Final Project Report

HerbVita

Wenhan Miao 301388465

Mark Soh 301404127

Table of Contents

Medicinal Herbal Web App (HerbVita)

Executive Summary

Develop a web application that serves as an educational resource on natural remedies. The site will provide detailed information about each herb (e.g., Ashwagandha, Lemon Balm) including its benefits, possible side effects, and recommended usage. For registered members, the site will allow them to personalize their experience by saving favorites which customize their homepage as well as providing feedback on the herbs by leaving comments. The goal is to inform users, rather than to sell products.

Project Proposal

This project outlines the development of a web application featuring a comprehensive database of these remedies, each with detailed information on its attributes, including benefits, potential side effects, and recommended usage.

HerbVita will cater to both non-registered visitors, who can browse and filter the extensive content, and registered members, who will gain access to personalized features. These features include the ability to save favorite remedies for a customized homepage experience and the option to contribute to a community by posting comments on remedy detail pages. User registration will be straightforward, and secure authentication will be implemented.

The application will be built using PHP for the backend and MySQL for data storage, leveraging AJAX for dynamic filtering and updates. The database will be structured across user, remedy, and potentially comment/favorite tables to efficiently manage content and user interactions. The core objective of HerbVita is to provide reliable and objective information to users, empowering them with knowledge about natural remedies without promoting any commercial products.

Data Requirements Analysis

Data Sources:

The primary data sources for HerbVita are gathered from credible online databases, such as MedlinePlus (medlineplus.gov), where it has information such as its usage and has other references to back up its claims.

Herb Attributes:

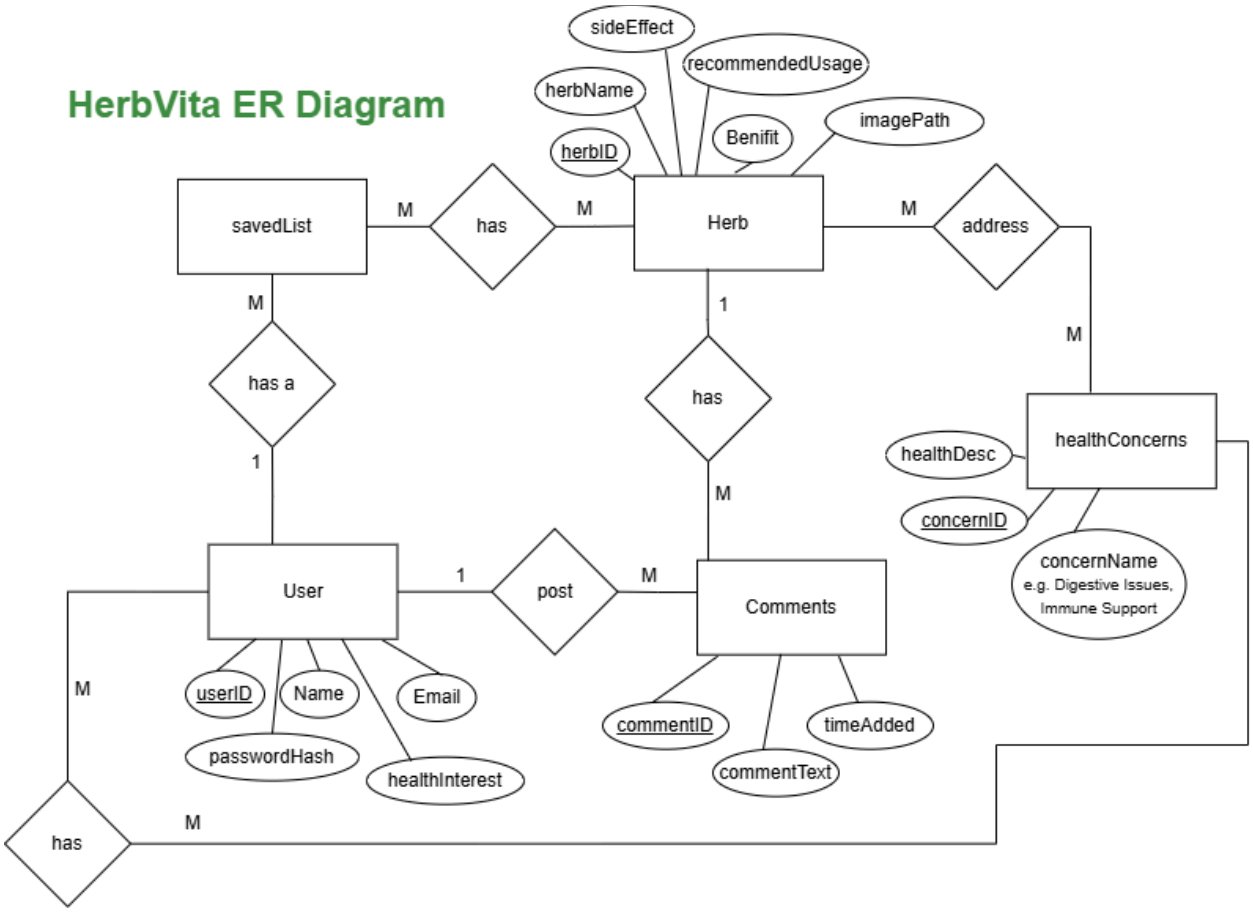
To fulfill our core purpose of informing users effectively, we identified key attributes necessary for the herb data. The herb table will include herb name, sideEffect, recommended usage, benefit, and the health concerns it addresses. To ensure comprehensive and engaging content, we determined to feature eight common health concerns, each associated with six relevant herbs to provide sufficient information when users filter by these concerns. This foundational data was initially compiled manually in a spreadsheet before being imported into phpMyAdmin.

Database Key Entities:

The core entities within the HerbVita application are:

- ★ Herb: Represents individual natural remedies.
- ★ Health Concerns: Represents the various health issues or symptoms that herbs/remedies may address (e.g., Anxiety, Insomnia, Inflammation).
- ★ User: Represents individuals interacting with the application.
- ★ Comments: Represents user-generated feedback on specific herbs/remedies.
- ★ SavedList (Member-Specific): Represents the saved herbs/remedies for registered members.

ER Diagram



Implementation Decision/Database structure

For the final iteration of our database, we have shrunk it to 5 tables instead of 6 by getting rid of the featureHerb table. Our database now consists of Comments, healthConcerns, Herb, savedList and User.

	Table	Action	Rows	Type	Collation	Size	Overhead
☐	comments	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	healthconcerns	Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	herb	Browse Structure Search Insert Empty Drop	48	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	savedlist	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	user	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	16.0 KiB	-
	5 tables	Sum	64	InnoDB	utf8mb4_general_ci	160.0 KiB	0 B

Check all With selected:

Comments Table

Registered members can utilize the **comments table** to leave feedback and remarks on individual herbs. This table stores the comment text, links it to the userID of the commenting member and the relevant herbID, and records the timeAdded. This structure ensures that only authenticated members can contribute to the community discussion.

ientID	herbID	userID	commentText	timeAdded
19	18	adminMark	this flower is really good in helping me sleep. ha...	2025-04-08 12:07:05
20	18	admin2	nice i also think is great for me to especially af...	2025-04-08 12:09:13
21	9	adminWenhan	My mom like this herb	2025-04-08 21:30:06
22	41	adminWenhan	Quite spicy on its own, I would try the extract ne...	2025-04-08 21:31:07
23	37	adminWenhan	This is actually so good	2025-04-08 21:31:24
24	32	adminWenhan	I use products with lavender essential oils, smell...	2025-04-08 21:32:11
25	19	adminWenhan	Not a fan, it dyed my fingers yellow and tasted te...	2025-04-08 21:34:20
26	18	adminWenhan	I use it as an indoor home decoration, never tried...	2025-04-08 21:37:47
27	8	adminWenhan	taste good	2025-04-09 12:53:20
28	6	adminEva	This is quite nice	2025-04-09 15:57:47
29	26	adminEva	this smells really good in cream	2025-04-09 15:58:24
30	25	adminEva	I use this after a sunburn	2025-04-09 15:58:53
31	25	Wenhan123	i liket this	2025-04-09 16:09:07

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	commentID	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
2	herbID	int(11)			No	*None*			Change Drop More
3	userID	varchar(11)	utf8mb4_general_ci		No	*None*			Change Drop More
4	commentText	text	utf8mb4_general_ci		No	*None*			Change Drop More
5	timeAdded	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More

To guarantee the uniqueness of each comment, commentID is designated as the primary key with the auto_increment attribute for automatic unique ID assignment. The herbID and userID fields are implemented as foreign keys to link each comment back to the specific herb it refers to and the member who posted it, respectively.

