Oregon State University

Beaver Botanica
Midterm Project Report

Adison Daggett, Kathryn Butler, Luke Scovel, William Brennan,
Anshu Avinash, Finlay Curtiss, Jake Thompson

Software Engineering 1

Manish Motwani

February 10, 2025

Table of Contents

Abstract

The campus of OSU is home to a wide diversity of plants contributed by talented botanists over the years. It is hard for new botany students to know where all these plants are. They are left with little option save to ask for help from professors or explore the campus themselves. Our goal is to help these botany students by providing them with an online map that can tell them where plants lie on OSU's campus.

This project is a plant identification app for the Oregon State University campus. Users will take pictures of plants around campus, identify them, and upload them to a website that will show a map of campus with the locations of each identified plant. There will be layers to the map for each season, so users can see what plants are in bloom at any given time. Each plant, when clicked on, will show details of the plant that the user provides.

The Problem with Current Software

There are three major problems that plague botanical software. These are poor design, broad location size, and overly scientific nomenclature. For example, while Oregon State University does offer plant identification software, it encapsulates the entire state of Oregon and only uses the scientific names of these plants. A student with a passing interest in botany would not be able to understand what they are looking for without having to do their research. Educational resources deserve to be easy to use and understandable to someone uneducated on the subject.

Our Solution - Beaver Botanica

Beaver Botanica is a plant identification software by students of Oregon State, for the students of Oregon State. Unlike most identification software, Beaver Botanica only encapsulates the Oregon State University campus. Our software will be a web app based on HTML, CSS, and Javascript using the React.JS library. The backend of our software will be built in MySQL and Javascript using the Node.js and Express libraries. We have chosen all these languages mainly because everyone on the team has experience with them and can therefore competently make their changes or give their feedback if need be. The goal of Beaver Botanica is to be a community-driven plant identification app that is user-friendly and allows people of all backgrounds to engage with botany.

Testing Our Solution

We'll be testing our solution in three areas: User Experience, Performance, and Reliability. In terms of user experience, we want our website to be intuitive for all users, regardless of their botany knowledge. To test this, we'll have students and faculty use the platform and provide feedback. This will help us create an engaging platform that makes finding and identifying plants easy and enjoyable.

Since Beaver Botanica will be used while walking around campus, it needs to be fast and responsive. We'll conduct real-world testing around OSU's campus to check how well the site functions under different network conditions. This will meet our goal of creating a fast learning website that students can use on the go.

Plant identification is a core function of our platform, so we'll test the reliability of our platform by determining whether users can correctly identify and label plants using our system. This includes comparing user-submitted data to verified sources, and adjusting our search features to improve accuracy. This will help make Beaver Botanica a reliable resource for learning and discovery.

To test for reliability, we need a reliable dataset. OSU's Department of Horticulture has a database of over 1,800 landscape plants, which will provide a strong foundation of verified plant species and will ensure that our database starts off with accurate information. Since Beaver Botanica is a community-driven platform, users will play a key role in expanding our dataset. Students and faculty will be able to identify plants, share their observations, and contribute new data points. Over time, this will help build a dynamic and ever-growing map of OSU's plant life.

To measure the success of Beaver Botanica, we will evaluate two metrics: identification accuracy and user experience. Since plant identification is at the core of our platform, we need to ensure that the information users submit is reliable. We will compare user-submitted plant identifications with verified sources from the OSU Department of Horticulture. This will help us assess how well the platform guides users in correctly identifying plants.

Beyond being functional, Beaver Botanica should be engaging and easy to use. To measure this, we will gather feedback through user surveys and track engagement metrics, such as how often users return to the platform and contribute new plant observations. By analyzing these metrics, we can ensure Beaver Botanica makes learning about plants on campus an enjoyable experience.

Software Architecture

Components

Beaver Botanica will be a website with the intention of it being used on mobile devices as users will likely want to use it while walking around on campus. The three main components of the software are the client-side web page, the server-side web page, and the database.

The client-side web page will be the users' method of interacting with the data stored in our database, allowing them to view, edit, and add entries that can be viewed by others.

The data will be displayed in a user-friendly format that is abstracted away from the actual database and server architecture. User input that will query the database could be search criteria or selecting a plant.

