

User Request Summarized



hello



Hello! How can I assist you today?



Lets make a tap list of beers for a bar, that is a webpage, that will be displayed on a monitor or television mounted in a portrait orientation. I'd like this webpage to be able to be edited easily from a mobile device. At the top of the tap list should be a place for an image or banner with the business name on it



Great! Here's a basic outline for how we can create this tap list webpage:

1. HTML Structure:

- o Banner at the top for the business name.
- o A list of beers with details such as name, type, ABV (Alcohol By Volume), and description.
- o A footer (optional) for any additional information.

2. CSS Styling:

- o Make it look good in portrait orientation.
- o Ensure it is responsive for mobile editing.

3. JavaScript:

- o Allow for easy editing of the beer list from a mobile device.

Here's a simple implementation:

HTML

html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (styles.css)

css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #ccc;
    padding: 10px 0;
}

.beer h2 {
    margin: 0;
    font-size: 1.5em;
}

.beer p {
    margin: 5px 0;
}

.beer span {
```

```
        font-weight: bold;
    }
```

JavaScript (script.js)

javascript

```
document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');

    const beers = [
        { name: 'IPA', type: 'Ale', abv: '6.5%', description: 'A hoppy beer with citrus notes.' },
        { name: 'Stout', type: 'Ale', abv: '8%', description: 'A dark beer with rich coffee and chocolate flavors.' },
        // Add more beers here
    ];

    beers.forEach(beer => {
        const beerElement = document.createElement('div');
        beerElement.classList.add('beer');
        beerElement.innerHTML = `
            <h2>${beer.name}</h2>
            <p>Type: <span>${beer.type}</span></p>
            <p>ABV: <span>${beer.abv}</span></p>
            <p>${beer.description}</p>
        `;
        tapList.appendChild(beerElement);
    });
});
```

Editing Functionality

For editing functionality, you could use a simple approach where the webpage has an admin mode. Here's an example:

HTML (add an edit button)

html

```
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here --&gt;
    &lt;/div&gt;
    &lt;button id="edit-button"&gt;Edit&lt;/button&gt;
    &lt;script src="script.js"&gt;&lt;/script&gt;
&lt;/body&gt;</pre>

```

JavaScript (script.js - add editing functionality)

javascript

```
document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
```

```

const editButton = document.getElementById('edit-button');

const beers = [
    { name: 'IPA', type: 'Ale', abv: '6.5%', description: 'A hoppy beer with citrus notes.' },
    { name: 'Stout', type: 'Ale', abv: '8%', description: 'A dark beer with rich coffee and chocolate flavors.' },
    // Add more beers here
];

const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach(beer => {
        const beerElement = document.createElement('div');
        beerElement.classList.add('beer');
        beerElement.innerHTML = `
            <h2>${beer.name}</h2>
            <p>Type: <span>${beer.type}</span></p>
            <p>ABV: <span>${beer.abv}</span></p>
            <p>${beer.description}</p>
        `;
        tapList.appendChild(beerElement);
    });
}

editButton.addEventListener('click', () => {
    if (editButton.textContent === 'Edit') {
        editButton.textContent = 'Save';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            beerElement.innerHTML = `
                <input type="text" value="${beers[index].name}" class="beer-name">
                <input type="text" value="${beers[index].type}" class="beer-type">
                <input type="text" value="${beers[index].abv}" class="beer-abv">
                <input type="text" value="${beers[index].description}" class="beer-description">
            `;
        });
    } else {
        editButton.textContent = 'Edit';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            const name = beerElement.querySelector('.beer-name').value;
            const type = beerElement.querySelector('.beer-type').value;
            const abv = beerElement.querySelector('.beer-abv').value;
            const description = beerElement.querySelector('.beer-description').value;
            beers[index] = { name, type, abv, description };
        });
        displayBeers();
    }
});
displayBeers();
});

```

This setup gives you a simple, mobile-friendly way to display and edit a beer tap list. The editing is done directly on the page, and changes are reflected immediately.



Can we make this have space for 20 beer options



Sure, we can modify the script to include 20 beer options. Here is the updated version:

HTML

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <button id="edit-button">Edit</button>
    <script src="script.js"></script>
</body>
</html>
```

CSS (styles.css)

```
css

body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #ccc;
    padding: 10px 0;
}

.beer h2 {
    margin: 0;
    font-size: 1.5em;
}

.beer p {
    margin: 5px 0;
}
```

```
.beer span {
    font-weight: bold;
}
```

JavaScript (script.js)

```
javascript

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');

    const beers = Array.from({ length: 20 }, (_, i) => ({
        name: `Beer ${i + 1}`,
        type: 'Type',
        abv: 'ABV',
        description: 'Description'
    }));

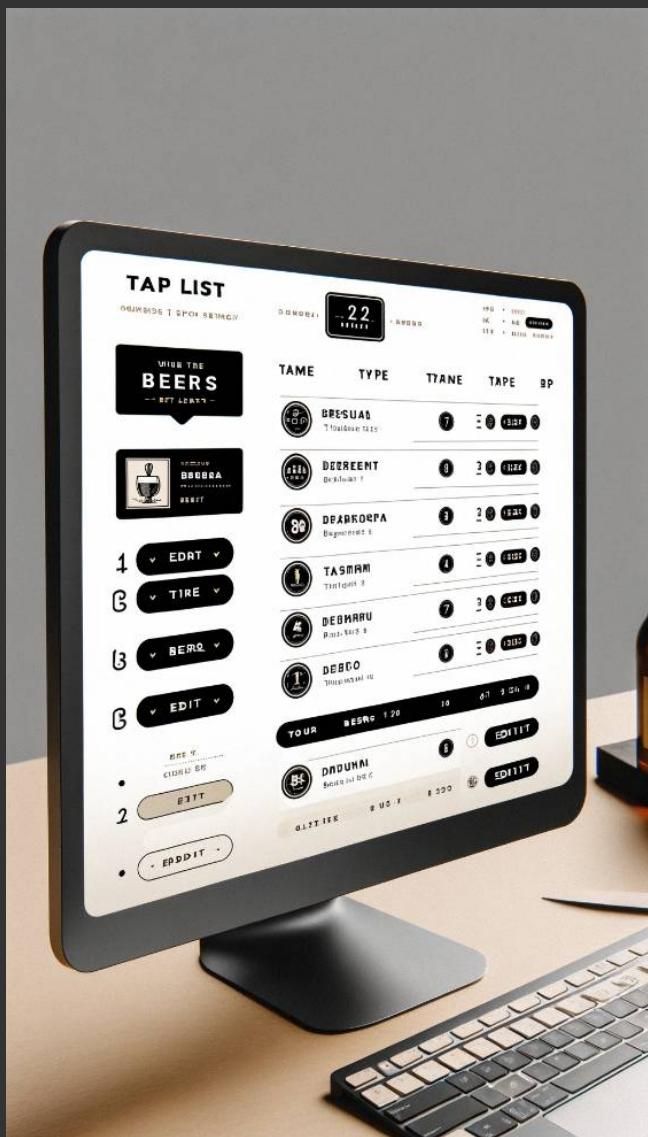
    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach(beer => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <h2>${beer.name}</h2>
                <p>Type: <span>${beer.type}</span></p>
                <p>ABV: <span>${beer.abv}</span></p>
                <p>${beer.description}</p>
            `;
            tapList.appendChild(beerElement);
        });
    };

    editButton.addEventListener('click', () => {
        if (editButton.textContent === 'Edit') {
            editButton.textContent = 'Save';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                beerElement.innerHTML =
                    <input type="text" value="${beers[index].name}" class="beer-name">
                    <input type="text" value="${beers[index].type}" class="beer-type">
                    <input type="text" value="${beers[index].abv}" class="beer-abv">
                    <input type="text" value="${beers[index].description}" class="beer-description">
            });
        } else {
            editButton.textContent = 'Edit';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                const name = beerElement.querySelector('.beer-name').value;
                const type = beerElement.querySelector('.beer-type').value;
                const abv = beerElement.querySelector('.beer-abv').value;
                const description = beerElement.querySelector('.beer-description').value;
                beers[index] = { name, type, abv, description };
            });
            displayBeers();
        }
    });
    displayBeers();
});
```