HealthConcerns Table

concernID	concernName	healthDesc
1	Digestive Issues	Support a healthy gut with natural herbal remedies...
2	Immune Support	Boost your body's natural defenses with the power ...
3	Stress and Anxiety	Find natural calm and relaxation with traditional ...
4	Inflammation and Pain	Discover nature's solutions for managing discomfor...
5	Skin Health	Nourish and support your skin naturally with herba...
6	Sleep Disorders	Encourage restful sleep with the gentle power of h...
7	Cardiovascular Health	Support a healthy heart and circulation with natur...
8	Detoxification	Gently cleanse and support your body's natural det...

The **healthConcerns table** is the backbone of personalized herb recommendations. It's a structured list of health interests (e.g., "Digestive Issues"), allowing users to choose their specific concerns during signup. This table uses concernID as its primary key, ensuring each health interest has a unique identifier.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**concernID**	int(11)			No	*None*			Change Drop More
2	**concernName**	varchar(50)	utf8mb4_general_ci		No	*None*			Change Drop More
3	**healthDesc**	text	utf8mb4_general_ci		No	*None*			Change Drop More

This concernID is then used as a foreign key in other tables, connecting users to their selected health interests and, ultimately, to the herbs best suited for those concerns when they log in. Without this table, personalizing the herb recommendations would be impossible.

Herb Table

3	Chamomile	Drowsiness	Tea	Relieves indigestion and promotes relaxation	1	img/chamomile.jpg
4	Fennel	Rare allergic reactions	Tea or raw	Reduces bloating and aids digestion	1	img/fennel.jpg
5	Lemon Balm	Rare allergic reactions	Tea or tincture	Helps with bloating and indigestion	1	img/lemonbalm.jpg
6	Slippery Elm	Nausea (rare)	Tea or capsule	Soothes the stomach and digestive tract	1	img/slipperyelm.jpg
7	Echinacea	Mild upset stomach	Tea or capsules	Boosts the immune system and helps fight infection...	2	img/echinacea.jpg

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	**herbID** 🔑	int(11)			No	*None*			Change Drop More
☐	2	**herbName**	varchar(20)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	3	**sideEffect**	varchar(255)	utf8mb4_general_ci		No	*None*			Change Drop More
☐	4	**recommendedUsage**	text	utf8mb4_general_ci		No	*None*			Change Drop More
☐	5	**Benefit**	text	utf8mb4_general_ci		No	*None*			Change Drop More
☐	6	**healthConcerns** 🔗	int(11)			No	*None*			Change Drop More
☐	7	**imagePath**	varchar(255)	utf8mb4_general_ci		No	*None*			Change Drop More

Our **herb table** houses all the manually entered information about the various herbs presented on the website. This table is the central repository for all herb-specific details. To uniquely identify each herb, herbID is designated as the primary key. The healthConcerns column functions as a foreign key, establishing a relationship with the healthConcerns table to indicate the health issues for which a particular herb is relevant.

Savelist Table

The **savelist table** provides functionality for registered users to bookmark herbs they wish to revisit. This feature allows members to save multiple herbs encountered on their homepage or during browsing, consolidating their selected herbs in one accessible location.

userID	herbID
wenny	3
adminWenhan	32
adminWenhan	37
adminWenhan	8
adminWenhan	18
adminEva	26
Wenhan123	25

The savelist table is structured with two foreign key columns. The userID column links to the user table, indicating which member saved a particular herb. The herbID column links to the herb table, specifying the herb that was saved. This table intentionally lacks a primary key because it models a many-to-many relationship: one user can save numerous herbs, and conversely, a single herb can be saved by many different users.

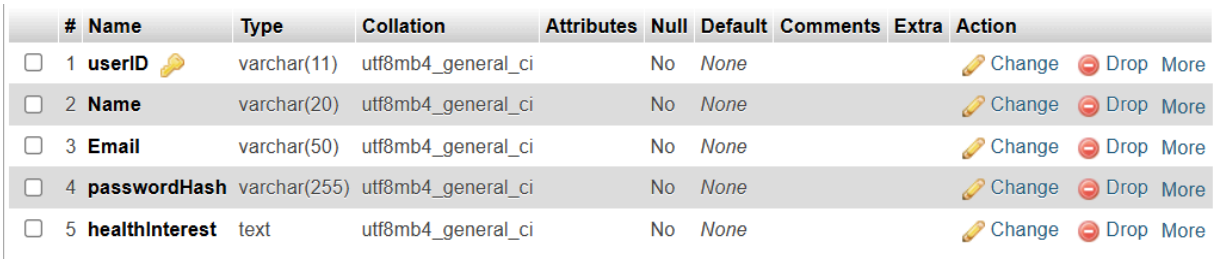
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**userID**	varchar(11)	utf8mb4_general_ci		No	*None*			Change Drop More
2	**herbID**	int(11)			No	*None*			Change Drop More

User Table

userID	Name	Email	passwordHash	healthInterest
admin	Admin	haha@haha.com	$2y$10$QkJbkEXehPKVkTlNjvqBnuZuoXW9/KaS4imuAMB2vUf...	1,5,6,7,8
admin2	adminOwen	w@gmail.com	$2y$10$JKUOKLjMBHDl2PdC7idUgOdH.9lW63muwKaRh12fqul...	1,2
adminEva	Eva	eva@gmail.ca	$2y$10$w6M2RFWQFFt.zYtw/.d.c.X/DbK69T6.V0e40bEc5Yh...	1
adminMark	admin	t@gmail.com	$2y$10$4FCU6GgSJVCpWzGpwBLOBO/ewSJKCeTly3raRuwbm6C...	1,2,3,8
adminMichae	Admin	haha@haha.ca	$2y$10$x9zm5wN/4SFay2fJrDPbzuOVTnmLRyYPUAnv8bgQVJv...	5
adminWenhan	Wenhan	wenhan@haha.usa	$2y$10$XdoryjfeuP4U/tAUCcOIcOwU910fMk0cVeejj/BQ5Yw...	2,5,8
Wenhan123	Wenhan	W@gmail.usa	$2y$10$EAvV1e3RYJZPeRea7gvX5OpjQVomp74dsvd4y/L0qZr...	4,5,6

The **user table** is designed to hold comprehensive information for all registered users. Upon signing up, the table records their name, a unique userID, email address, password (hashed), and their specified health interests. Following the same consistent table structure as presented in

checkpoint one, the userID is designated as the primary key, guaranteeing that each user in the system has a distinct identifier.



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**userID**	varchar(11)	utf8mb4_general_ci		No	*None*			Change Drop More
2	**Name**	varchar(20)	utf8mb4_general_ci		No	*None*			Change Drop More
3	**Email**	varchar(50)	utf8mb4_general_ci		No	*None*			Change Drop More
4	**passwordHash**	varchar(255)	utf8mb4_general_ci		No	*None*			Change Drop More
5	**healthInterest**	text	utf8mb4_general_ci		No	*None*			Change Drop More

This user table is fundamental to the application's login and signup functionality. When a user registers, their information is stored, allowing them to log in securely later. To protect user credentials, the password_hash column stores password data in a secure, hashed format.