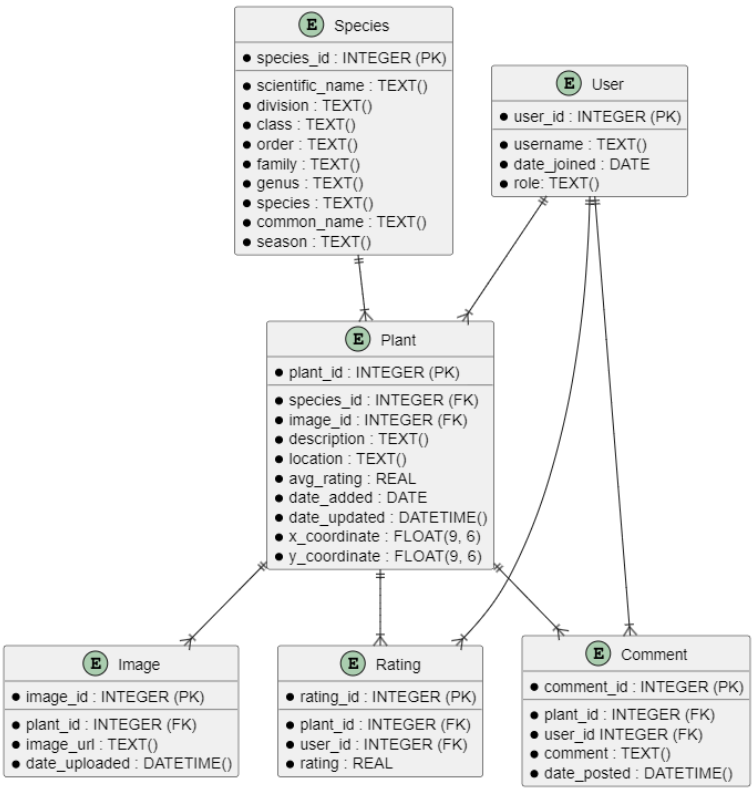
The website's server will be hosted on a personal OSU webpage, which will simplify the networking requirements from our team to ensure any user can access it at any time. The server will interface with the database by taking a user's input, converting it to an SQL query, querying the database, then sending the necessary data to the user's browser, which will then display the data through the website.

The database will be stored on a Bluehost server that can be accessed by the server through queries. More information on the database's structure and design is in the Data section below.

The website serves as an interface with the server, which in turn interfaces with the database. The website will serve as the main method of interacting with both of the other components, as it has a graphical interface that allows editing and viewing data from the database. The server's interface with the website and database is not graphical, but rather JS queries with the website and SQL queries with the database.

Data

The Beaver Botanica database will store information about individual plants, taxonomies of plants, images of plants, user profiles, post ratings, and comments. The schema below contains a detailed description of the data we will store:



Schema of the Beaver Botanica Database

The above database schema describes all of the data we will store for our platform. This structure will ensure that all plant data and user interactions are organized efficiently.

Assumptions

We will assume that our architecture will later be built upon and updated in the future. We will also assume that our users will not break any student or user conduct rules and that they will correctly submit plant entries. While the rating system we are implementing can be used to determine the accuracy of plant entries, we don't have the resources to moderate posts and comments, or to create an automatic content moderation system.

We will assume that the number of users on our site will be small, as botany is a niche interest. While testing and developing, there will likely be a maximum of 10 users. Attempting to make the software handle hundreds or even thousands of users well would provide additional complications.

Decision 1: Static Image or Mapping Software API

For displaying the locations of plants on campus, we had to decide between using a static image to represent a map of campus or using a mapping software's API.

Static image

Pros	Cons
• Simpler to implement	• Lacks some features • More complicated to determine geographic coordinates

Mapping Software API

Pros	Cons
• More features (navigation, coordinates, map layers, scalability)	• Much more complicated to implement • Most features are unnecessary

In the end, we decided to use a static image so we could focus our time on developing and documenting other parts of our software, rather than learning how to implement an API. In addition, many of the features of using mapping software aren't necessary, so the extra work wouldn't provide much benefit.

Decision 2: Website or Mobile App

For the implementation of our software, we decided between creating a website or a mobile app.

