Seneca Park Zoo
Conservation Impact Tracker

Seneca Park Zoo Tigers

Jared Jasie Anthony Palumbo
Nikolas Tilley Edward Wong

Project Sponsor

Tom Snyder, Pamela Reed-Sanchez

Faculty Coach

J Scott Hawker

Project Overview

This project is the first continuation of the Seneca Park Zoo Conservation Impact project started in the Spring of 2019. The project, sponsored by the Seneca Park Zoo Society, aims to leverage current technologies to aggregate and disseminate conservation data produced by zoos at scale which will ultimately benefit the zoos as well as their donors.

This project intends to give zoos the ability to easily track and manage their conservation projects, and share that information with other zoos. Zoo staff will be able to select what projects they want to check on and add new data to it. The adding of new data will be done by importing a spreadsheet into the MyConservationLife database that was established by the previous team. Conversely, zoo staff will also be able to download data from the database in the form of a spreadsheet. In addition to feature, they may download a starter template spreadsheet in which they can fill with data of their own.

Zoo guests who donate towards an ecological asset will be able to see its growth over time through the data zoo staff imports into the database. They will be able to see every project they have donated to as well as all the assets they have sponsored. These assets can be viewed chronologically within a list and also geospatially on a map. Additionally, donors will also be able to explore other projects associated with non-local zoos.

The customers for the Seneca Park Zoo Society range from individual visitors of the zoo to organizations making large donations. At the zoo, someone can purchase a donor code to sponsor a new tree being planted. The unique donor code then allows them to register the tree online and make an account for themselves. Once the tree has been planted, they will be able to view geospatial, temporal, and visual data that has been collected by someone where a reforestation project is taking place.

Basic Requirements

Inheriting the project from a previous senior project means that we inherited a backlog of pre-existing function requirements. Additionally, with the size of the task at hand it was expected that we would be making functional requirements outside the scope of our own development. As a result, we will only discuss the requirements that our team was directly responsible for, in order of relevance.

In order to start small and learn the technology stack, the team started with implementing the ability to create and update conservation projects. The user story associated with this feature was:

> **Project Creation LWC** *"As a zoo staff, I want to create new conservation projects so that I can keep track of my project."*

The requirement derived from this user story was that the system shall have a Salesforce Lightning Web Component that enables a Zoo Staff user to create and edit projects stored in the Open Source Database. Additionally, the system shall display a list of existing projects. This assumes that the Zoo Staff user is a registered user and their account has access to the Project Creation LWC.

The next feature that was implemented was the ability to query asset data geospatially from the Open Source Database. This feature is closely related to the subsequent UI requirement, but due to the nature of our system architecture we were able to clearly define separate requirements:

Geospatial API: *"As a researcher or donor, I want an API to query for data via their location, so that I can study their geospatial relationships."*

After some further elicitation, the requirement that we created was the following: Using a RESTful interface, the system shall provide an interface for querying assets given a center point and a radius. The system shall provide an interface for querying assets within a rectangular area. The system shall provide an interface for querying assets enclosed by a set of coordinate points. This interface allows the database to be interacted with in various ways. Programmers and technical researchers can use libraries like curl to query data from the command line or programers can create graphical user interfaces to query and parse the data. As stated, there was a UI requirement to go with the Geospatial API:

Geospatial Map Frontend: *"As a researcher, I want to be able to query for data via their location using a Map, so that I can study their geospatial relationships."*

In order to contain the scope of this user story, we concluded that a minimal implementation would allow the user to draw shapes on a Leaflet Map and display assets that were contained within their geometry. Thus, the Geospatial Map shall be an LWC that provides graphical tools to query data by location using the three interfaces available in the API (point-radius, rectangle, polygon).