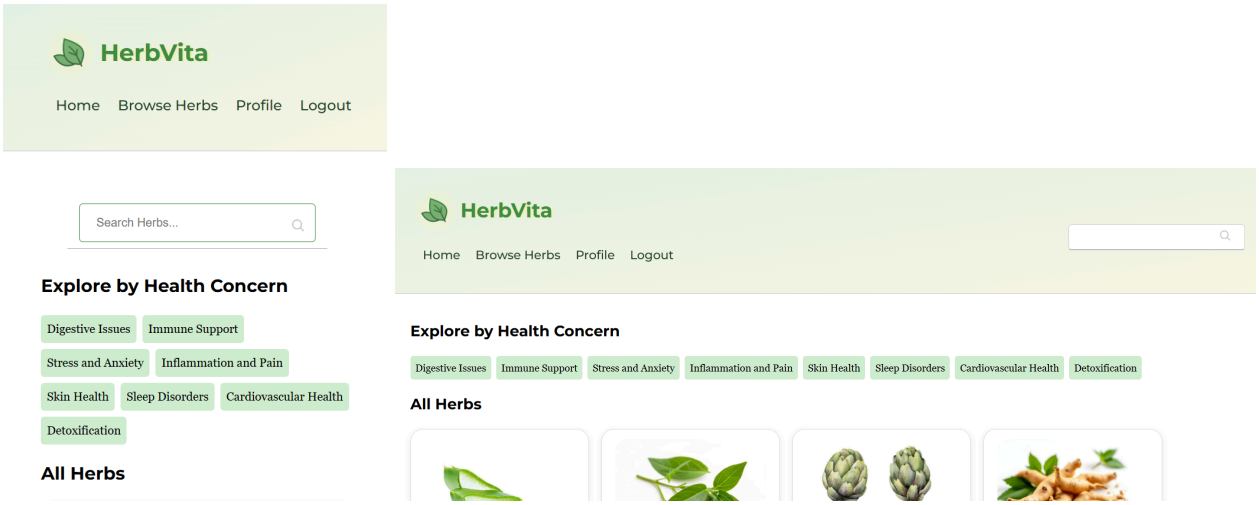
Data Acquisition & Population

The HerbVita database contains information on medicinal herbs, including their benefits, side effects, and user interactions. This data was sourced from MedlinePlus, a reputable database of online articles and research on medicinal properties. As no pre-existing SQL/JSON files were available, the entire database schema and data were created from scratch using phpMyAdmin, with manual data entry into the defined tables.

Implementation Details Functionality 1: Herb Search

Frontend Snapshot

The herb search functionality is typically implemented with a prominent search bar located in the header or main section of the website(depending on the screen size for responsiveness).



(Small screen on the left, full browser screen on the right)

This is the code snippet below is for the main search bar as an example.

```
<div id="search-bar-main">
    <form action="livesearch.php" method="get">
        <input type="text" name="query" placeholder="Search Herbs.
        <span class="search-icon"></span>
        <div id="livesearch-main"></div>
    </form>
</div>
```

Backend Data Processing

When a user initiates a search (either by pressing the "Enter" button once they type in the name of the herb, or through real-time suggestions), the frontend sends the search query to the backend. The backend then processes this query against the herb table in the MySQL database. The image below is from livesearch.php.

```php
if (strlen($q) > 0) {
    $q = $conn->real_escape_string($q);
    // $query = "SELECT herbName FROM herb WHERE herbName LIKE '%" . $q . "%' LIMIT 5";
    $query = "SELECT herbID, herbName FROM herb WHERE LOWER(herbName) LIKE LOWER('" . $q . "%') GROUP BY herbName ORDER BY herbName LIMIT 5";

    $result = $conn->query($query);

    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            if ($hint === "") {
                $hint = "<div onclick='goToHerbPage(\"" . $row['herbID'] . "\")'>" . htmlspecialchars($row['herbName']) . "</div>";
            } else {
                $hint .= "<br /><div onclick='goToHerbPage(\"" . $row['herbID'] . "\")'>" . htmlspecialchars($row['herbName']) . "</div>";
            }
        }
    }
}
```

This snapshot above shows the PHP code for retrieving info from the database by the letters the user types in the search bar, limited to 5 suggestions for a clean look.

Frontend Dynamic Support

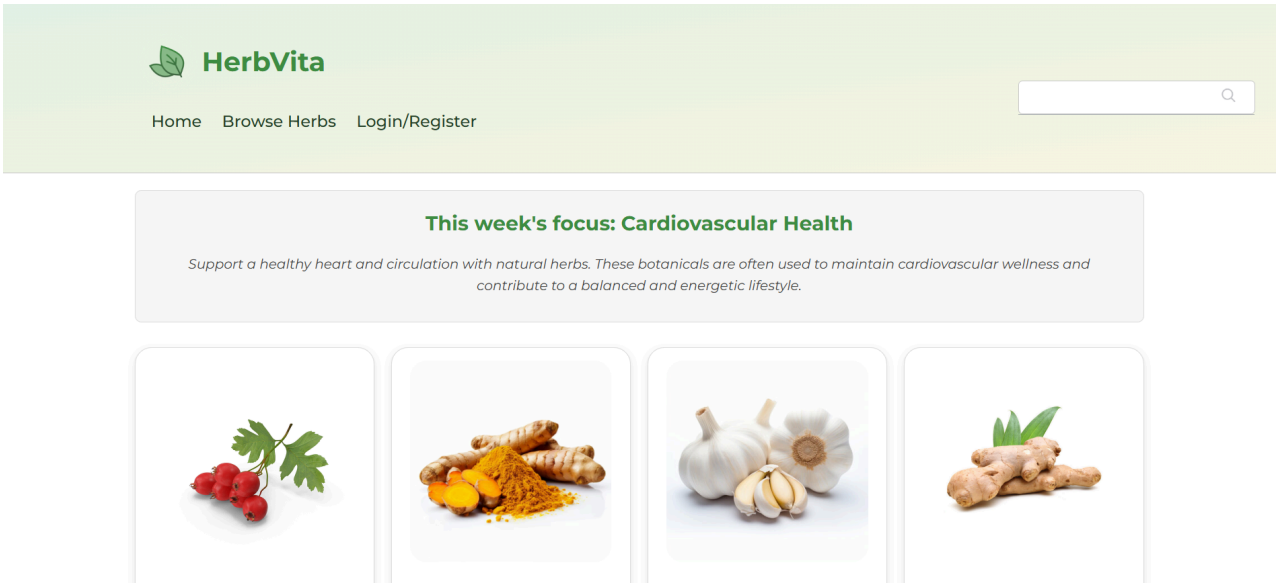
```javascript
function showResult(str) {
    if (str.length === 0) {
        document.getElementById("livesearch").innerHTML = "";
        document.getElementById("livesearch").style.border = "0px";
        return;
    }

    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
        if (this.readyState === 4 && this.status === 200) {
            // checks if successful
        document.getElementById("livesearch").innerHTML = this.responseText;

        document.getElementById("livesearch").style.border = "1px solid #A5ACB2";
        }
    }
    xmlhttp.open("GET", "livesearch.php?q=" + str, true);
    xmlhttp.send();
}
```

Implemented in script.js, the live search feature enhances the user experience by offering immediate feedback during typing. JavaScript initiates an asynchronous HTTP request via XMLHttpRequest to the livesearch.php file (detailed above). This PHP script then queries the herb database, employing a SQL SELECT statement with the LIKE operator to find relevant herb names. A limited number of these matches are sent back to the browser and dynamically rendered below the search input, all without navigating to a new URL.

Implementation Details Functionality 2: Weekly rotation

Frontend Snapshot

The featured health concern is displayed prominently on the homepage for visitors.



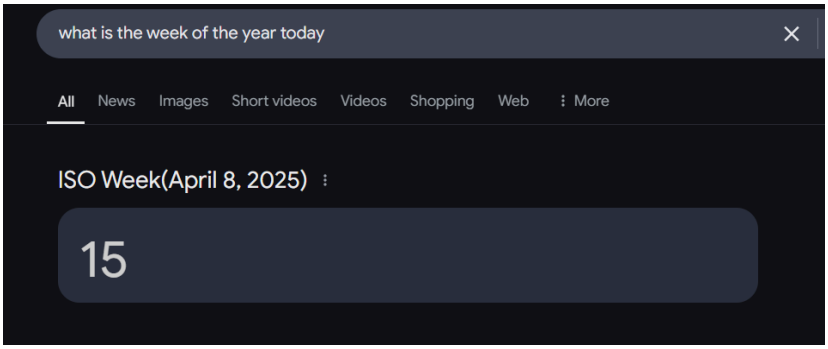
Backend Data Processing

The core of this functionality relies on the modulo operator to cycle through the available health concerns. The modulo operator (%) returns the remainder of a division. In this context, it's used to calculate an index that corresponds to a specific health concern based on the current week number.

```php
function getFeaturedConcernDetails() {
    $currentWeek = getCurrentWeekNumber();
    global $conn;
    $concernDetails = null;
    $concernSql = "SELECT concernID, concernName, healthDesc FROM healthconcerns ORDER BY concernID ASC";
    $concernResult = $conn->query($concernSql);
    if ($concernResult->num_rows > 0) {
        $concerns = $concernResult->fetch_all(MYSQLI_ASSOC);
        if (!empty($concerns)) {
            // takes the current week number of the year(starting from 0) and devide by the number of conc
            this week
            $index = ($currentWeek - 1) % count($concerns);
            $concernDetails = $concerns[$index];
        }
    }
    return $concernDetails;
}
```