This setup will display a list of 20 beers with placeholders that can be edited directly on the page.



can you show me roughly what this webpage will look like to an end user?



Here is a rough illustration of what the tap list webpage will look like to an end user. The design includes a banner at the top with space for the business name and image, followed by sections for 20 different beers. Each beer section displays the name, type, ABV, and description, with an 'Edit' button at the bottom for easy modifications.



And can we have it so the end user does not see the edit functionality, but admins can easily access the edit functionailty on a mobile device through some sort of admin panel or admin portal



Sure, we can implement an admin panel that is accessible through a hidden URL or a login page. Here's how you can structure it:

HTML (with admin panel)

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <div id="admin-panel" style="display:none;">
        <button id="edit-button">Edit</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

CSS (styles.css)

```
css

body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #ccc;
```

```

        padding: 10px 0;
    }

    .beer h2 {
        margin: 0;
        font-size: 1.5em;
    }

    .beer p {
        margin: 5px 0;
    }

    .beer span {
        font-weight: bold;
    }

#admin-panel {
    position: fixed;
    bottom: 10px;
    right: 10px;
}

```

JavaScript (script.js)

```

javascript

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const adminPanel = document.getElementById('admin-panel');

    const beers = Array.from({ length: 20 }, (_, i) => ({
        name: `Beer ${i + 1}`,
        type: 'Type',
        abv: 'ABV',
        description: 'Description'
    }));

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach(beer => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML =
                `

## ${beer.name}</h2> <p>Type: <span>${beer.type}</span></p> <p>ABV: <span>${beer.abv}</span></p> <p>${beer.description}</p> `; tapList.appendChild(beerElement); }); }; editButton.addEventListener('click', () => { if (editButton.textContent === 'Edit') { editButton.textContent = 'Save'; const beerElements = document.querySelectorAll('.beer'); beerElements.forEach((beerElement, index) => { beerElement.innerHTML = `


```

```

        const name = beerElement.querySelector('.beer-name').value;
        const type = beerElement.querySelector('.beer-type').value;
        const abv = beerElement.querySelector('.beer-abv').value;
        const description = beerElement.querySelector('.beer-description').value;
        beers[index] = { name, type, abv, description };
    });
    displayBeers();
});
displayBeers();

// Admin login
const queryString = window.location.search;
const urlParams = new URLSearchParams(queryString);
const admin = urlParams.get('admin');

if (admin === 'true') {
    adminPanel.style.display = 'block';
}
);

```

Accessing the Admin Panel

To access the admin panel, you can append `?admin=true` to the URL. For example, if your tap list webpage is hosted at `https://example.com/taplist`, admins can access the editing functionality by visiting `https://example.com/taplist?admin=true`.

This way, regular users will only see the tap list, while admins can access the editing functionality by navigating to the special URL.



is there a way to password protect this admin panel?



Yes, you can password protect the admin panel by implementing a simple login form. When the user successfully logs in, the admin panel will be displayed. Here's how you can do it:

HTML (with login form)

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>

```

```
</div>
<div id="admin-panel" style="display:none;">
    <button id="edit-button">Edit</button>
</div>
<div id="login-form" class="login-form" style="display:none;">
    <h2>Admin Login</h2>
    <input type="password" id="password" placeholder="Enter Password">
    <button id="login-button">Login</button>
</div>
<script src="script.js"></script>
</body>
</html>
```

CSS (styles.css)

css

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #ccc;
    padding: 10px 0;
}

.beer h2 {
    margin: 0;
    font-size: 1.5em;
}

.beer p {
    margin: 5px 0;
}

.beer span {
    font-weight: bold;
}

#admin-panel {
    position: fixed;
    bottom: 10px;
    right: 10px;
}

.login-form {
    position: fixed;
    top: 50%;
```

```

left: 50%;
transform: translate(-50%, -50%);
padding: 20px;
border: 1px solid #ccc;
background: white;
box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
}

.login-form h2 {
margin: 0 0 10px 0;
}

.login-form input {
width: 100%;
padding: 10px;
margin: 10px 0;
box-sizing: border-box;
}

.login-form button {
padding: 10px;
width: 100%;
cursor: pointer;
}

```

JavaScript (script.js)

javascript

```

document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');
  const loginForm = document.getElementById('login-form');
  const loginButton = document.getElementById('login-button');
  const passwordInput = document.getElementById('password');

  const beers = Array.from({ length: 20 }, (_, i) => ({
    name: `Beer ${i + 1}`,
    type: 'Type',
    abv: 'ABV',
    description: 'Description'
  }));

  const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach(beer => {
      const beerElement = document.createElement('div');
      beerElement.classList.add('beer');
      beerElement.innerHTML = `
        <h2>${beer.name}</h2>
        <p>Type: <span>${beer.type}</span></p>
        <p>ABV: <span>${beer.abv}</span></p>
        <p>${beer.description}</p>
      `;
      tapList.appendChild(beerElement);
    });
  };

  editButton.addEventListener('click', () => {
    if (editButton.textContent === 'Edit') {
      editButton.textContent = 'Save';
      const beerElements = document.querySelectorAll('.beer');
      beerElements.forEach((beerElement, index) => {
        beerElement.innerHTML = `
          <input type="text" value="${beers[index].name}" class="beer-name">
          <input type="text" value="${beers[index].type}" class="beer-type">
          <input type="text" value="${beers[index].abv}" class="beer-abv">
          <input type="text" value="${beers[index].description}" class="beer-description">
        `;
      });
    }
  });
}