To populate the assets in the database, the system required a way for users to upload their data. The sponsor emphasized "flexibility" but also valued correctness and understood the vagueness of flexibility. The desire is to expand the acceptable data formats in the future but

more importantly we wanted to show an alternative to the paper methods many zoos and researchers are still using:

Asset Data Import: *"As a Zoo Staff or Field Participant, I would like to be able to import my asset data into the Open Source Database via a CSV, so that it can be studied on a Map."*

The system shall provide an LWC that allows the user to upload a CSV containing data about assets in the Open Source Database. This requirement assumes that the asset type exists in the database and that the CSV is properly formatted. To make it easier on the user to format their CSVs, the system provides an LWC that allows the user to download the required headers of an existing asset type in the database:

Asset Headers Download LWC: *"As a Zoo Staff or Field Participant, I want to be able to export CSV headers for an existing asset type, so that I can format my data in a way that will be accepted by the Open Source Database."*

In combination with the Asset Type Definition LWC the previous team had created, the Asset Headers Download LWC provides a measure of flexibility the sponsor desired while maintaining some consistency in data structure.

Zoo Staff and Field Participants may want access to a more traditional method of studying ecological data. Thus, we were required to provide a means of exporting asset data in CSV format:

Asset Data Export LWC: "As a Zoo Staff or Field Participant, I would like to export Asset CSV data, so that I can study it using alternative methods."

The system shall have an LWC that allows a registered user to select an existing asset and download its data in a CSV format. The CSV shall NOT include data marked as PRIVATE. The last part of this requirement is to ensure that possible sensitive data (e.g. the locations of endangered species) are not publicly available for exploitation.

In order to track the impact of a donation, it was required that the Open Source Database track how assets' data changes over time:

Temporal API: *"As a researcher or donor, I want an API to query asset data's historical values, so that I can study how the asset changed over time."*

This requirement was conceptualized by the previous team and fleshed out by ours. The system shall provide an API that allows users to retrieve historical asset data between a start date and an end date.

Finally, with most systems, requirements change over time and this final one was negotiated near the end of the project. We received direction that the new priority was to implement a feature to pair assets with donor codes and to retrieve assets by donor code. The user story that corresponds with this feature was as follows:

Donor Code Map: *"As a Donor, I want to see the assets that I purchased on a Map, so that I can see my impact."*

Assuming that the Donor is registered with MyConservationLife and that they have registered their Donation code on Salesforce, the system shall have an LWC that can display the

donor's assets on a map. This requirement implied that we are to track the donation code in the Open Source Database. It also implies that the system shall have an API to query assets via their donation code.

Above are the major requirements that were in the scope of our team's development. There are many more requirements that still need to be elicited and implemented: some of these are Asset Stages, Donor Code Pairing LWC, API Authentication, and more. With only a finite amount of time to dedicate to a large and ambitious project, we leave these to future teams to expound upon and negotiate.

Constraints

1. **Project Tech Stack:** The tech stack for this project was set according to the specifications of our sponsor. It consisted of Salesforce Lightning Web Components on the front-end and Heroku, Node.js, and PostGIS handling the back-end. This was the same set of technologies that the previous team worked with and they established a well-developed framework for us to inherit. Since none of us were experienced in any part of the stack and the stack possessed a high learning curve, it took a lot of time for us to grasp how to use it. This led to a slow start when beginning the project.

2. **Database Design:** As stated before, a previous team had the opportunity to work on this project before us. One of their main deliverables was a PostGIS relational database that they successfully designed and established. While it was designed to be flexible to accept any sort of ecological data, it was not without issues. After our team took over, we had concerns that with the way the database was designed, there could be performance and scalability issues when containing a production-level amount of data. We brought up the idea of switching to a document-oriented database. However, the previous team had already implemented much of their codebase to fit with their relational database design and we did not think it was worth the time and effort to research, establish a new database, and change what they already implemented to fit the new database. On top of this, there were new features that were prioritized to be completed such as the geospatial map and importing/exporting CSV capabilities.

3. **Salesforce Consultant Availability:** A Salesforce consultant was supposed to be available from the start of the Fall semester. Unfortunately, due to prolonged contract negotiations with our sponsor, they were unavailable until the middle of the Spring semester. Having a consultant available from the get-go would have allowed us to get acclimated with Salesforce and Lightning Web Components faster. When they became available, they were a helpful resource in terms of learning deeper features of the Salesforce platform and visualizing what the end product was projected to look like.