Website

Pros	Cons
• Team has more experience • Works cross-platform	• May require additional testing on different devices

Mobile App

Pros	Cons

• Optimized for mobile devices	• Team has limited experience • More work to port to other devices and operating systems

The pros of a website largely outweighed any cons, so we decided to build Beaver Botanica as a website. Being able to use our previous experience with developing websites helps us focus on implementing the features we want, while the cross platform functionality reduces work during implementation and testing.

Software Design

Client-Side Web Page

The interface will be developed using React.js with HTML and CSS to create an interactive, mobile-friendly platform. The UI will consist of various reusable components, including a navigation bar, forms for plant searches and submissions, and a map component that displays plant data. The navigation bar will allow for easy navigation around the website. The form components will allow users to quickly search for plants and input plant data effectively. The map component will display plants dynamically and allow for users to locate the plants around campus. These components will work together to make plant identification accessible and engaging for users.

Server-Side Web Page

The server-side of the application will be written using JavaScript through the express.js and node.js libraries. These server-side JavaScript libraries will handle incoming requests from the client by performing database operations and handling authentication. These libraries will decide what gets served to whom. When an event is sent to the server, the listener will route the operation to the correct JS function to handle whatever internal processing needs to happen.

Database

We will be using MySQL to set up and run the database. The database will be hosted on an external Bluehost domain and managed through the MySQL Workbench application. It consists of the following six tables:

- Users: Stores user information such as user ID, username, date joined, and role.
- Species: Stores plant species information including species ID, common name, scientific name, and taxonomic classification.
    - One-to-Many relationship with Plants
- Plants: Stores individual plant instances with details like plant ID, species ID, image ID, description, location, and coordinates.
    - Many-to-One relationship with Users

- Images: Stores images of plants with image ID, plant ID, image URL, and date uploaded.
    - Many-to-One relationship with Plants
- Comments: Stores user comments on plants with comment ID, plant ID, user ID, comment text, and date posted.
    - Many-to-One relationship with Plants
    - Many-to-One relationship with Users
- Ratings: Stores user ratings for plants with rating ID, plant ID, user ID, and rating value.
    - Many-to-One relationship with Plants
    - Many-to-One relationship with Users

The SQL server will be responsible for storing, receiving, and delivering data to and from the server when prompted through SQL CRUD operations.

Coding Guideline

SQL

We decided on the Mozilla Style Guide for two reasons. The main reason is for the sake of readability. The second reason is that it is similar in style to how our backend developers write SQL. Because of that, the adjustment to the style guide would be minimal. We hope to enforce the usage of the style guide by reading over each other's work to make sure that the guide is being used. If it is not being used in some way, then we can edit it or ask the contributor to edit it themselves.

Javascript

We decided to choose the Google Javascript Style for two reasons. The first reason was a lack of adjustment for the developers as we already program Javascript in a similar style. The second reason is for the sake of readability for those who wish to read over the code at a later date. We plan on enforcing the style guide by reading each other's code and making adjustments when needed if the other person forgets a rule.

HTML/CSS

We chose the Google Style Guidelines because the majority of us already use a similar style of programming in HTML and CSS. The lack of adjustment will make abiding by these guidelines easier for the front-end developers. We plan on enforcing the style guide by reading each other's code and making adjustments when needed if the other person forgets a rule.

Process Description (Risk Assessment)

The following are the top five risks to the successful completion of our project:

1. Scalability

One of the largest risks with developing Beaver Botanica is in making sure the system is scalable. As a seven-person team, we will not be able to map out all the plants spanning the whole OSU campus. Our solution is to allow users of OSU Plant Map to upload pictures and information to the database.

The likelihood of this occurring is high, and the impact if this occurs is also high. We know this from our research into Oregon State University's existing Department of Horticulture website. This website documents the over 1800 landscape plant species found across Oregon, which is a feat much larger than the scope of our project. Additionally, OSU's campus is extensive, which makes it impractical for a small team to catalog all plant life. Giving users the ability to add plant instances is crucial to the success of our project, but comes with its own unique challenges. User-generated plant findings can provide broad campus coverage but can introduce issues like data inconsistency and quality control.