```

```

    `;
  });
} else {
  editButton.textContent = 'Edit';
  const beerElements = document.querySelectorAll('.beer');
  beerElements.forEach((beerElement, index) => {
    const name = beerElement.querySelector('.beer-name').value;
    const type = beerElement.querySelector('.beer-type').value;
    const abv = beerElement.querySelector('.beer-abv').value;
    const description = beerElement.querySelector('.beer-description').value;
    beers[index] = { name, type, abv, description };
  });
  displayBeers();
}
};

displayBeers();

// Admin login
const queryString = window.location.search;
const urlParams = new URLSearchParams(queryString);
const admin = urlParams.get('admin');

if (admin === 'true') {
  loginForm.style.display = 'block';
}

loginButton.addEventListener('click', () => {
  const password = passwordInput.value;
  if (password === 'your_secure_password') { // Replace with your actual password
    loginForm.style.display = 'none';
    adminPanel.style.display = 'block';
  } else {
    alert('Incorrect password');
  }
});
});

```

Accessing the Admin Panel

To access the admin panel, navigate to the URL with `?admin=true` appended, for example, `https://example.com/taplist?admin=true`. Admins will see the login form, and upon entering the correct password, they will gain access to the editing functionality. Replace `'your_secure_password'` with a secure password of your choice.



and how secure is this login form? I'd it not be easy to steal the password by inspecting the source of the page or something simple like that



You're correct that client-side authentication (like the simple password check in JavaScript) is not very secure because the password can be easily seen in the source code. To implement a more secure login, you should use server-side authentication. Here's a more secure approach using a basic backend with Node.js and Express:

Server-side setup (Node.js and Express)

First, set up a Node.js server with Express to handle the authentication.

1. Install Node.js and Express

If you haven't already, install Node.js and npm. Then, create a new project and install Express:

```
bash
mkdir taplist
cd taplist
npm init -y
npm install express body-parser
```

2. Create `server.js`

Create a file called `server.js`:

```
javascript
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password

app.use(bodyParser.urlencoded({ extended: true }));

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

3. Create `public/index.html`

This will be the main tap list page without the admin panel:

```
html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
```

```
<body>
  <div class="banner">
    
  </div>
  <div class="tap-list" id="tap-list">
    <!-- Beers will be dynamically inserted here -->
  </div>
  <script src="script.js"></script>
</body>
</html>
```

4. Create `public/admin.html`

This page will be shown only to authenticated users:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Admin Tap List</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="banner">
    
  </div>
  <div class="tap-list" id="tap-list">
    <!-- Beers will be dynamically inserted here -->
  </div>
  <div id="admin-panel">
    <button id="edit-button">Edit</button>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

5. Create `public/styles.css`

The same CSS as before:

```
css

body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: start;
}

.banner {
  width: 100%;
  text-align: center;
  margin: 20px 0;
}

#business-name {
```

```

        max-width: 100%;
        height: auto;
    }

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #ccc;
    padding: 10px 0;
}

.beer h2 {
    margin: 0;
    font-size: 1.5em;
}

.beer p {
    margin: 5px 0;
}

.beer span {
    font-weight: bold;
}

#admin-panel {
    position: fixed;
    bottom: 10px;
    right: 10px;
}

.login-form {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    padding: 20px;
    border: 1px solid #ccc;
    background: white;
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
}

.login-form h2 {
    margin: 0 0 10px 0;
}

.login-form input {
    width: 100%;
    padding: 10px;
    margin: 10px 0;
    box-sizing: border-box;
}

.login-form button {
    padding: 10px;
    width: 100%;
    cursor: pointer;
}

```

6. Create `public/script.js`

The same JavaScript as before:

javascript

```

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');

```

```

const editButton = document.getElementById('edit-button');
const adminPanel = document.getElementById('admin-panel');

const beers = Array.from({ length: 20 }, (_, i) => ({
    name: `Beer ${i + 1}`,
    type: 'Type',
    abv: 'ABV',
    description: 'Description'
}));

const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach(beer => {
        const beerElement = document.createElement('div');
        beerElement.classList.add('beer');
        beerElement.innerHTML =
            `

## ${beer.name}



Type: <span>${beer.type}</span></p>
            

ABV: <span>${beer.abv}</span></p>
            

${beer.description}</p>
            `;
        tapList.appendChild(beerElement);
    });
};

editButton.addEventListener('click', () => {
    if (editButton.textContent === 'Edit') {
        editButton.textContent = 'Save';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            beerElement.innerHTML =
                `
                
                
                
                `;
        });
    } else {
        editButton.textContent = 'Edit';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            const name = beerElement.querySelector('.beer-name').value;
            const type = beerElement.querySelector('.beer-type').value;
            const abv = beerElement.querySelector('.beer-abv').value;
            const description = beerElement.querySelector('.beer-description').value;
            beers[index] = { name, type, abv, description };
        });
        displayBeers();
    }
});
displayBeers();
});


```

7. Create `public/login.html`

The login form:

```

html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Login</title>
    <link rel="stylesheet" href="styles.css">
</head>

```

```
<body>
  <div class="login-form">
    <h2>Admin Login</h2>
    <form action="/login" method="POST">
      <input type="password" name="password" placeholder="Enter Password">
      <button type="submit">Login</button>
    </form>
  </div>
</body>
</html>
```

Running the Server

Run the server using Node.js:

```
bash
```

```
node server.js
```

Now, the tap list page will be available at `http://localhost:3000`. The admin panel can be accessed by visiting `http://localhost:3000/login.html` and entering the correct password.

This approach ensures that the password is not visible in the client-side code and provides a more secure method for accessing the admin panel.



can you make these scripts you've generated into downloadable files with the appropriate file extensions so I can download them please?



I have created the necessary files for the tap list web application. You can download the files using the links below:

-
-
-
-
-
-

Download these files, place them in the appropriate directories, and run the server using Node.js as described earlier. If you have any questions or need further assistance, feel free to ask!



When I open the index.html file, there's no beers listed. Explain to me how the list gets generated



The list of beers is dynamically generated using JavaScript when the page loads. Here's an explanation of how the list gets generated:

1. **JavaScript Code:** The JavaScript code in `script.js` runs after the page is fully loaded. This code creates an array of beer objects and dynamically inserts these objects into the HTML.
2. **Beers Array:** The `beers` array contains 20 objects, each representing a beer with properties like `name`, `type`, `abv`, and `description`.
3. **Display Function:** The `displayBeers` function iterates over the `beers` array, creating a new HTML element for each beer and appending it to the `tapList` div.

JavaScript Code in Detail (script.js)

javascript

```
document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list'); // Get the tap list container
    const editButton = document.getElementById('edit-button'); // Get the edit button
    const adminPanel = document.getElementById('admin-panel'); // Get the admin panel

    // Array of beer objects
    const beers = Array.from({ length: 20 }, (_, i) => ({
        name: `Beer ${i + 1}`,
        type: 'Type',
        abv: 'ABV',
        description: 'Description'
    }));

    // Function to display the beers
    const displayBeers = () => {
        tapList.innerHTML = ''; // Clear the tap list
        beers.forEach(beer => {
            const beerElement = document.createElement('div'); // Create a new div for each
            beerElement.classList.add('beer'); // Add the 'beer' class
            beerElement.innerHTML =
                `

## ${beer.name}


                <p>Type: <span>${beer.type}</span></p>
                <p>ABV: <span>${beer.abv}</span></p>
                <p>${beer.description}</p>`;
            // Set the inner HTML with beer details
            tapList.appendChild(beerElement); // Append the beer element to the tap list
        });
    };

    // Event listener for the edit button
    editButton.addEventListener('click', () => {
        if (editButton.textContent === 'Edit') {
            editButton.textContent = 'Save'; // Change button text to 'Save'
            const beerElements = document.querySelectorAll('.beer'); // Get all beer
            elements
            beerElements.forEach((beerElement, index) => {
                beerElement.innerHTML =

```

```

        `; // Replace the beer details with input fields for editing
    });
} else {
    editButton.textContent = 'Edit'; // Change button text back to 'Edit'
    const beerElements = document.querySelectorAll('.beer'); // Get all beer
elements
    beerElements.forEach((beerElement, index) => {
        const name = beerElement.querySelector('.beer-name').value;
        const type = beerElement.querySelector('.beer-type').value;
        const abv = beerElement.querySelector('.beer-abv').value;
        const description = beerElement.querySelector('.beer-description').value;
        beers[index] = { name, type, abv, description }; // Update the beers array
with new values
    });
    displayBeers(); // Re-display the updated list
}
);
displayBeers(); // Initial display of the beers
});

```

Steps to Ensure the List Displays

- Ensure JavaScript is Enabled:** Make sure JavaScript is enabled in your browser.
- Check for Errors:** Open your browser's developer console (usually by pressing F12 or right-clicking and selecting "Inspect" then "Console") to check for any errors that might be preventing the script from running.
- Correct File Paths:** Ensure that `script.js` is correctly linked in your `index.html` file and is located in the same directory as the HTML file.

Here's a quick checklist to verify:

- Your `index.html` should have the script tag at the end of the body.
- The `script.js` file should be in the same directory as `index.html`.

If everything is set up correctly, the beers list should be dynamically generated and displayed when you open `index.html` in a browser. If you encounter any issues or need further assistance, please let me know!



where do I input the beer information for this list?



You can input the beer information directly in the `script.js` file. Currently, the `beers` array is populated with placeholder values. You can replace these placeholders with actual beer information.

Step-by-Step Instructions

- Open `script.js`:** Open the `script.js` file in a text editor.

2. **Edit the `beers` Array:** Replace the placeholder values in the `beers` array with your actual beer information. Here's an example of how you might do it:

javascript

```
document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');

  // Replace this array with your actual beer information
  const beers = [
    { name: 'Pale Ale', type: 'Ale', abv: '5.4%', description: 'A refreshing pale ale with citrus notes.' },
    { name: 'Stout', type: 'Stout', abv: '6.2%', description: 'A dark stout with rich coffee and chocolate flavors.' },
  ];
}
```

javascript

```
{ name: 'Lager', type: 'Lager', abv: '4.8%', description: 'A crisp and clean lager perfect for any occasion.' },
{ name: 'IPA', type: 'Ale', abv: '6.5%', description: 'A hoppy IPA with strong citrus and pine notes.' },
{ name: 'Porter', type: 'Porter', abv: '5.9%', description: 'A robust porter with hints of caramel and chocolate.' },
{ name: 'Wheat Beer', type: 'Wheat', abv: '5.2%', description: 'A smooth wheat beer with a touch of spice and fruit.' },
{ name: 'Saison', type: 'Farmhouse Ale', abv: '7.0%', description: 'A Belgian-style saison with a dry finish and complex flavor.' },
{ name: 'Pilsner', type: 'Lager', abv: '5.0%', description: 'A light and refreshing pilsner with a crisp finish.' },
{ name: 'Amber Ale', type: 'Ale', abv: '5.5%', description: 'An amber ale with a balanced malt profile and a hint of caramel.' },
{ name: 'Sour Ale', type: 'Ale', abv: '4.5%', description: 'A tart and tangy sour ale with a fruity character.' },
{ name: 'Brown Ale', type: 'Ale', abv: '5.3%', description: 'A smooth brown ale with nutty and chocolate notes.' },
{ name: 'Barleywine', type: 'Ale', abv: '10.0%', description: 'A strong and rich barleywine with complex malt flavors.' },
{ name: 'Blonde Ale', type: 'Ale', abv: '4.7%', description: 'A light and easy-drinking blonde ale with a clean finish.' },
{ name: 'Double IPA', type: 'Ale', abv: '8.5%', description: 'An intensely hoppy double IPA with a strong malt backbone.' },
{ name: 'Session IPA', type: 'Ale', abv: '4.5%', description: 'A lower ABV IPA with all the hop flavor and less alcohol.' },
{ name: 'Fruit Beer', type: 'Ale', abv: '5.0%', description: 'A refreshing beer with natural fruit flavors.' },
{ name: 'Belgian Tripel', type: 'Ale', abv: '9.0%', description: 'A strong Belgian tripel with fruity and spicy notes.' },
{ name: 'Cider', type: 'Cider', abv: '5.0%', description: 'A crisp and refreshing apple cider.' },
{ name: 'Kolsch', type: 'Ale', abv: '4.8%', description: 'A German-style kolsch with a light and clean profile.' },
{ name: 'Hefeweizen', type: 'Wheat', abv: '5.4%', description: 'A traditional hefeweizen with banana and clove notes.' }
];

const displayBeers = () => {
  tapList.innerHTML = '';
  beers.forEach(beer => {
    const beerElement = document.createElement('div');
    beerElement.classList.add('beer');
    beerElement.innerHTML =
      `