4. **Heroku:** A minor constraint was that the Heroku account we used was on a Free plan. This limited our databases to only be able to hold 10,000 rows of data. This is considered minor as Heroku waits for a long duration before taking action against overloaded databases. Additionally, we did not reach a point where we performed formal load and stress testing so we did not require too many rows.

5. **Packaging Salesforce LWCs:** In terms of packaging and delivering our Lightning Web Components, we had to use the Salesforce platform to do so. Salesforce allows Lightning Web Components to be bundled into unmanaged packages. These packages can be transferred from a sandbox Salesforce org to be used in a production org on a different

Salesforce account. This was not a problematic constraint as it was a simple and quick process.

Development Process

As this was a continuation project, the majority of the technical decisions had already been made for us by the previous team. Included in that was using JavaScript for the codebase, Heroku and PostGIS for data management & storage, and Salesforce as the user interface.

At the start of the project, we had a few primary directives:

- Understand the decisions made by the previous team. This was both for our knowledge, as well as a "check" to ensure that those decisions were viable and something we believed we could build upon.
- Familiarize ourselves with the given tools. Prior to this project, our collective experience with JavaScript, Heroku, PostgreSQL, Salesforce, and all encompassing components was near-zero. As a result, the first few sprints of our "development" was reading documentation, doing tutorials, and trying to get a clear enough understanding so that we weren't developing from a disadvantageous position.
- Clarify the task backlog. Our Trello board was already set up with a plethora of tasks to be completed, some of which were foreign to us. In order to dispel any confusion, we reviewed a significant number of the tasks to both understand what they meant and to ensure that they were in the correct order; in terms of process as well as necessity.

Once we understood what we were expected to do and what the tasks were, we developed our own methodology that we called "SCRUMptious (a more palatable SCRUM)". We used the scrum methodology as our basis in order to be flexible with our development, and made modifications and changes that suited our project as we went.

There are a few primary features of SCRUMptious that framed our process:

- **Daily (meeting day) standups.** In an effort to keep people on track, stay aware of the team's general path, and provide support as needed, we started all of our meetings with a person-by-person review of the work we had completed, our current tasks & progress, and any blockers or issues we had. In the Fall we met four times a week (Monday, Tuesday, Thursday, Friday) but retired the Friday meetings in the Spring. We determined that regularly meeting four times a week was too often, and in some cases was detrimental to our workflow. There were some weeks where we met four times (or more), but only on a case-by-case basis.

- **Backlog grooming.** We utilized group estimation in order to determine the amount of effort for a task, with the following breakdown: 1 point - 1 to 2 days of work; 2 points - 2 days to 1 week of work; 3 points - 1 week of work; 5 points -

1+ week of work; 8 points - 2 weeks (full sprint) of work. Anything greater than an 8 would be split into sub tasks. Once a general understanding of the task was reached, we would all simultaneously use our fingers to show how many points we believed the task deserved. If we were not at an initial consensus, we would negotiate the effort and explain our reasonings for the points we assigned until a consensus was reached.

- **Burndown Charts.** The charts would help us plan our future sprints more effectively to understand and predict the amount of work we would be able to complete. This was based on the story points assigned vs completed.

- **Retrospectives**. At the end of each sprint, we had a retrospective meeting to discuss four things: What went well (continue), what went poorly (stop), what we needed to start doing (start), and general things we needed to understand (keep in mind). This was done to help make improvements in our process and workflow, and ensure that any issues didn't linger.

- **Four-up Charts.** We used this lightweight artifact as a part of our stand-ups to keep track of what we were doing, allow us to quickly familiarize ourselves with progress, and keep our sponsor informed of the project status.

Overall, we found that this method was very effective with communicating with our sponsor. Our progress and status was readily available each week, and we used the Four-Ups to guide our meetings and any demonstrations of new functionality. Additionally, our Tuesday meetings were used to clarify any major questions we had, and we used our backlog to effectively set the expectations for the project and understand as a collective what tasks were prioritized. For any small questions, we used Slack to communicate with Tom. This line of communication allowed us to not get stuck if we needed information before the next Tuesday meeting, and we were able to set up some meetings and discussions on-the-fly.