The following provides a list of steps we will take to increase the scalability of our project:

- Implement a verification system where plant uploads are reviewed by moderators or verified users before being added to the main database.
- Allow community-driven validation by providing users with the ability to flag plant inaccuracies.
- Conduct initial research to determine the species of common plants on OSU's campus.

Our plan to detect this problem is to monitor submission frequency and quality using automated scripts and fact-checking. We can implement automated scripts that can detect duplicate submissions, and we may be able to determine if a submission is bad based on user input. For instance, if a user inputs a plant that is known to be under 2 feet tall at full growth, but labels it as 14 feet tall. We can also implement community feedback mechanisms, such as upvoting correct entries.

If this problem of too much-unverified plant data occurs, our mitigation plan is to introduce a system that only allows confirmed, trusted users to add plant data to the system. Additionally, we could create a small tutorial to guide users through the correct usage of the website.

Since submitting our Requirements document, we expanded our original paragraph to include reasoning for this problem, as well as a detection and mitigation plan. We also introduced a concrete plan in the case of too much incorrect information.

2. Data Accuracy

Ensuring the accuracy of plant data uploaded by users is a significant risk. Incorrect or misleading information could reduce the app's reliability and usefulness. To mitigate this, we will implement a verification process for user-submitted data.

The likelihood of this occurring is high, and the impact if this occurs is also high. We have gathered from experience and research that crowdsourced data collection often introduces inconsistencies, as seen in similar user-generated content platforms. Because our target audience is the student and faculty population at Oregon State University, users may not be familiar with similar species of plants and may misidentify them due to a general lack of botanical knowledge.

The following are steps to improve data accuracy:

- Require users to provide images, location, and detailed descriptions for verification.
- Allow experienced users or moderators to review and approve submissions before they are made public.
- Provide educational resources to help users correctly identify plants.

To detect potential data inaccuracies, we will monitor flagged entries for potential errors, and track submission patterns to identify users with consistently incorrect inputs. Our mitigation plan to safeguard data accuracy is similar to that of ensuring scalability. Should too many incorrect entries occur, we can implement a trust system where known users can input new plants, and the general population can view and comment.

This section has been improved since we submitted our Requirements document. We added a mitigation and detection plan and introduced a verification system for trusted users.

3. User Engagement

Another risk is the potential lack of user engagement. If users do not participate in uploading and verifying plant data, the app will not reach its full potential. We will address this by incorporating peer-to-peer interaction elements to encourage use.

We believe that the likelihood of this occurring is medium, while the impact it will have if it occurs is high. Initial feedback from users is mixed, with some claiming that our project “I think you guys have a really good idea moving forward! I think it would be really helpful to have some sort of help for those with not much experience.” Others state, “I feel like your project is too niche, with a potentially large maintenance cost.

More specifically, I can rarely think of anyone who may be interested in researching flora and trees around." Some users are enthusiastic about learning more about OSU's campus botany, while others are not.

To reduce this impact, we have proposed several steps:

- Implement a rewards system, such as badges or points for verified contributions.
- Enable social features, such as user profile viewing and leaderboards.
- Introduce periodic challenges or community events to maintain interest.

To detect a problem with user engagement, we will conduct surveys and user feedback sessions to identify barriers to engagement. We can also monitor the rate of new user signups and repeated usage. If this occurs, our plan to mitigate it is to increase promotional efforts through campus partnerships. This may include posters with a link to the website on university-approved billboards, adding a post in the campus newsletter, and submitting our website to the Department of Horticulture. We may also simplify the contribution process if it is a challenge listed in the reasons why users are not engaged with our website.

This section has been altered from the Requirements document. We improved the detection plan and added additional details to mitigate user disengagement.

4. Technical Challenges

Developing this application involves overcoming various challenges such as integrating the Google Maps API, ensuring cross-platform compatibility, and maintaining a responsive design. To mitigate this, we will allocate time for testing and debugging and seek assistance if needed.

The likelihood of this challenge affecting us is high, and the impact it has is also high. Based on our initial research, integrating third-party APIs like Google Maps often introduces unforeseen complexity, and tends to introduce compatibility and performance issues. Ensuring our app works both in a mobile and website environment requires extensive testing, which will be difficult to conduct in a small time frame.