Week 1: (1 - 1) % 8 = 0 % 8 = 0

Week 2: (2 - 1) % 8 = 1 % 8 = 1



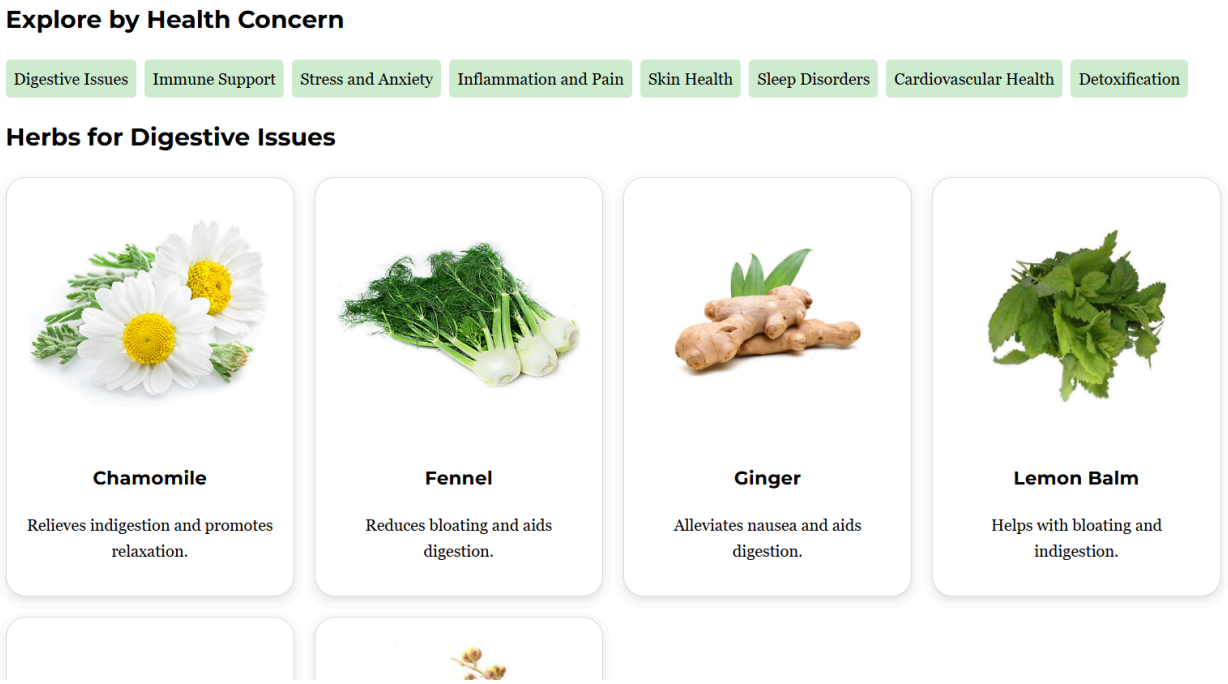
This week(April 8th 2025): (15-1) % 8 = 14 % 8 = 6, which corresponds to the concernID = 7, since my conccernID starts at 1 instead of 0. Which is correct.

	concernID	concernName	healthDesc
Edit Copy Delete	1	Digestive Issues	Support a healthy gut with natural herbal remedies...
Edit Copy Delete	2	Immune Support	Boost your body's natural defenses with the power ...
Edit Copy Delete	3	Stress and Anxiety	Find natural calm and relaxation with traditional ...
Edit Copy Delete	4	Inflammation and Pain	Discover nature's solutions for managing discomfor...
Edit Copy Delete	5	Skin Health	Nourish and support your skin naturally with herba...
Edit Copy Delete	6	Sleep Disorders	Encourage restful sleep with the gentle power of h...
Edit Copy Delete	7	Cardiovascular Health	Support a healthy heart and circulation with natur...
Edit Copy Delete	8	Detoxification	Gently cleanse and support your body's natural det...

Implementation Details Functionality 3: Browse by Category

Frontend Snapshot

Herbs are categorized based on health concerns like digestion, immune support, and stress relief.



Backend Data Processing

```php
if (isset($_POST['concern_id']) && is_numeric($_POST['concern_id'])) {
    $concernID = $conn->real_escape_string($_POST['concern_id']);

    $sql = "SELECT herbID, herbName, Benefit, imagePath
            FROM herb
            WHERE healthConcerns = $concernID
            GROUP BY herbName";

    $result = $conn->query($sql);
```

```php
if ($result->num_rows > 0) {
    echo '<div class="herb-grid">';
    while ($row = $result->fetch_assoc()) {
        $imagePath = htmlspecialchars($row['imagePath']);
        $herbDetailsLink = 'herbDetails.php?id=' . $row['herbID'];

        echo '<div class="herb-item" onclick="window.location.href=\'' . $herbDetailsLink . '\'">';
            echo '<img src="' . $imagePath . '" alt="' . htmlspecialchars($row['herbName']) . '">';
            echo '<h3>' . $row['herbName'] . '</h3>';
            echo '<p>' . substr($row['Benefit'], 0, 100) . '.</p>';
            echo '</div>';
    }
    echo '</div>';
} else {
    echo '<p>No herbs found for this health concern.</p>';
}
```

This PHP code shows how the herbs are displayed after being fetched from the database.

Frontend Dynamic Support

```javascript
concernLinks.forEach(link => {
    link.addEventListener('click', function() {

        const concernId = this.dataset.concernId;

        if (concernId) {
            fetch('herbs_by_concern.php', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/x-www-form-urlencoded',
                },
                body: 'concern_id=' + concernId
            })
            .then(response => response.text())
            .then(data => {
                // Update the content of the featured herbs section with the filtered results
                if (featuredHerbsSection) {
                    featuredHerbsSection.innerHTML = '<h2>Herbs for ' + this.textContent + '</h2>' + data;
                }
            })
            .catch(error => {
                console.error('Error fetching herbs:', error);
                if (featuredHerbsSection) {
                    featuredHerbsSection.innerHTML = '<p>Error loading herbs.</p>';
                }
            });
        }
    });
});
```

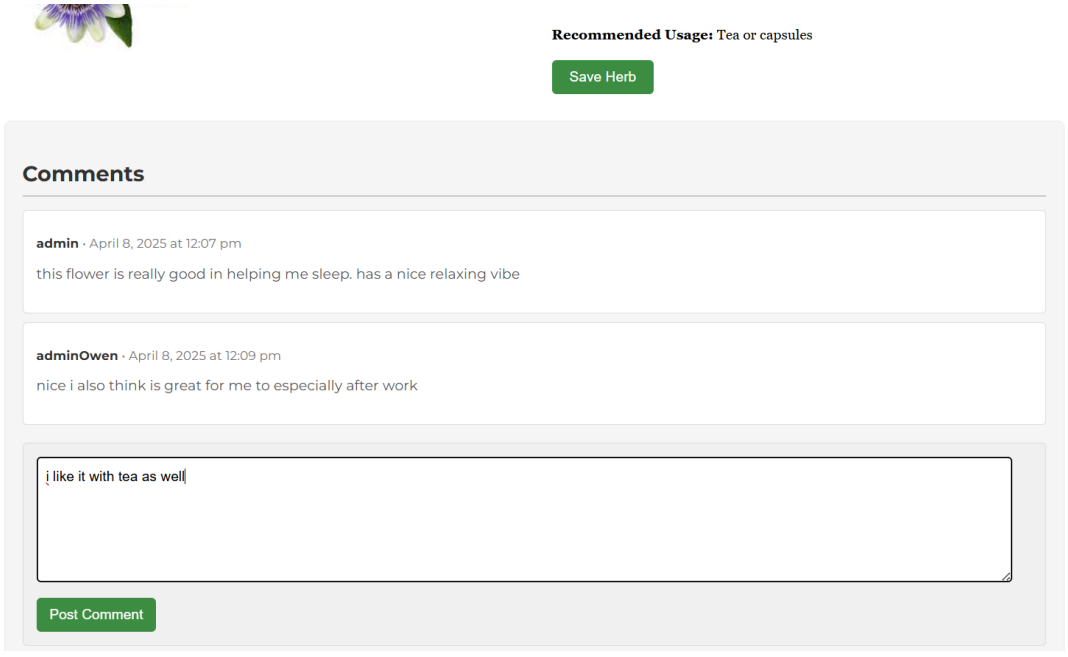
This section describes how clicking on health concern links triggers the filtering of featured herbs. Implemented across herbs_by_concern.php and script.js, this feature dynamically updates the displayed herbs based on the selected health concern.