We defined five roles for each person (Jared had two - more on that later) to help us specialize our work and to avoid confusion. Below are the definitions in italics, with a brief description of the status of this role after.

- **Team Coordinator/Meeting Facilitator (Jared Jasie):** *A fully democratic team often lets things fall through cracks because no one feels the responsibility for keeping track of what the team needs to do and making sure that one or more team members are responsible for completing it. This person is also in charge of leading meetings and keeping things moving.*
    - Jared generally kept the meetings flowing and guided us through the schedule that Nikolas developed. Others stepped in at times when Jared was unavailable.

- **Sponsor Communicator (Edward Wong):** *Some sponsors may say that anyone on the team can contact him or her. Even if the sponsor says that, the team can still use a single point of contact with the sponsor. If the sponsor has a question, he or she may not know the correct team member to contact. If all questions go to one person, it is known that that team member has responsibility for making sure that an answer gets back to the sponsor in a timely fashion.*
    - Generally speaking, Edward took charge of this, but at different times we all reached out and had communication either with Tom or with Brian, the Salesforce contractor. While having one person facilitate this is good on paper, it doesn't always hold up as it can create a bottle-neck, and questions for specific tasks are better handled by the person doing that task.

- **Website Coordinator (Anthony Palumbo):** *Sponsors will often want to use the website as the mechanism for receiving project updates, weekly reports, and artifacts. Someone needs to have responsibility for getting the latest material from all the team members and doing updates on a regular basis. Some of this can be automated by detecting updates to a website module in the project's CVS or SVN repository.*
    - Anthony took charge of this as needed, but it wasn't needed as often as anticipated. He balanced this by adding the role of Source Control coordinator.

- **Meeting Scribe (Nikolas Tilley):** *Some development methodologies may emphasize short meetings where full meeting minutes may not make as much sense. From almost every meeting, no matter what the development methodology, assignments of tasks to team members, with expectations on the completion time frame, will be made. It is important that these assignments and completion commitments be captured from every meeting and posted for the entire team to see. If that is not done, it is very easy for a team member to not complete a task claiming "I did not know I had to do that." To maintain a consistent style in reporting assignments, selecting one person to be responsible for capturing team assignments works best.*
    - Nikolas did a great job with this, setting the meeting agenda, notes, and Four-Ups as needed before the meetings, and taking thorough notes during meetings to ensure that we had a good reference for any questions.

- **CICD/DevOps (Jared Jasie):** *Continuous Integration, Continuous Delivery, and automation are all important pieces of the modern development process. Being able to have things like unit and integration tests run automatically is critical to continued success, and adds time back to developers' schedules as they have fewer items to worry about and have less to do manually. The CICD/DevOps engineer will be responsible for managing the automation pipeline on Jenkins, as well as solving any issues with the tooling - such as source control or the VM. One person will be in charge of this, but all developers will be responsible for monitoring their builds and ensuring that they're passing.*
    - This was a big part of the project in the first few Fall months, but as the effort to maintain it and develop it increased and the return-on-investment decreased, it

was largely abandoned. This is because we determined that Jared's time and effort could be better spent on actual product deliverables.

Overall the roles were effective, but not without fault or issue. We all tried our best to do our jobs, but supported each other and picked up any slack as needed.

Project Schedule: Planned and Actual

Our project schedule was largely dictated by the backlog that was created during the previous team's time with the project. Initially, we had the goal of releasing a preliminary version of the software for other organizations to start using on April 1, 2020. However, that plan changed as COVID-19 started to slow our progress. Each sprint during our planning meeting with the sponsor, we would prioritize the product backlog and pull the tasks we estimated we would be able to complete into our sprint backlog. Early on in the project, we found that our estimation was not as accurate as we thought it was and we found that our sprint velocity was increasing but not as fast as we would have liked.