## ${beer.name}


      <p>Type: <span>${beer.type}</span></p>
      <p>ABV: <span>${beer.abv}</span></p>
      <p>${beer.description}</p>
    `;
    tapList.appendChild(beerElement);
  });
}
```

```

        });
    };

    editButton.addEventListener('click', () => {
        if (editButton.textContent === 'Edit') {
            editButton.textContent = 'Save';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                beerElement.innerHTML =
                    <input type="text" value="${beers[index].name}" class="beer-name">
                    <input type="text" value="${beers[index].type}" class="beer-type">
                    <input type="text" value="${beers[index].abv}" class="beer-abv">
                    <input type="text" value="${beers[index].description}" class="beer-
description">
            });
        } else {
            editButton.textContent = 'Edit';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                const name = beerElement.querySelector('.beer-name').value;
                const type = beerElement.querySelector('.beer-type').value;
                const abv = beerElement.querySelector('.beer-abv').value;
                const description = beerElement.querySelector('.beer-description').value;
                beers[index] = { name, type, abv, description };
            });
            displayBeers();
        }
    });
    displayBeers();
});

```

Steps to Update Beer Information

- Open `script.js`:** Open the `script.js` file in your text editor.
- Locate the `beers` Array:** Find the `beers` array in the code.
- Update the Beers:** Replace the placeholder beer objects with your actual beer information. Each beer object should have a `name`, `type`, `abv`, and `description` property.
- Save the File:** Save the `script.js` file after making your changes.
- Refresh the Browser:** Open `index.html` in your browser and refresh the page to see the updated list of beers.

If you need further assistance or have additional requirements, feel free to ask!



can we update the project to make it so the beer information is stored in a json file?



Sure, storing the beer information in a JSON file can make it easier to manage and update. We will modify the project to load beer information from a `beers.json` file.

Step-by-Step Instructions

1. Create `beers.json`

Create a new file named `beers.json` in the `public` directory and populate it with the beer information:

json

```
[{"name": "Pale Ale", "type": "Ale", "abv": "5.4%", "description": "A refreshing pale ale with citrus notes."}, {"name": "Stout", "type": "Stout", "abv": "6.2%", "description": "A dark stout with rich coffee and chocolate flavors."}, {"name": "Lager", "type": "Lager", "abv": "4.8%", "description": "A crisp and clean lager perfect for any occasion."}, {"name": "IPA", "type": "Ale", "abv": "6.5%", "description": "A hoppy IPA with strong citrus and pine notes."}, {"name": "Porter", "type": "Porter", "abv": "5.9%", "description": "A robust porter with hints of caramel and chocolate."}, {"name": "Wheat Beer", "type": "Wheat", "abv": "5.2%", "description": "A smooth wheat beer with a touch of spice and fruit."}, {"name": "Saison", "type": "Farmhouse Ale", "abv": "7.0%", "description": "A Belgian-style saison with a dry finish and complex flavor."}, {"name": "Pilsner", "type": "Lager", "abv": "5.0%", "description": "A light and refreshing pilsner with a crisp finish."}, {"name": "Amber Ale", "type": "Ale", "abv": "5.5%", "description": "An amber ale with a balanced malt profile and a hint of caramel."}, {"name": "Sour Ale", "type": "Ale", "abv": "4.5%", "description": "A tart and tangy sour ale with a fruity character."}, {"name": "Brown Ale", "type": "Ale", "abv": "5.3%", "description": "A smooth brown ale with nutty and chocolate notes."}, {"name": "Barleywine", "type": "Ale", "abv": "10.0%", "description": "A strong and rich barleywine with complex malt flavors."}, {"name": "Blonde Ale", "type": "Ale", "abv": "4.7%", "description": "A light and easy-drinking blonde ale with a clean finish."}, {"name": "Double IPA", "type": "Ale", "abv": "8.5%", "description": "An intensely hoppy double IPA with a strong malt backbone."}, {"name": "Session IPA", "type": "Ale", "abv": "4.5%", "description": "A lower ABV IPA with all the hop flavor and less alcohol."}, {"name": "Fruit Beer", "type": "Ale", "abv": "5.0%", "description": "A refreshing beer with natural fruit flavors."}, {"name": "Belgian Tripel", "type": "Ale", "abv": "9.0%", "description": "A strong Belgian tripel with fruity and spicy notes."}, {"name": "Cider", "type": "Cider", "abv": "5.0%", "description": "A crisp and refreshing apple cider."}, {"name": "Kolsch", "type": "Ale", "abv": "4.8%", "description": "A German-style kolsch with a light and clean profile."}, {"name": "Hefeweizen", "type": "Wheat", "abv": "5.4%", "description": "A traditional hefeweizen with banana and clove notes."}]
```

2. Update `script.js`

Modify `script.js` to load the beer information from `beers.json`:

javascript

```
document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');
  let beers = [];

  const loadBeers = async () => {
    try {
      const response = await fetch('beers.json');
      beers = await response.json();
      displayBeers();
    } catch (error) {
      console.error('Failed to load beers:', error);
    }
  };
  loadBeers();
});
```

```

        }

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach(beer => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <h2>${beer.name}</h2>
                <p>Type: <span>${beer.type}</span></p>
                <p>ABV: <span>${beer.abv}</span></p>
                <p>${beer.description}</p>
            `;
            tapList.appendChild(beerElement);
        });
    };

editButton.addEventListener('click', () => {
    if (editButton.textContent === 'Edit') {
        editButton.textContent = 'Save';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            beerElement.innerHTML =
                <input type="text" value="${beers[index].name}" class="beer-name">
                <input type="text" value="${beers[index].type}" class="beer-type">
                <input type="text" value="${beers[index].abv}" class="beer-abv">
                <input type="text" value="${beers[index].description}" class="beer-
description">
        });
    } else {
        editButton.textContent = 'Edit';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            const name = beerElement.querySelector('.beer-name').value;
            const type = beerElement.querySelector('.beer-type').value;
            const abv = beerElement.querySelector('.beer-abv').value;
            const description = beerElement.querySelector('.beer-description').value;
            beers[index] = { name, type, abv, description };
        });
        displayBeers();
    }
});
loadBeers();
});

```

Updated File List

- **server.js**: No changes needed.
- **index.html**: No changes needed.
- **admin.html**: No changes needed.
- **styles.css**: No changes needed.
- **script.js**: Updated to load beer information from `beers.json`.
- **beers.json**: New file containing beer information.

I'll create these updated files and make them available for download.

3. Update Files and Download

I'll generate the files with these updates for you to download.

The project files have been updated to load beer information from a `beers.json` file. You can download the updated files using the links below:

-
-
-
- (updated)
- (new)

Download these files, place them in the appropriate directories, and run the server using Node.js as described earlier. If you have any questions or need further assistance, feel free to ask!



When i open index.html, only the banner image is there with no beer information



It sounds like the `beers.json` file might not be loading correctly. Let's make sure the paths are correct and that the JSON file is accessible.

Here are a few things to check and update:

1. **File Structure:** Ensure that your file structure looks like this:

```
arduino
      taplist/
      └── server.js
      └── public/
          ├── index.html
          ├── admin.html
          ├── styles.css
          ├── script.js
          └── beers.json
```

2. **Check JSON File:** Ensure that `beers.json` is correctly formatted and placed in the `public` directory.
3. **Server Configuration:** Make sure your server is running and serving the `public` directory correctly.

Updated `script.js`

Let's make sure the JavaScript code is correctly fetching the `beers.json` file. Here's the updated `script.js` with added error logging for better debugging:

```
javascript
```

```
document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
```

```

const adminPanel = document.getElementById('admin-panel');
let beers = [];

const loadBeers = async () => {
  try {
    const response = await fetch('beers.json');
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    beers = await response.json();
    displayBeers();
  } catch (error) {
    console.error('Failed to load beers:', error);
  }
};

const displayBeers = () => {
  tapList.innerHTML = '';
  beers.forEach(beer => {
    const beerElement = document.createElement('div');
    beerElement.classList.add('beer');
    beerElement.innerHTML =
      `

## ${beer.name}</h2> <p>Type: <span>${beer.type}</span></p> <p>ABV: <span>${beer.abv}</span></p> <p>${beer.description}</p> `; tapList.appendChild(beerElement); }); }; editButton.addEventListener('click', () => { if (editButton.textContent === 'Edit') { editButton.textContent = 'Save'; const beerElements = document.querySelectorAll('.beer'); beerElements.forEach((beerElement, index) => { beerElement.innerHTML = ` `; }); } else { editButton.textContent = 'Edit'; const beerElements = document.querySelectorAll('.beer'); beerElements.forEach((beerElement, index) => { const name = beerElement.querySelector('.beer-name').value; const type = beerElement.querySelector('.beer-type').value; const abv = beerElement.querySelector('.beer-abv').value; const description = beerElement.querySelector('.beer-description').value; beers[index] = { name, type, abv, description }; }); displayBeers(); } }); loadBeers(); );


```

Running the Server

Ensure you run the server with Node.js to correctly serve the files:

```
bash
node server.js
```

Accessing the Page

Open your browser and navigate to `http://localhost:3000/index.html` (or the appropriate URL where your server is running).

Debugging

- **Browser Console:** Open the browser's developer console (F12 or right-click > Inspect > Console) to check for any error messages.
- **Network Tab:** Check the Network tab in the developer tools to ensure `beers.json` is being loaded correctly.

If everything is set up correctly, you should see the list of beers displayed below the banner. Let me know if you encounter any issues or need further assistance!



this is the output when I try to run server.js with node

```
"Uncaught Error Error: Cannot find module 'express'
Require stack:
- /Users/duncananderson/Desktop/Taplist-Project/server.js
  at Module._resolveFilename (internal/modules/cjs/loader:1145:15)
  at Module._load (internal/modules/cjs/loader:986:27)
  at Module.require (internal/modules/cjs/loader:1233:19)
  at require (internal/modules/helpers:179:18)
  at <anonymous> (/Users/duncananderson/Desktop/Taplist-Project/server.js:2:17)
  at Module._compile (internal/modules/cjs/loader:1358:14)
  at Module._extensions..js (internal/modules/cjs/loader:1416:10)
  at Module.load (internal/modules/cjs/loader:1208:32)
  at Module._load (internal/modules/cjs/loader:1024:12)
  at executeUserEntryPoint (internal/modules/run_main:174:12)
  at <anonymous> (internal/main/run_main_module:28:49)"
```



The error message indicates that the `express` module is not installed. You need to install the required dependencies using npm. Here's a step-by-step guide to setting up and running the server:

Step-by-Step Guide

1. Navigate to Project Directory

Open your terminal and navigate to the project directory:

```
bash
cd /Users/duncananderson/Desktop/Taplist-Project
```

2. Initialize npm (if you haven't already)

If you haven't initialized npm in your project directory, run:

```
bash
npm init -y
```

3. Install Express and Body-Parser

Install the required packages by running:

```
bash
npm install express body-parser
```

4. Verify Dependencies in `package.json`

Ensure that your `package.json` file has the correct dependencies listed. It should look something like this:

```
json
{
  "name": "taplist-project",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "body-parser": "^1.19.0",
    "express": "^4.17.1"
  },
  "author": "",
  "license": "ISC"
}
```

5. Run the Server

Start the server using npm:

```
bash
```

```
npm start
```

Alternatively, you can use:

```
bash
```

```
node server.js
```

Full Project Setup

Ensure your project directory structure looks like this:

```
java
```

```
taplist/
├── package.json
├── package-lock.json (generated after running npm install)
└── server.js
└── public/
    ├── index.html
    ├── admin.html
    ├── styles.css
    ├── script.js
    └── beers.json
```

`**server.js**`

Ensure `server.js` looks like this:

```
javascript
```

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password

app.use(bodyParser.urlencoded({ extended: true }));

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.listen(PORT, () => {
```

```
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

Access the Application

After starting the server, open your browser and navigate to `http://localhost:3000/index.html` to see the tap list.

Debugging

- **Check Console:** Open the browser console to see if there are any JavaScript errors.
- **Check Network Requests:** Ensure `beers.json` is being loaded correctly in the Network tab of the developer tools.

Following these steps should help you set up the project and run the server correctly. If you encounter any issues, please let me know!



When I navigate to http://localhost:3000/index.html, there are still no beers listed



Let's go through some troubleshooting steps to ensure everything is set up correctly.

Check File Structure and Content

1. Ensure File Structure

Make sure your file structure looks like this:

```
java

    taplist/
        package.json
        package-lock.json
        server.js
        public/
            index.html
            admin.html
            styles.css
            script.js
            beers.json
```

2. Verify `beers.json` Content

Ensure `beers.json` is correctly formatted and located in the `public` directory. It should look like this:

json