The following are steps we will take to minimize technical challenges:

- Conduct thorough research and prototyping before fully integrating the API.
- Schedule regular debugging sessions and maintain clear documentation.

To detect an issue with integrating the API, we can use logging and error-tracking tools. We can also conduct usability testing on various devices to ensure compatibility across

platforms. If we have issues with integration, we can allocate additional time and resources for debugging and adjustments. We can also seek assistance from OSU faculty or online developer communities if problems persist.

We have added to this section extensively since our last submission of the Requirements document. We initially acknowledged technical risks but have now outlined specific detection and mitigation strategies. We also added a focus on automated performance testing to improve early issue detection.

5. Data Security and Privacy

Because users will be submitting plant data, including images and descriptions, we must make sure their data is secure and private. This is a major risk because mishandling user data could lead to privacy concerns, loss of trust, and potential violations of OSU's data policies. Additionally, if user-uploaded images or location data are exposed without proper security measures, there's a risk of unauthorized access or misuse.

The likelihood of a true data breach is relatively low, while the impact it has is high. From our research, we've found that many crowdsourced platforms have experienced data breaches. This is especially true when the data is not encrypted or secured in some way. We also know from experience that storing images and geolocation data is considered pseudo-sensitive information, and introduces potential vulnerabilities that can be exploited. While we do not encourage our users to post personally relevant information, there is a chance they might.

In order to reduce the likelihood of a data breach occurring, we can follow these steps:

- Implement secure authentication to prevent unauthorized access.
- Store only necessary data and avoid collecting personally identifiable information.
- Use HTTPS for all connections to protect data transmission.
- Regularly review and update security protocols based on OSU's data security guidelines.

In order to detect unauthorized access, we can monitor access logs for suspicious activity, such as multiple unauthorized data access attempts. To detect and limit the likelihood of a data breach, we can conduct periodic security checks and vulnerability assessments. We can also ask users to use a different password for our website than they normally do, to ensure that if there is a data breach, users will not have their "regular" passwords compromised. If a vulnerability is detected, we can mitigate it by patching the affected components and updating security measures. In the event of a data breach, we will notify affected users and follow OSU's data protection response protocols.

This section has been created since the submission of the Requirements document. We previously hadn't considered that a data breach or security was an issue, but after careful consideration, we decided that even though our application will not need sensitive information, data security is an important priority.

Project Schedule

Week - Milestone		
Tasks	Dependencies	Effort
Week 3 - Planning and Foundation		
Finalize requirements, assign roles, set up infrastructure	None	1 week
Design database schema	None	1 week
Set up development environment	None	1 week
Wireframe UI	None	1 week
Week 4 - Map and Database		
Develop database for plant data	Database schema	1 week
Refine wireframes	Initial wireframes	1 week
Implement basic campus map	Wireframes	1 week
Integrate database with map	Map implementation, Database	1 week
Week 5 - Plant Identification and Upload		
Store plant photos in database	Database setup	1 week
Create UI for adding plants	Wireframes	1 week
Implement plant upload & identification	UI, Database	1 week
Week 6 - User Authentication and Social Features		
Implement user authentication	Database setup	1 week
Create login & account forms	UI, Authentication	1 week
Implement social features (liking,	Authentication	1 week

commenting)		
Week 7 - Advanced Map Features		
Enable filtering and search	Database setup	1 week
Implement advanced search UI	Wireframes	1 week
Integrate API for filters	Map and Database	1 week
Week 8 - Testing and Refinement		
Conduct user testing	Core features complete	1 week
Fix bugs and usability issues	Testing feedback	1 week
Improve UI based on feedback	Testing feedback	1 week
Week 9 - Finalizing Additional Features		
Implement non-functional features	Core features complete	1 week
Continue UI improvements	Testing feedback	1 week
Test non-functional features	Feature completion	1 week
Week 10 - Final Testing and Launch		
Final bug fixes	All prior development	1 week
Final documentation	Feature completion	1 week
Final presentation preparation	Feature completion	1 week
Deploy mobile website	All development & testing complete	1 week

Team Structure

Adison Daggett - Front-End Designer and Developer

The role of the front-end designer has two parts: designing the front end and implementing it. This role allows for consistent design and implementation of an agreed-upon front end for the software.