This JavaScript code implements the AJAX logic for filtering herbs by health concern. It listens for clicks on relevant buttons, retrieves the selected concern's ID, and sends an asynchronous request to herbs_by_concern.php. Upon receiving the filtered herb list from the server, it dynamically updates the "browse by concern" section of the page to display the matching herbs, including error handling for potential issues.

Implementation Details Functionality 4: Comments

Frontend Snapshot

The Remedy Details page features a commenting section where logged-in members can leave multiple comments. Comments are displayed chronologically, with the latest appearing first. Additionally, members can engage with each other's contributions, fostering a community discussion as illustrated in the accompanying frontend snapshot showing two users interacting.



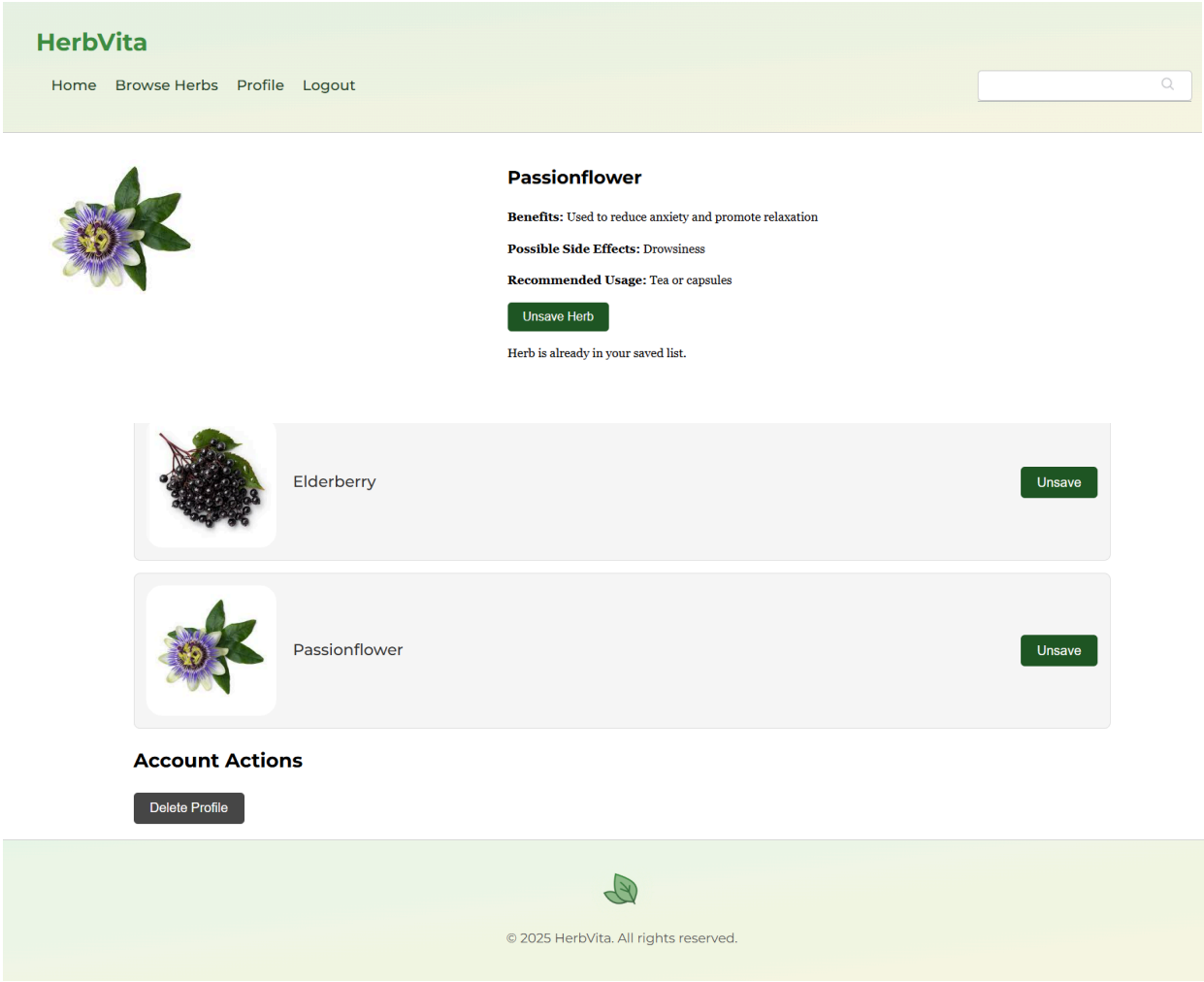
Backend Data Processing

```php
$sql_comments = "SELECT c.`commentText`, u.`Name`, c.`timeAdded`
                        FROM `comments` c
                        JOIN `user` u ON c.`userID` = u.`userID`
                        WHERE c.`herbID` = ?";
        $stmt_comments = $conn->prepare($sql_comments);
        $stmt_comments->bind_param("i", $herbId);
        $stmt_comments->execute();
        $result_comments = $stmt_comments->get_result();

        if ($result_comments->num_rows > 0) {
            echo "<div class='comments-list'>";
            while ($row = $result_comments->fetch_assoc()) {
                echo "<div class='comment-box'>";
                echo "<p><strong>" . htmlspecialchars($row['Name']) . "</strong> • " .
                     date('F j, Y \a\t g:i a', strtotime($row['timeAdded'])) . "</p>";
                echo "<p>" . htmlspecialchars($row['commentText']) . "</p>";
                echo "</div>";
            }
            echo "</div>";
        }

        $stmt_comments->close();
        ?>

        <!-- Comment Form -->
        <?php if (isset($_SESSION['loggedin']) && $_SESSION['loggedin'] === true): ?>
```

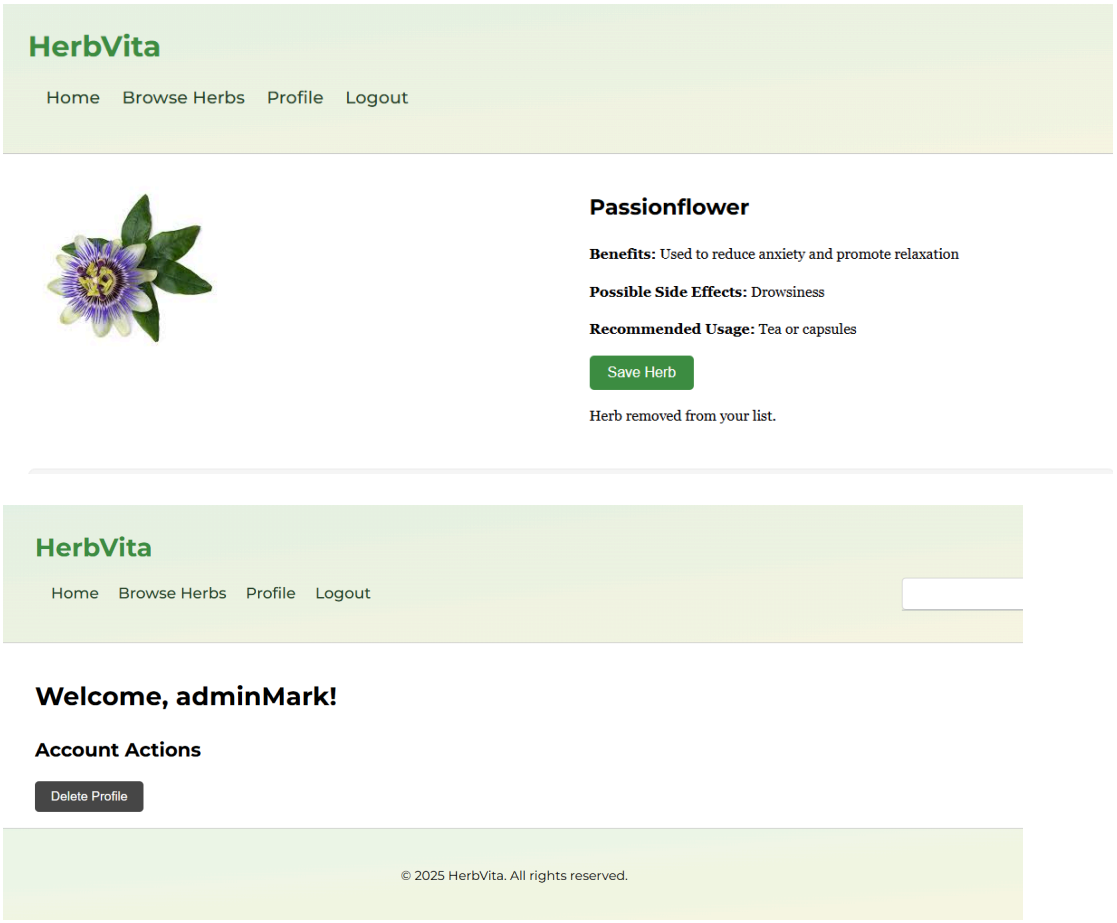
The backend manages comment retrieval and submission. Lines 1-8 establish a database connection and query to fetch all comments, including usernames and timestamps, for page loading. Line 26 checks for user login status ($_SESSION['loggedin']). Logged-in users see a form (<form> action="submitComment.php") for posting new comments, while non-logged-in users are prompted to log in. The backend then dynamically generates a comment-box for each comment, displaying the commenter's name and posting time (lines 10-12).