```
[
  { "name": "Pale Ale", "type": "Ale", "abv": "5.4%", "description": "A refreshing pale ale with citrus notes." },
  { "name": "Stout", "type": "Stout", "abv": "6.2%", "description": "A dark stout with rich coffee and chocolate flavors." },
  { "name": "Lager", "type": "Lager", "abv": "4.8%", "description": "A crisp and clean lager perfect for any occasion." },
  { "name": "IPA", "type": "Ale", "abv": "6.5%", "description": "A hoppy IPA with strong citrus and pine notes." },
  { "name": "Porter", "type": "Porter", "abv": "5.9%", "description": "A robust porter with hints of caramel and chocolate." },
  { "name": "Wheat Beer", "type": "Wheat", "abv": "5.2%", "description": "A smooth wheat beer with a touch of spice and fruit." },
  { "name": "Saison", "type": "Farmhouse Ale", "abv": "7.0%", "description": "A Belgian-style saison with a dry finish and complex flavor." },
  { "name": "Pilsner", "type": "Lager", "abv": "5.0%", "description": "A light and refreshing pilsner with a crisp finish." },
  { "name": "Amber Ale", "type": "Ale", "abv": "5.5%", "description": "An amber ale with a balanced malt profile and a hint of caramel." },
  { "name": "Sour Ale", "type": "Ale", "abv": "4.5%", "description": "A tart and tangy sour ale with a fruity character." },
  { "name": "Brown Ale", "type": "Ale", "abv": "5.3%", "description": "A smooth brown ale with nutty and chocolate notes." },
  { "name": "Barleywine", "type": "Ale", "abv": "10.0%", "description": "A strong and rich barleywine with complex malt flavors." },
  { "name": "Blonde Ale", "type": "Ale", "abv": "4.7%", "description": "A light and easy-drinking blonde ale with a clean finish." },
  { "name": "Double IPA", "type": "Ale", "abv": "8.5%", "description": "An intensely hoppy double IPA with a strong malt backbone." },
  { "name": "Session IPA", "type": "Ale", "abv": "4.5%", "description": "A lower ABV IPA with all the hop flavor and less alcohol." },
  { "name": "Fruit Beer", "type": "Ale", "abv": "5.0%", "description": "A refreshing beer with natural fruit flavors." },
  { "name": "Belgian Tripel", "type": "Ale", "abv": "9.0%", "description": "A strong Belgian tripel with fruity and spicy notes." },
  { "name": "Cider", "type": "Cider", "abv": "5.0%", "description": "A crisp and refreshing apple cider." },
  { "name": "Kolsch", "type": "Ale", "abv": "4.8%", "description": "A German-style kolsch with a light and clean profile." },
  { "name": "Hefeweizen", "type": "Wheat", "abv": "5.4%", "description": "A traditional hefeweizen with banana and clove notes." }
]
```

3. Verify `script.js` Content

Ensure `script.js` is correctly fetching the `beers.json` file. Here is the updated content with additional logging for debugging:

javascript

```
document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');
  let beers = [];

  const loadBeers = async () => {
    try {
      const response = await fetch('beers.json');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      beers = await response.json();
      console.log('Beers loaded:', beers); // Debug log
    } catch (error) {
      console.error(error);
    }
  };

  loadBeers().then(() => {
    tapList.innerHTML = '';
    beers.forEach((beer) => {
      const item = document.createElement('div');
      item.classList.add('tap-item');
      item.textContent = `${beer.name} - ${beer.type} - ${beer.abv} - ${beer.description}`;
      tapList.appendChild(item);
    });
  });
});
```

```

        displayBeers();
    } catch (error) {
        console.error('Failed to load beers:', error);
    }
};

const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach(beer => {
        const beerElement = document.createElement('div');
        beerElement.classList.add('beer');
        beerElement.innerHTML =
            `<h2>${beer.name}</h2>
            <p>Type: <span>${beer.type}</span></p>
            <p>ABV: <span>${beer.abv}</span></p>
            <p>${beer.description}</p>
            `;
        tapList.appendChild(beerElement);
    });
};

editButton.addEventListener('click', () => {
    if (editButton.textContent === 'Edit') {
        editButton.textContent = 'Save';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            beerElement.innerHTML =
                `<input type="text" value="${beers[index].name}" class="beer-name">
                <input type="text" value="${beers[index].type}" class="beer-type">
                <input type="text" value="${beers[index].abv}" class="beer-abv">
                <input type="text" value="${beers[index].description}" class="beer-
description">
                `;
        });
    } else {
        editButton.textContent = 'Edit';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            const name = beerElement.querySelector('.beer-name').value;
            const type = beerElement.querySelector('.beer-type').value;
            const abv = beerElement.querySelector('.beer-abv').value;
            const description = beerElement.querySelector('.beer-
description').value;
            beers[index] = { name, type, abv, description };
        });
        displayBeers();
    }
});
loadBeers();
});

```

4. Start the Server

Ensure your server is running:

```
bash
```

```
node server.js
```

5. Open the Console and Network Tab

- o Open your browser's developer tools (F12 or right-click > Inspect).
- o Check the Console tab for any error messages.

- Check the Network tab to ensure `beers.json` is being loaded correctly.

Debugging Tips

- Console Errors:** Look for any error messages in the Console tab.
- Network Requests:** Ensure `beers.json` is successfully loaded in the Network tab. You should see a 200 status code for the `beers.json` request.

If the `beers.json` file is not being loaded or there are errors, the console and network tabs should provide clues to the issue. Let me know what you find in the console or if you need further assistance!



This was in the console tab

```
"Uncaught TypeError: Cannot read properties of null (reading 'addEventListener')
at HTMLDocument.<anonymous> (script.js:36:16)"
```



The error message indicates that the script is trying to add an event listener to an element that doesn't exist in the DOM when the script runs. This likely means that the `editButton` element is not present in `index.html`. Let's make sure the `index.html` file includes the necessary elements.

Ensure `index.html` is Correct

Ensure that `index.html` includes the `editButton` and has the correct structure:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <div id="admin-panel" style="display: none;">
        <button id="edit-button">Edit</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Updated `script.js`

Let's update `script.js` to handle the case where the `editButton` might not be present:

javascript

```

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const adminPanel = document.getElementById('admin-panel');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            console.log('Beers loaded:', beers); // Debug log
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach(beer => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML =
                `