Kathryn Butler - Technical Documentation and Developer

Technical documentation is an incredibly important role in software engineering. Our team needs this designated role to clarify the process for adding and verifying plants, detail the system's database integration, and outline user features such as the search

and map functionality. Kathryn is suited for this role due to her strong communication skills developed from her leadership roles in film production and peer tutoring. She also structures documentation effectively, making it easy for others to navigate and use as a reference.

Luke Scovel - Team Coordinator and Developer

For a team to function, it has to be a collaborative effort with everyone on the same track. To disseminate important information and reiterate what tasks need to be done without multiple voices clamoring over each other, a coordinator is needed. More developers are needed to bounce ideas off of each other and do the grunt work of coding.

William Brennan - Developer

A developer is someone who designs and writes the code to be implemented in the software. This role is needed for this project because we need someone to design and program the database and logic controller for the software. William Brennan is suited for this role because he is a computer science student and he has taken courses in web development.

Anshu Avinash - Front-End Designer

By bridging the gap between functionality and user experience, a front-end designer makes sure that the user interface's implementation and design meet the needs of both users and business objectives. Also, it streamlines the handoff between design and coding, improving productivity for the entire team.

Finlay Curtiss - Developer and Technical Documentation

Multiple developers are needed to ensure that all features are implemented properly and in a reasonable time frame. Finlay has experience with developing and can contribute their knowledge and perspective to the technical implementation of the software. Technical documentation is also necessary, for the development team, users, and any developers who wish to work on this software in the future. Finlay has experience with technical writing from their classes and job.

Jake Thompson - Developer

We need many developers due to the timeline of the project and the scope of what is needed to be implemented. Without developers nothing gets made. Jake has experience with developing database design and other skills needed for this project.

Test Plan & Bugs

Unit Testing

We plan to use unit testing to verify that individual components are working in isolation. Our scope is backend functions, frontend components, and plant identification and data upload features. We plan on using Jest and the React testing Library for

frontend unit testing. If we decide to create our backend code using Python, we will likely use Pytest to conduct backend testing.

System Testing

Our system testing will verify that each component works together correctly. This includes database interaction with the front end, Google Maps API integration, and user authentication and sessions. We will likely use the free version of Postman API testing to validate our API usage. We will also try out Cypress to conduct end-to-end testing and frontend-backend interactions. We will also test how the website handles large-scale plant data uploads.

Usability Testing

We will conduct usability testing to ensure our website is both informative and engaging. This includes UI/UX design, accessibility compliance, and ease of use. An easy way to test if a user enjoys our website is to conduct beta testing with students and faculty at OSU. We also can use Google Lighthouse to test performance and accessibility. From this testing, we can ensure the website works smoothly on desktop and mobile, and identify any confusing UI elements that need to be fixed. For each of these tests, we will use GitHub to log, track, and resolve bugs.

Documentation Plan

User Documentation

The purpose of user documentation is to teach users how to navigate through the website effectively. This includes a User Guide, either in PDF or as a web version. It will include:

- How to register, log in, and manage an account.
- How to search for plants and filter by location, season, or type.
- How to add, edit, and verify plant information.
- How to upload and identify plants.
- How to use social features (liking, flagging, commenting).
- We will also implement a small help guide within the website itself. It will contain contextual help within the site, tooltips that explain features, and an FAQ section explaining common issues.

Admin Documentation

Admin documentation is important to include to assist administrators in managing data, moderating content, and overseeing system security. We will provide a PDF or website version of an Admin Guide, which will include:

- How to verify and moderate user submissions.

- How to handle flagged content.
- Managing user roles and permissions.
- How to monitor server performance and logs.
- How to troubleshoot common system issues.

Developer Documentation

We plan to include developer documentation so future developers can maintain and expand the website. This includes a Developer Guide, which will include:

- API Documentation - expected inputs/outputs, and authentication details.
- Example API calls and responses.
- Project architecture overview.
- Code structure and key files/modules.
- How to set up the development environment.
- Database schema and relationships.
- How to run unit and integration tests.
- Common debugging techniques.