Implementation Details Functionality 5: User SavedList

Frontend Snapshot



Members can save herbs exclusively from the individual herb's detail page. Upon saving, a persistent notification confirms the action, even after navigating away. Saved herbs are then accessible within the member's profile page.



Members can remove saved herbs by clicking an "unsave" option. Upon doing so, a confirmation message indicates the herb has been removed from their saved list. This change is immediately reflected on their profile page, with the herb no longer appearing in their saved collection.

Backend Data Processing

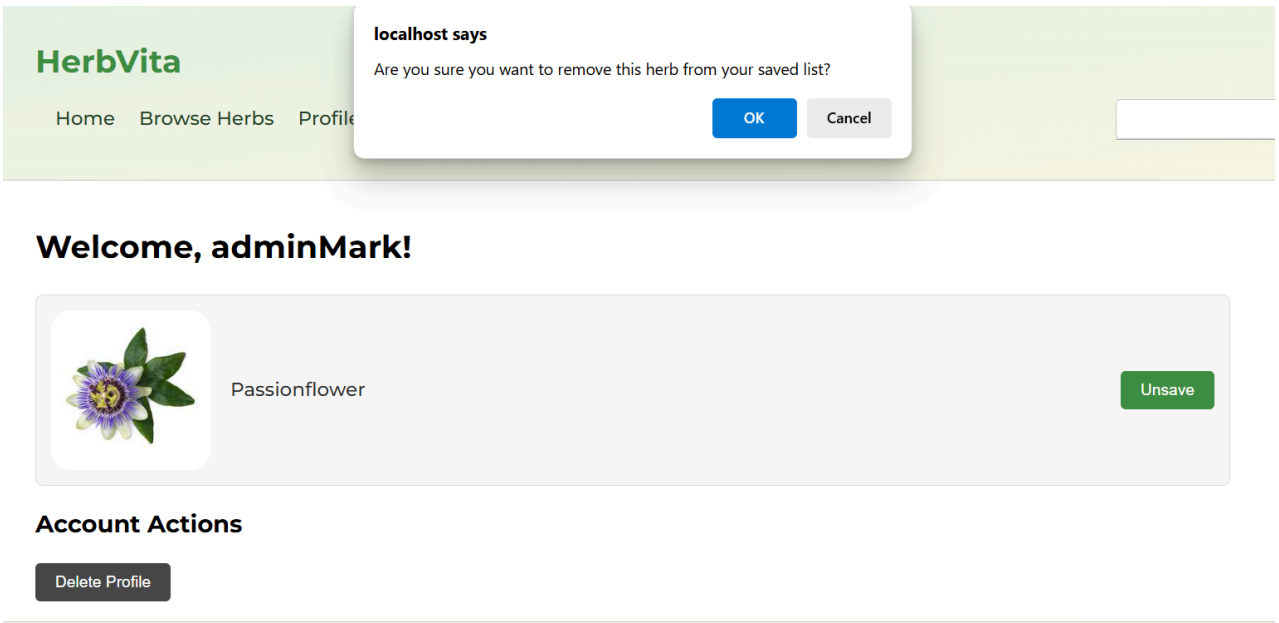
```php
if (isset($_POST['save_herb']) && !$isHerbSaved) {
            $insertSql = "INSERT INTO savedlist (userID, herbID) VALUES (?, ?)";
            $insertStmt = $conn->prepare($insertSql);
            $insertStmt->bind_param("si", $userId, $herbId);
            if ($insertStmt->execute()) {
                $saveMessage = "Herb saved to your list!";
                $isHerbSaved = true;
            } else {
                $saveMessage = "Error saving herb.";
            }
            $insertStmt->close();
        } elseif (isset($_POST['unsave_herb']) && $isHerbSaved) {
            $deleteSql = "DELETE FROM savedlist WHERE userID = ? AND herbID = ?";
            $deleteStmt = $conn->prepare($deleteSql);
            $deleteStmt->bind_param("si", $userId, $herbId);
            if ($deleteStmt->execute()) {
                $saveMessage = "Herb removed from your list.";
                $isHerbSaved = false;
            } else {
                $saveMessage = "Error unsaving herb.";
            }
            $deleteStmt->close();
        }
```

This backend PHP section processes the logic for saving and unsaving herbs. When a member clicks the "save" button ($_POST['save_herb']), the code checks for existing saves of that herb by the user. If the herb is not already saved, it proceeds to store the association in the database, providing feedback to the user with a "Herb saved to your list" message that persists even across refreshes or revisits. Conversely, if the "unsave" button ($_POST['unsave_herb']) is clicked, the code executes a deleteSql query to remove the corresponding userID and herbID entry from the database, updating the user's status with a success or error message based on the database operation's outcome.

```php
if (isset($_POST['save_herb'])&& isset($_SESSION['loggedin']) && $_SESSION['loggedin'] === true) {
        $userId = $_SESSION['userID'];

        $checkSql = "SELECT * FROM savedlist WHERE userID = ? AND herbID = ?";
        $checkStmt = $conn->prepare($checkSql);
        $checkStmt->bind_param("si", $userId, $herbId);
        $checkStmt->execute();
        $checkResult = $checkStmt->get_result();

        if ($checkResult->num_rows == 0) {
            $insertSql = "INSERT INTO savedlist (userID, herbID) VALUES (?, ?)";
            $insertStmt = $conn->prepare($insertSql);
            $insertStmt->bind_param("si", $userId, $herbId);
            if ($insertStmt->execute()) {
                $saveMessage = "Herb saved to your list!";
            } else {
                $saveMessage = "Error saving herb.";
            }
            $insertStmt->close();
        } else {

            $saveMessage = "Herb is already in your saved list.";
        }
        $checkStmt->close();
    }
```

Building upon the previous saving logic, this PHP code segment first ensures user authentication through an initial if statement. It also implements duplicate prevention for saved herbs using a $checkSql query (SELECT * FROM savedlist WHERE userID = ? AND herbID = ?) before proceeding with the save operation.

