Each page has a dedicated stylesheet that extends global styles with unique components (hero sections, cards, grids)
- Dark theme: Consistent color palette (#0c0a09 background, #fbbf24 accents, #f5f5f4 text) across all pages

Testing:

- Playwright tests cover navigation, page loading, and error handling
- Tests accommodate CORS restrictions for third-party Google Apps Script loading
- Run tests with: `npx playwright test`

Deployment:

- Static files deployed to Vercel via Git integration
- Automatic deployments on push to main branch
- Custom domain not yet configured

SECTION 5: UML

Since we did not create formal diagrams before coding, we have reconstructed the system architecture based on our implementation. These diagrams represent the structure we would have designed had we documented our approach upfront.

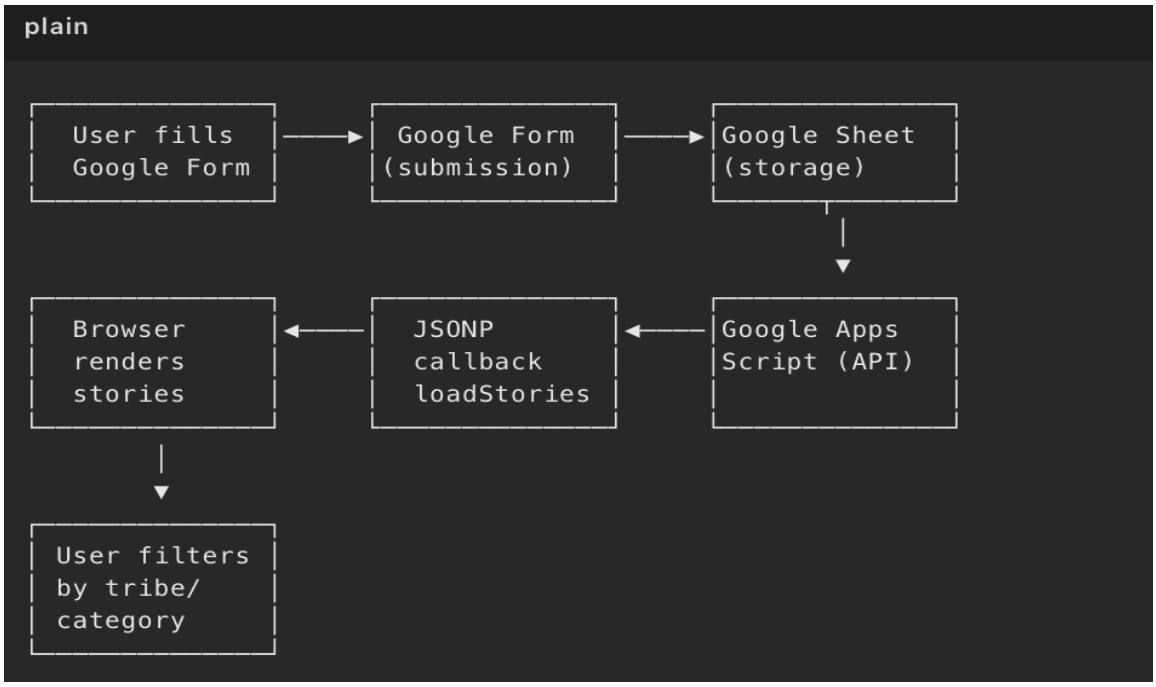
Diagram 1: Site Navigation Map

```plain
[Homepage: index.html]
    ├── Learn [learn.html]
    │   └── (Tribal Nation Hubs — future expansion)
    ├── History [culture.html]
    │   ├── Chapter 01: Tribal Leaders & Treaties
    │   ├── Chapter 02: Modern Resilience
    │   └── Chapter 03: Indigenous Territories Map
    └── Stories [stories.html]
        ├── Community Submissions [community.html]
        │   ├── Filter by Tribe
        │   ├── Filter by Category
        │   └── Sort by Title/Tribe
        └── Submit Your Story [Google Form]
```