Requirements

Functional Requirements

Searching for a Plant - Adison

Actors: An OSU Student
Goal: To search for a plant
Triggers: The user can recognize the plant based on the image displayed after filtering the results.
Preconditions: the user searches for the plant by sorting through media results.
Step: The user sorts the media by taxonomic classification.
Postconditions: The plant they are looking for will be displayed with a detailed description of what it is.
Extensions: The user is unable to find the plant they are looking for, but is able to change that by registering it into the software.
Exceptions: The plant does not show up because it has yet to be cataloged.

Adding a Known Plant to the Database - Kathryn

Actors: An OSU student or faculty member.
Goal: To add a plant instance to the database. The plant type is already in the database.
Trigger: The user fills out all plant information and clicks the “submit” button.
Preconditions: the user searches for the plant by sorting through media results.
Steps:

- The user begins to add the plant, through clicking on an “Add plant” button.
- The user inputs the common name and scientific name of the plant.

- The user is prompted to take a picture of the plant.
- The user is asked to give a location on a map where this plant was found.

Postconditions: The plant instance is added to the database with a label of "unverified" or a similar marking to avoid mislabeling plants. The map is updated to add the plant instance.
Extensions: The user could decline to take a picture of the plant, which could still go though as a successfully added plant instance. The user might also stop in the middle of adding a plant, in which case the software would either save the progress or discard it.
Exceptions: The user may not find the plant they're looking for, in which case they would be directed to a form requesting the addition of a new plant.

Commenting on a Post - Luke

Actors: A OSU Student
Triggers: The user fills out a comment box with text (maybe with images?) and clicks submit.
Preconditions: The user is logged in to the website and on the map page. At least one post has been made.
Steps:

- The user clicks on the desired plant post on the map
- The system displays the plant details page
- The user clicks on the comment button
- The user enters text into the comment box
- The user clicks the submit button
- The system refreshes the page and displays the comment at the end of the comment chain for all users.

Postconditions: The comment is now displayed for all users on the post.
Extensions:

- The user can cancel their comment by clicking on the cancel button or by navigating away from the page.
- The user times out
- The user saves the comment as a draft
- The user is an administrator and pins the comment

Exceptions:

- The text contains invalid characters
- The text contains a spam link
- No text is provided
- An image is the wrong format
- The post is deleted before the comment is posted

Verifying User-Added Plant Entries - William

Actors: Beaver Botanica moderator

Trigger: The user clicks on the verify button on a plant post.
Preconditions: A plant entry with the unverified property is in the database.
Steps:

- The user logs in using their moderator account.
- The user clicks on one of the plants on the map.
- The user can scroll through the description of the plant to check its attributes.
- If the plant is unverified the user clicks on a verify button to verify it.

Postconditions: The plant entry that had the unverified property gets the verified property.
Extensions: In the case that a moderator clicks on a plant entry and does not want to verify it, they can remove the plant entry by clicking on the delete post button. This will remove the plant entry from the database and map.
Exceptions: In the event that clicking on a verify button does not add the verified property to a plant, the user will receive a message notifying them that the plant was unsuccessfully verified. If a plant was already verified then clicking on the verify button will not change the property of the plant, and the user will receive a message notifying them the plant was already verified.

Editing a Plant's Details - Anshu

Actors: OSU Student or faculty member. Goal: To edit the details of the plant entry in the database. Trigger: On the plant's detail page, the user selects the "Edit Plant" button.
Preconditions: The user might be the original submitter and has the authority to change the plant data.
Steps:

- The user accesses the information page of the plant.
- The user selects the "Edit Plant" button.
- The system shows a form with the current plant information already filled in (e.g., common name, scientific name, description, location, image).
- The user changes the desired fields, like adding a new image or changing the plant's name and description.
- User clicks on the "save change" button to confirm the edits.

Postconditions: New plant details have been added to the data. The plant details page reflects the uploaded information.
Extensions: By using the "cancel" option, the user can edit, remove the modifications, and return to the plant's information page. If a user uploads an image in a format that isn't supported, the system will prompt them to upload an image in a format that is accepted such as JPEG or PNG. Exceptions: When a user leaves a mandatory field empty, the system indicates an error and asks them to fill it up. When the user enters an incorrect value, such as a special character in the plant name, then the system verifies the information and displays an error notification.