Frontend Dynamic Support (AJAX, if applicable)



```
if (confirm("Are you sure you want to remove this herb from your saved list?")) {
                // Create AJAX request
                const xhr = new XMLHttpRequest();
                xhr.open("POST", "unsave_herb.php", true);
                xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```

This code implements a dynamic front-end feature using AJAX to unsave herbs from a user's saved list. When a user initiates the action by clicking a button to unsave a herb, the code first displays a confirmation prompt to ensure the user wants to proceed with the removal. If confirmed, the function creates an AJAX request using the XMLHttpRequest object to communicate with the server-side script (unsave_herb.php). This request sends the herbId of the herb to be removed from the saved list, ensuring the data transfer is both asynchronous and seamless.

File Structure

For our project, we structure our files into different file folders for readability and organization. For the frontend folder, it contains all the files that structure the frontend design. The backend folder would contain all the files of the php that handles the backend process of each frontend file. We also made a css folder specifically for the css style file to make it more structured.

Challenges/Design Decision Documentation

We did vigorous troubleshooting throughout the process of building the site to make sure there are no oversight issues. Here are some issues that we found:

ISSUE ONE: storing multiple health interests in your database.

The system encountered an issue where selecting multiple health interests during user registration or profile editing resulted in only one interest ID being stored in the database. Each health interest option has a distinct numerical value (e.g., 1, 2, 3). The desired behavior was to store all selected IDs (e.g., "1,2,3"). After troubleshooting, the solution implemented was to modify the healthInterest column in the SQL database from its original INT(11) data type to TEXT. This adjustment enabled the column to store a string containing multiple health interest IDs, thus correctly capturing all user selections.

Original Code:

```
CREATE TABLE `user` (
  `userID` varchar(11) NOT NULL,
  `Name` varchar(20) NOT NULL,
  `Email` varchar(50) NOT NULL,
  `passwordHash` varchar(255) NOT NULL,
  `healthInterest` int(11) NOT NULL
)
```

Edited Code:

```sql
CREATE TABLE `user` (
  `userID` varchar(11) NOT NULL,
  `Name` varchar(20) NOT NULL,
  `Email` varchar(50) NOT NULL,
  `passwordHash` varchar(255) NOT NULL,
  `healthInterest` text NOT NULL
)
```

ISSUE TWO: Displays the same herbs for the users even when they have nothing saved.

The system was displaying the same set of herbs to all users, even those with no saved preferences.

userID	herbID
0	11
0	48
0	45
0	25
0	13

```php
// Check if the herb is already saved
$checkSql = "SELECT * FROM savedlist WHERE userID = ? AND herbID
$checkStmt = $conn->prepare($checkSql);
$checkStmt->bind_param("ii", $userId, $herbId);
$checkStmt->execute();
$checkResult = $checkStmt->get_result();
```

This was traced to an issue where an String value (userID) was being incorrectly bound with an integer value, leading to implicit type conversion and resulting in zeros. As non-numeric strings convert to 0 in PHP when treated as integers, this causes incorrect data retrieval.

```
$checkStmt->bind_param("si", $userId, $herbId);
```

```
$insertStmt->bind_param("si", $userId, $herbId);
```

userID	herbID 1
0	26
wenny	25

These subsequent screenshot confirms that the fix now correctly associates saved herbs with individual userIDs, ensuring unique saved lists per user.

ISSUE THREE: Comment ID not auto increment

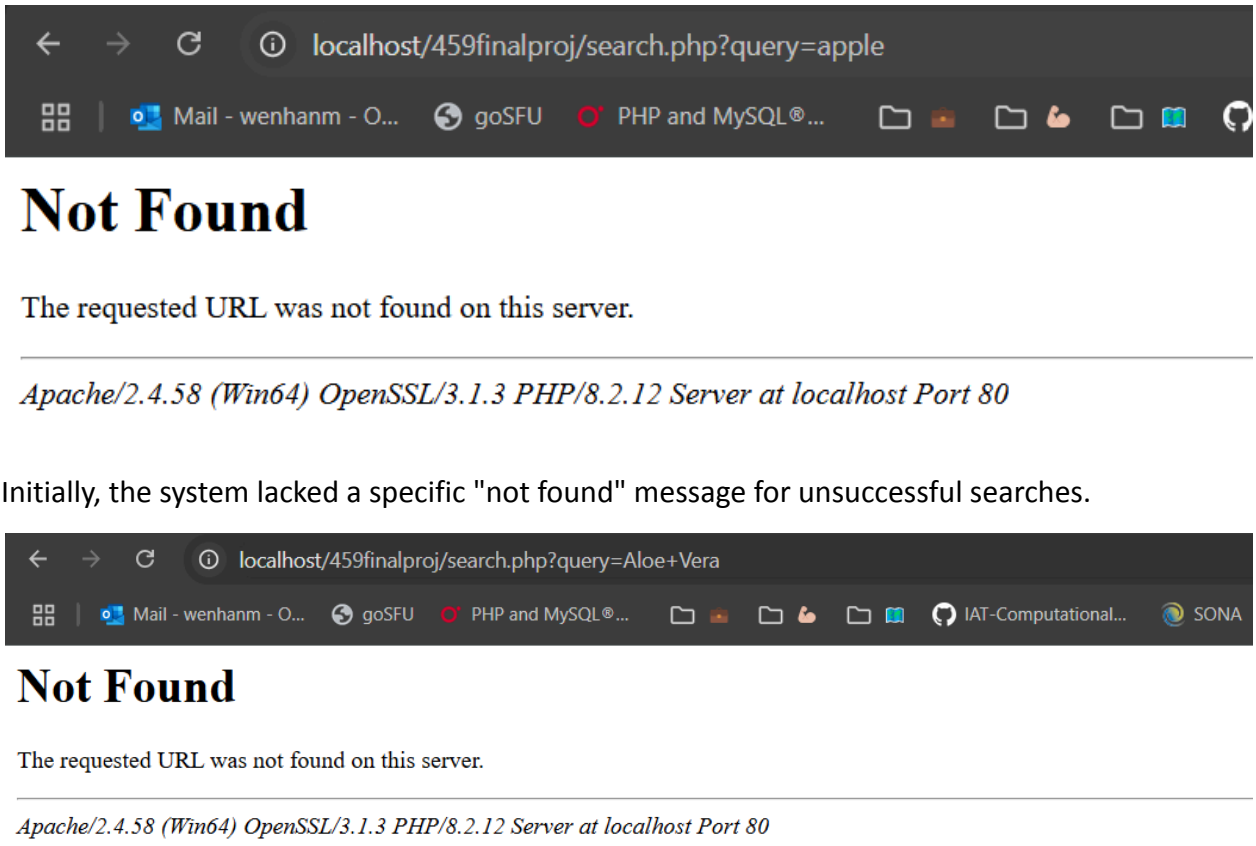
During testing, an issue was discovered where users could successfully post their first comment on an herb, but subsequent attempts to comment on the same herb resulted in an error.

Fatal error: Uncaught mysqli_sql_exception: Duplicate entry '0' for key 'PRIMARY' in C:\documents\htdocs\459FinalProj\submitComment.php:21 Stack trace: #0 C:\documents\htdocs\459FinalProj\submitComment.php(21): mysqli_stmt->execute() #1 {main} thrown in **C:\documents\htdocs\459FinalProj\submitComment.php** on line **21**

The error encountered when commenting multiple times on the same herb was due to the commentID primary key not having the AUTO_INCREMENT property set. This constraint allowed only one unique value for the primary key. The solution involved enabling AUTO_INCREMENT for the commentID in phpMyAdmin, which now automatically generates a unique ID for each new comment, thus resolving the "duplicate entry" error.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	commentID	int(11)			No	*None*		AUTO_INCREMENT	Change Drop More
2	herbID	int(11)			No	*None*			Change Drop More
3	userID	varchar(11)	utf8mb4_general_ci		No	*None*			Change Drop More
4	commentText	text	utf8mb4_general_ci		No	*None*			Change Drop More
5	timeAdded	timestamp			No	current_timestamp()		ON UPDATE CURRENT_TIMESTAMP()	Change Drop More

ISSUE FOUR: Error message when pressed on Enter to search for livesearch function



Initially, the system lacked a specific "not found" message for unsuccessful searches.

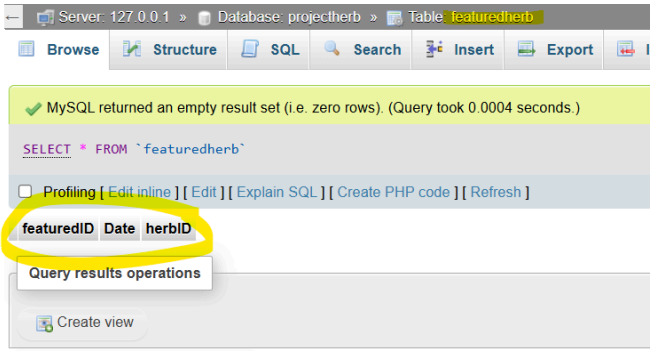
Furthermore, testing revealed that pressing Enter during the AJAX-based live search would prematurely interrupt the process, even for valid queries. To address this, we implemented a code block that checks the entered text.