The first big milestone we encountered was a demo in February, where the functionality we had completed was presented during a talk hosted at RIT. Another issue we ran into that slowed development involved our contractor/consultant. His contract with the Seneca Park Zoo Society to work on the project was not renewed until February 2020, giving us only about 1-2 months to work with him until our code freeze.

System Design

Salesforce and the Lightning Web Components Framework:

Our whole front-end was established using the Lightning Web Components Framework and managed using the Salesforce platform. The previous team made the front-end to fit different views for different user roles such as donors, field researchers, and zoo administrators. The work that our team has done involved features that was mainly intended for field researchers and admins. The front-end design is mainly being handled by the project's Salesforce consultant. Our job for this part of the project was just the development, packaging, and the delivering of functional LWCs.

The benefits of using the Lightning Web Components Framework is that it allows for the creation of simple drag-and-drop web page elements. With these, people who do not have much expertise with web development can take LWCs and create full web pages of their own design. The Salesforce platform allows LWCs to be easily distributable between Salesforce orgs and accounts so anything created in a sandbox can be transferred to a production environment very quickly.

Node.js API:

In order for our LWCs to communicate and work with the data stored on our Heroku-hosted databases, an API needed to be established to facilitate this. Node.js allows for the use of JavaScript's Fetch API to create GET, POST, and PUT endpoints for our LWCs to connect to. We were able to create new endpoints in addition to the endpoints already created by the previous team to enable our LWCs to access and process ecological data.

The decision to use Node.js allows us to use JavaScript as the main programming language for both the back-end and front-end. This allows us to use the same programming language, potentially saving us from learning a new language.

Heroku:

The Heroku platform hosts our development databases as well as the production database that is planned to be used with the production version of the MyConservationLife Conservation Impact Tool. With just a Free Heroku account, we have been able to use four Heroku apps with Heroku Postgres add-ons. These add-ons provide PostgreSQL support and allow these apps to host our development and production databases.

PostGIS Database:

The Open Source Database stores its conservation data in a PostgreSQL database that has the PostGIS extension installed. PostgreSQL and PostGIS together enable geospatial capabilities required for the system. It can efficiently store latitudinal and longitudinal points and retrieve them via bounding box queries or distance queries.

The previous team had chosen this because it provides extensibility and it gives the sponsor flexibility in where the database is hosted. This is an important consideration due to the magnitude of the data that is anticipated to be stored in the database. We decided to continue

using the previous team's choice of database but there are known limitations. Our team was not successful in creating a method of scaling the database horizontally. We have concerns about the performance of the joins that are required to keep the database schema flexible.

Distributed Database:

Being able to distribute the conservation data horizontally was a task set by the initial team. Their plan was to have a central server managed by the Seneca Park Zoo Society that would allow Zoos to plug into and share their own data. Queries would be made to the central server which would be forwarded to all the participating Zoos. The result set would then be joined together and returned to the user.

Since then, it was decided that MyConservationLife was not going to be individually packaged up and sold to Zoos, but rather be marketed as a single Salesforce Org. There are also issues of performance, reliability, and security if Zoos were to be managing their own databases like this. For these reasons, we believe that this initial plan should be forgone. We still think that distributing the database horizontally is necessary, but via sharding and under a single organization's management.

Process and Product Metrics

Measurements

- **Lines of Code:** We initially planned on using lines of code as one of our measurements, however due to the nature of the repository it is not feasible to actually use this measurement. NPM generates a package-lock.json file which is generally thousands of lines of code, this greatly impacted the addition/deletion count for each contributor of the project making the numbers available on GitHub inaccurate.
- **Story Points:** We estimated the number of points for each task during our spring planning meeting, then updated the tasks in Trello with the actual value once completed. At first we found our estimation to be inaccurate, however by the second semester we were able to estimate more accurately and take on a more appropriate amount of work.
- **Tests:** When we took over the project, the code base already had a good number of unit tests that tested the LWCs. For the first semester all of the new features we implemented had 100% test coverage. For the second semester, we were more focused on completing as much functionality as possible, so we were not able to complete 100% test coverage for new features, however we did have tests for each new feature.