Navigating to a Plant's Location - Finlay

Actors: A user of the Beaver Botanica app.
Goal: The user should be able to find the locations of a certain plant on the OSU campus and get directions to the plant's location.
Triggers: The user is viewing a specific plant after searching and then presses the button that opens the map view. The user then selects a location and chooses to get directions to it.
Preconditions: At least one plant is in the database and that plant has at least one location placed on the map.
Steps:

- Selects a plant after entering a search query
- Open the map view for that plant to view all locations of the plant
- Select one location marker and click the "Get Directions" button
- The application will then display directions to reach the selected location

Postconditions: The application displays a map view with any locations for the plant displayed as markers.
Extensions: The user might not choose to receive directions to the plant's location, instead navigating by themselves.
Exceptions: A plant might not have a location associated with it, in which case the map would not display any locations and the user would not be able to get directions. The mapping service we use might also be down, which would cause no map to be displayed to the user, meaning they couldn't get directions.

Filtering the Map - Jake

Actors: An OSU Student
Goal: To view the locations of all plants in a specific Genus that are in season
Triggers: The user clicks the filter button
Preconditions: The user is on the Map page and has a specific Genus in mind they want to see locations for plants that are in season.
Postconditions: The map shows the locations of all plants that match the given filter.
Steps:

- The user clicks the filter button
- The system displays the filter sidebar (this will hide the add or Info sidebars)
- The user either selects the desired Genus from the Genus dropdown or starts typing in the dropdown to find it faster (for got what this input type is actually called)
    - The system should then fill higher level taxonomy filters
- The user checks the box in season
- The System Filters the map based on the set filters

Extensions/variations:

- The user changes their mind and wants to clear the filters this can be done with the clear all button
- The user decides they actually wanted a different genus in the same family
    - Now they just select the genus dropdown and it should only show genus in that family
    - Should also clear all taxonomy filters below genus.
- The user decides they actually wanted to see the whole Family
    - The user clicks the clear button for Genus

Exceptions:

- The user can't find the desidered genus in the drop down
    - This should mean that there are no plants in that genus in the database.
    - Maybe suggest Advanced Search as the result might be misfiled.
- There is nothing on the map
    - This means that no plants of that genus are in season.
    - Should display a message that no plants are in season for a given filter and recommend turning off the in season filter.

Non-Functional Requirements

Dark Mode and High-Contrast Mode

Both modes improve user accessibility by providing alternative color schemes, such as dark mode for low-light environments and high-contrast mode for users with visual impairments or color blindness. They must load without impacting software speed, and may automatically adjust based on the user's device settings. The user must be able to activate dark or high-contrast mode in no more than four clicks. The colors used in high-contrast mode must be tested to comply with industry standards.

Expandability

The software should be able to store and display all plants on the OSU campus (2000+) and also allow them to be searched. With the software allowing more plants to be added, it enables the users to add plants that aren't already in the system so they can all be catalogued.

Incorrect/Malicious Entry Prevention

The software should prevent users from creating incorrect or malicious entries for plants and/or their locations. It should be built in a way that totally prevents these entries being displayed as correct information or removes it from the database and software quickly.

External Requirements

Error-Free

Beaver Botanica must prevent common errors and crashes from occurring. The website should provide clear messages and suggestions when handling cases such as no search results. Invalid inputs, including non-text characters in search fields or unsupported image formats, must be sanitized or blocked before processing. Additionally, failures in data submission or retrieval should be detected, with proper error messages and retry methods available.

Installable

Since users will be actively exploring OSU's campus, Beaver Botanica must be fully mobile-friendly. The website must be responsive, and must adapt to different screen sizes and touch interactions. The publicly accessible URL should allow users to open the platform easily without requiring special installations. Features such as location-based search and interactive map navigation should be optimized for mobile devices.

Buildable

The software should have up-to-date documentation that allows other developers to help developers set up, modify, and expand the project to other college campuses or locations. The software should also allow developers to customize their version of the application by using a different map, different plants, or other changes to features.

Scope

Beaver Botanica must be robust, reliable, and bug-free. It should provide all essential features outlined in the functional and nonfunctional requirement sections. This includes plant identification, user-generated submissions, an interactive map, and community engagement tools such as comments and ratings.