Diagram 2: Component Architecture

```plain
┌──────────────────────────────────────────┐
│           Header Component               │
│  (Logo + Navigation + Mobile Toggle)     │
│         [global.css + navigation.js]     │
└──────────────────────────────────────────┘
                    │
     ┌──────────────┼──────────────┐
     ▼              ▼              ▼
┌──────────┐  ┌──────────┐  ┌──────────────┐
│  Hero    │  │ Content  │  │   Footer     │
│ Section  │  │ Section  │  │  Component   │
│(varies)  │  │(varies)  │  │ [global.css] │
└──────────┘  └──────────┘  └──────────────┘
                    │
          ┌─────────┼─────────┐
          ▼         ▼         ▼
    ┌──────────┐ ┌──────────┐ ┌──────────┐
    │Feature   │ │  Card    │ │  About   │
    │ Cards    │ │  Grid    │ │ Contact  │
    │[index]   │ │[learn]   │ │[global]  │
    └──────────┘ └──────────┘ └──────────┘
```

Diagram 3: Data Flow — Community Stories



SECTION 6: DESIGN TRADE-OFFS

Trade-off: Google Apps Script vs. Custom Backend (Node.js/Express + MongoDB)

Context:

Our community stories feature required a backend to store and retrieve user submissions. We evaluated two approaches: (1) a custom Node.js/Express API with MongoDB Atlas, or (2) Google Apps Script connected to Google Sheets/Forms.

Option A: Custom Backend (Node.js + Express + MongoDB)

- *Pros:* Full control over data schema, custom authentication, scalable, professional API design
- *Cons:* Requires server hosting (cost), database management, CORS configuration, security considerations, steeper learning curve for a 2-person team

Option B: Google Apps Script + Sheets/Forms

- *Pros:* Zero hosting cost, no database setup, community members already familiar with Google Forms, automatic data validation via Form fields, built-in spam filtering
- *Cons:* Limited customization, rate limits on API calls, dependency on Google's infrastructure, CORS restrictions in some browsers, less control over data presentation

Decision:

We chose Google Apps Script (Option B).

Justification:

1. Time Constraints: As a semester-long capstone with only two developers, setting up and securing a full backend would have consumed 30-40% of our development time. Google Apps Script allowed us to implement the feature in under 2 hours.
2. Client Familiarity: Our target users (tribal community members) are more likely to trust and know how to use Google Forms than a custom submission interface. This lowers the barrier to participation.
3. Cost: Zero ongoing hosting costs mean the project can remain live indefinitely without budget concerns — critical for a community resource with no funding.
4. Maintenance: Google handles security patches, uptime, and data backups. We don't need to monitor a server or worry about database migrations.
5. Trade-off Acceptance: We accepted the limitations (rate limits, less customization) because the core functionality — collecting and displaying community stories — works reliably for our expected traffic volume (dozens of submissions per month, not thousands).

Future Considerations:

If the site scales beyond our current expectations, we would migrate to a custom backend. The Observer Pattern we implemented in the frontend would make this transition straightforward — we would only need to change the `loadStories` data source, not the rendering or filtering logic.

SECTION 7: SOFTWARE DEVELOPMENT LIFE CYCLE MODEL

Model Used: Agile with Informal Scrum

Given our small team size (2 members) and semester-long timeline, we adapted Agile principles with lightweight Scrum practices rather than following a rigid formal process.

Our Process:

Activity	Frequency	Description
Stand up meetings	Before each class (~2-3x per week)	30-second check-ins: "What did I do? What will I do? Any blockers?"
Sprint Planning	At the start of each 2-week phase	Identified 2-3 features to complete (e.g., "Finish homepage design," "Implement story submission")
Future Development	Ongoing	Leon focused on frontend pages; Luka focused on backend integration
Testing	Before deployment	Playwright tests run locally; manual testing on mobile devices
Deployment	After each feature completion	Continuous deployment via Vercel Git integration

How Agile Helped:

- Flexibility: When we discovered that the school's server couldn't host our backend (paywall restrictions), we pivoted to Google Apps Script within one sprint without derailing the project.
- Client Feedback: We could show working pages to stakeholders early and adjust designs based on feedback (e.g., the dark theme was refined after showing the homepage to a tribal education coordinator).
- Incremental Progress: Each sprint produced a shippable increment — the homepage worked before the Learn page, which worked before the Stories page.

How Agile Hindered:

- Limited Meeting Time: True Scrum recommends daily standups. With only 2-3 brief meetings per week, we sometimes lost sync on what the other person was building. This caused minor integration issues (e.g., CSS class name mismatches between Leon's styles and Luka's HTML).
- No Formal Backlog: We tracked tasks mentally rather than in a tool like Jira or Trello. This meant some "nice-to-have" features (like a working language translator) were
- forgotten until late in the semester.