```
document.querySelector('#search-bar-header input[name="query"]').addEventListener('keydown', function(event) {
    if (event.key === 'Enter') {
    event.preventDefault(); // Prevent the default form submission

    const firstResult = document.querySelector('#livesearch div');
    if (firstResult) {
        const onclickAttribute = firstResult.getAttribute('onclick');
        if (onclickAttribute) {
            const herbIdMatch = onclickAttribute.match(/goToHerbPage\("(\d+)"\)/);
            if (herbIdMatch && herbIdMatch[1]) {
                // see if the searched result matches, if yes then take to the herb detail
                window.location.href = 'herbDetails.php?id=' + herbIdMatch[1];
                }
            }
        }

        // no action if theres no result match
    }
});
```

If it exactly matches an herb in the database, the user is directly navigated to the herb details page. If no match is found, "no suggestion" is displayed, and Enter key submission is disabled to prevent interruption.

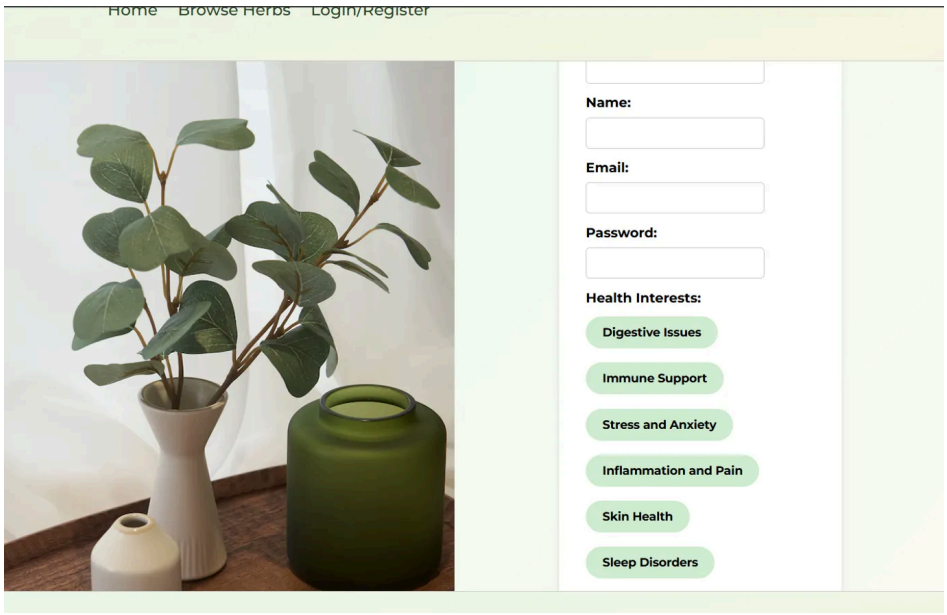
ISSUE FIVE: Unneeded table

During development, we determined that the featured herb table was redundant. Our primary mechanism for showcasing featured herbs relies on a weekly rotation based on health concerns, rendering the dedicated table unnecessary. Consequently, we decided to remove the featured herb table from the database schema.



ISSUE SIX: vertical layout overflow

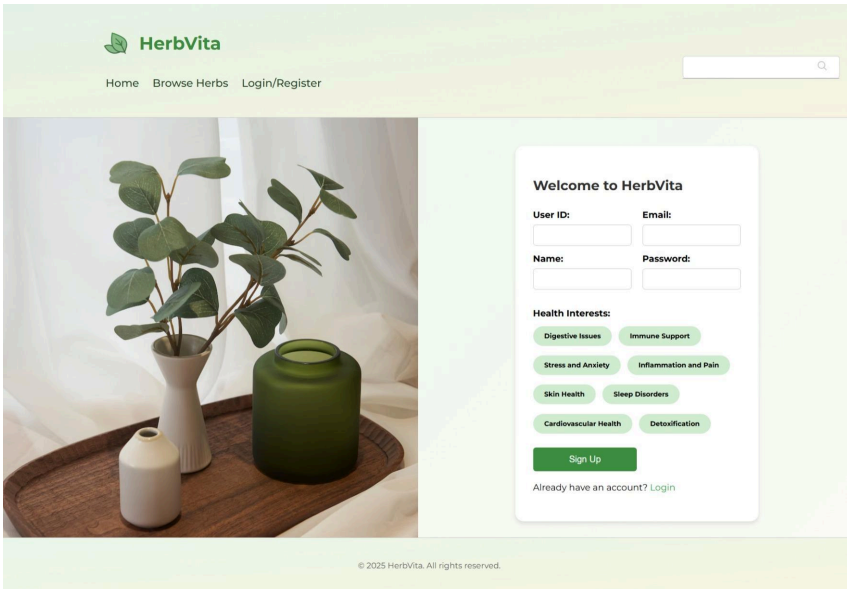
Another issue came up during the final stages of development. This was the signup page. This was an issue encountered when making our pill selector options for our health interest. We didn't notice at first when coding on a larger monitor. However when coding on our laptops, due to the length, the sign up container would break and overflow This was a tricky issue to solve. The solution we did was by playing around with the min-height and restructuring the html code elements.



```
<div class="two-column-container">
                            <!-- Left Column -->
                            <div class="form-column">
                                <div class='form-group'>
                                    <label for="userID">User ID:</label>
                                    <input type="text" id="userID" name="userID" required>
                                </div>

                                <div class='form-group'>
                                    <label for="Name">Name:</label>
                                    <input type="text" id="Name" name="Name" maxlength="20" required>
                                </div>
                            </div>

                            <!-- Right Column -->
                            <div class="form-column">
                                <div class='form-group'>
                                    <label for="Email">Email:</label>
                                    <input type="email" id="Email" name="Email" required>
                                </div>

                                <div class='form-group'>
                                    <label for="passwordHash">Password:</label>
                                    <input type="password" id="passwordHash" name="passwordHash" required>
                                </div>
                            </div>
                        </div>
```

I restructured it to have 2 columns to reduce the size of the sign up container and added more class attributes to control the css flexbox layouts. I also then decided to use min-height:100vh to ensure the element will be at least the height of the viewport to prevent overflow issues. This managed to fix the issue.

```
.sign-up-body {
    margin: 0;
    padding: 0;
    background-color: #f9fcf9;
    min-height: 100vh;
}

.container2 {
    width: 100%;
    min-height: 100vh;
    display: flex;
}
```



Conclusion & Future Improvements

To summarize, the key achievements of HerbVita, our website aims to provide a searchable herbal remedy database, allow users to find herbs based on the health concerns they selected, and provide educational content of the herbs. Some key improvements would be the database structure. For example, creating a many-to-many relationship for the healthInterest column in the data table is more sufficient to represent the relationship between users and health interest. We also could have used classes and namespaces better to improve our code structure to be more organized, to improve the code reusability and reduce naming conflicts when styling the css. Our project uses a procedural approach with individual PHP files that mix database access, business logic, and presentation which isn't the best. Another improvement would be the UI features. Adding more features to user profiles, like profile pictures and customizable dashboards would improve the functionality and features of our website. For the comments, adding a simple like button would be better for the user interactive experience. We also could have added an option to change preferences for the health interest the users select when signing up. This would give the user more freedom. Another implementation that we want to do if we had more time is changing the save and unsave button on the herbDetails into a star, where saved, it becomes yellow, and back to outline when it's not in the user's saved list.

References

W3Schools.com. (n.d.-c). https://www.w3schools.com/mysql/mysql_join.asp

W3Schools.com. (n.d.-d). https://www.w3schools.com/sql/sql_select.asp

W3Schools.com. (n.d.). https://www.w3schools.com/js/js_ajax_intro.asp

W3Schools.com. (n.d.-b). https://www.w3schools.com/xml/ajax_xmlhttprequest_create.asp

Functions - JavaScript | MDN. (2025, March 22). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions

Bootstrap Login Form - free examples, templates & tutorial. (n.d.). MDB - Material Design for Bootstrap. https://mdbootstrap.com/docs/standard/extended/login/

Coyier, C. (2024, August 12). *CSS Flexbox Layout Guide | CSS-Tricks*. CSS-Tricks. https://css-tricks.com/snippets/css/a-guide-to-flexbox/

How to implement one-to-one, one-to-many and many-to-many relationships while designing tables? (n.d.). Stack Overflow. https://stackoverflow.com/questions/7296846/how-to-implement-one-to-one-one-to-many-and-many-to-many-relationships-while-de

Hatoum, F. (2013b, August 28). *Practical uses for the modulo operator — Federico Hatoum*. Federico Hatoum. https://hatoum.com/blog/2012/12/practical-uses-for-modulo-operator.html

U.S. National Library of Medicine. (n.d.). Herbs and supplements. MedlinePlus. https://medlineplus.gov/druginfo/herb_All.html