Metrics

- **Velocity:** We completed the second semester with a velocity of 12.08, meaning we completed on average 12.08 story points per sprint. This number was considerably lower for the beginning of the first semester as we were just starting and spending a lot of time learning, however it increased significantly for the second semester.
- **Burndown:** Our burndown numbers fluctuated for each spring ranging from 0 to 1.32 story points completed per day. This number is not that significant as a metric since we had a large range of sprints over two semesters with fluctuating amounts of work to complete in each.
- **Lines of Code per Spring:** As stated in the measurements section, we cannot accurately measure the lines of code contributed by each developer since the lines of code provided by GitHub is very inaccurate.
- **Acceptance Test Coverage:** We manually tested and demonstrated all new features with our sponsor to confirm that the functionality we created was done correctly.

Product State at Time of Delivery

As of April 28th, 2020, we have added several key pieces of functionality to MyConservationLife. While we do not consider it to be production-ready at this time, it is in a usable state and we consider it to be a quality "Proof of Concept"; it is at a point where a reasonable amount of effort on top of our work could make it production ready. Our "MVP" and scope changed a few times in the closing months of this project, which necessitated a momentum shift from the work we were doing and planning on to something different. As a result, what we originally planned on and scoped out changed in some areas. Despite this, we still feel like we were able to produce a quality product that met our sponsor's expectations.

These are the major additions the Seneca Park Zoo Tigers have added to the tool:

- **Donation Code Management.** We have added functionality in conjunction with Brian to allow for the LWCs to handle and manage donor codes. This functionality includes linking it to specific assets (such as trees), displaying the donor codes on the Salesforce site, allowing for code assignment to donors, and functionality to sort based on donor code.
- **Import/Export CSV.** This functionality is focused on researchers who would use this tool. It allows for researchers to upload their data as CSV files, as well as download prior data and template CSVs. This makes data portability better and increases accessibility, as well as creating a standard to mitigate any data conflicts or inconsistencies.
- **Geospatial Search.** The geospatial search is new functionality that allows for users to create different shapes on a map (circle, rectangle, custom polygon). Using the latitude and longitude of these shapes, the system will return all data points in the database that are enclosed by the shapes. This is important in allowing researchers to understand where their assets are in relation to assets for other projects. This can help identify relationships between projects, and create a more holistic ecological impact.
- **Temporal Search.** The temporal search allows researchers to see how different assets have changed over time; both as individuals and in relation to the environment around them.

Project Reflection

What Went Well?

Overall, our team cohesion was very good and the relationship we had with Tom was good. Being able to have interpersonal connections allowed us to communicate effectively, and share information as needed. As a team, we felt comfortable relying on each other for help and reaching out if we had questions. This helped us overcome a lot of the roadblocks and issues we faced.

In addition to being cohesive, we were also flexible in the face of changing requirements and scope. As Tom requested many times, we tried to stay flexible both with our code and as individuals. Part-way into the Spring semester, our MVP and what we had planned as our backlog changed. We handled the change in stride and quickly pivoted to understanding, assigning, and completing the new tasks as fast as possible. We were able to complete the MVP to the sponsor's satisfaction, and possibly a bit more.

What Went Poorly?

Our communication and initiative with the Salesforce consultant could have been better. The lack of communication was compounded by the difficulty to meet due to time zones and scheduling conflicts. As a result, our work with him took some time to ramp up and get a better understanding of what he needed.

Our ramp-up times also took significantly longer than we anticipated or wanted as none of us had any experience with the tools we were using. This put us behind schedule from the start, and made any work going forward more difficult.

What Would We Do Differently?

One of the major things we'd do differently is "reverse-engineer" our cards and tasks. Our methodology for selecting our tasks wasn't good at looking at the big picture. The biggest component for choosing was "what's next on the backlog?", rather than viewing it holistically and working backwards after understanding what our larger task and purpose was.

We'd also communicate with the consultant more effectively and earlier. If we had access to him earlier in the project, we believe that we would have been able to produce a better product and more items if we had the support and "vision" that we got later.