## ${beer.name}


                <p>Type: <span>${beer.type}</span></p>
                <p>ABV: <span>${beer.abv}</span></p>
                <p>${beer.description}</p>
            `;
            tapList.appendChild(beerElement);
        });
    };

    if (editButton) {
        editButton.addEventListener('click', () => {
            if (editButton.textContent === 'Edit') {
                editButton.textContent = 'Save';
                const beerElements = document.querySelectorAll('.beer');
                beerElements.forEach((beerElement, index) => {
                    beerElement.innerHTML =
                        `
                        
                        
                        
                    `;
                });
            } else {
                editButton.textContent = 'Edit';
                const beerElements = document.querySelectorAll('.beer');
                beerElements.forEach((beerElement, index) => {
                    const name = beerElement.querySelector('.beer-name').value;
                    const type = beerElement.querySelector('.beer-type').value;
                    const abv = beerElement.querySelector('.beer-abv').value;
                    const description = beerElement.querySelector('.beer-description').value;
                    beers[index] = { name, type, abv, description };
                });
                displayBeers();
            }
        });
    }
});

```

```
    loadBeers();
});
```

Steps to Verify

- Check HTML Structure:** Ensure `index.html` has the correct structure, including the `editButton` and `adminPanel` elements.
- Run Server:** Start the server using `node server.js`.
- Open Page:** Navigate to `http://localhost:3000/index.html`.
- Check Console:** Open the developer console to check for any errors.
- Check Network:** Ensure `beers.json` is being successfully loaded.

If the beer list still doesn't display, let me know the exact error message or behavior you're observing in the console.



Looks good. Now does the admin.html also use that same json file? And if so, can we make it so when someone edits something on the admin.html page, it pushes to the index.html page for the end user to see



To achieve this functionality, we need to make the `admin.html` page also use the `beers.json` file and implement server-side logic to handle updates to the JSON file. Here's a step-by-step guide to make these changes:

Step-by-Step Guide

- Modify `admin.html` to Load and Edit Beers**
- Update `server.js` to Handle JSON Updates**
- Implement Changes in `script.js`**

1. Modify `admin.html`

Ensure `admin.html` includes an admin interface for editing the beers:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
```

```

        <!-- Beers will be dynamically inserted here -->
    </div>
    <div id="admin-panel">
        <button id="edit-button">Edit</button>
        <button id="save-button" style="display:none;">Save</button>
    </div>
    <script src="admin-script.js"></script>
</body>
</html>

```

2. Update `server.js` to Handle JSON Updates

Update `server.js` to include endpoints for loading and saving the `beers.json` file:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/update-beers', (req, res) => {
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing beers file');
        } else {
            res.send('Beers updated successfully');
        }
    });
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

```

3. Implement Changes in `admin-script.js`

Create `admin-script.js` for handling the admin page logic:

```
javascript

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const saveButton = document.getElementById('save-button');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('/beers');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            console.log('Beers loaded:', beers); // Debug log
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach(beer => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <h2>${beer.name}</h2>
                <p>Type: <span>${beer.type}</span></p>
                <p>ABV: <span>${beer.abv}</span></p>
                <p>${beer.description}</p>
            `;
            tapList.appendChild(beerElement);
        });
    };

    const enableEditing = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <input type="text" value="${beer.name}" class="beer-name">
                <input type="text" value="${beer.type}" class="beer-type">
                <input type="text" value="${beer.abv}" class="beer-abv">
                <input type="text" value="${beer.description}" class="beer-description">
            `;
            tapList.appendChild(beerElement);
        });
    };

    const saveBeers = async () => {
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            const name = beerElement.querySelector('.beer-name').value;
            const type = beerElement.querySelector('.beer-type').value;
            const abv = beerElement.querySelector('.beer-abv').value;
            const description = beerElement.querySelector('.beer-description').value;
            beers[index] = { name, type, abv, description };
        });
    };

    try {
        const response = await fetch('/update-beers', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(beers)
        });
    }
});
```

```

        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }
        alert('Beers updated successfully');
        displayBeers();
    } catch (error) {
        console.error('Failed to save beers:', error);
        alert('Failed to save beers');
    }
};

editButton.addEventListener('click', () => {
    editButton.style.display = 'none';
    saveButton.style.display = 'inline';
    enableEditing();
});

saveButton.addEventListener('click', () => {
    editButton.style.display = 'inline';
    saveButton.style.display = 'none';
    saveBeers();
});

loadBeers();
});

```

Updated File List

Ensure your file structure looks like this:

```

java

taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── styles.css
├── script.js
└── admin-script.js
beers.json

```

With these changes, the `admin.html` page will load the beer information from `beers.json`, allow admins to edit the information, and push the updates to the server. The server will then update the `beers.json` file, and the changes will be reflected in `index.html`.

Run the Server

Start the server:

```

bash

node server.js

```

Access the Pages

- **Admin Page:** Navigate to `http://localhost:3000/admin.html` to edit the beers.
- **User Page:** Navigate to `http://localhost:3000/index.html` to see the updated beers list.

This setup ensures that any changes made on the admin page are saved to `beers.json` and reflected on the user-facing page. Let me know if you encounter any issues or need further assistance!



Awesome, works great. Now we need to make it so the admin.html page is password protected, but protected in a way where you can't just see the password by inspecting the source of the page



To add secure password protection to the `admin.html` page, you can use server-side authentication. This involves setting up a login form and verifying the credentials on the server. Here's how you can achieve this:

Step-by-Step Guide

1. **Create a Login Page**
2. **Update the Server to Handle Login**
3. **Protect the `admin.html` Page**

1. Create a Login Page

Create a new `login.html` file in the `public` directory:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Login</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="login-form">
        <h2>Admin Login</h2>
        <form action="/login" method="POST">
            <input type="password" name="password" placeholder="Enter Password" required>
            <button type="submit">Login</button>
        </form>
    </div>
</body>
</html>
```

2. Update the Server to Handle Login

Update `server.js` to include session handling for login:

```
javascript
```

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/update-beers', (req, res) => {
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing beers file');
        } else {
            res.send('Beers updated successfully');
        }
    });
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

3. Protect the `admin.html` Page

Ensure `admin.html` is protected by checking the session. This is already handled in the updated `server.js` with this part of the code:

```
javascript

app.get('/admin.html', (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
});
```

Updated File List

Ensure your file structure looks like this:

```
java

taplist/
├── package.json
├── package-lock.json
└── server.js
└── public/
    ├── index.html
    ├── admin.html
    ├── login.html
    ├── styles.css
    ├── script.js
    └── admin-script.js
└── beers.json
```

Run the Server

Start the server:

```
bash

node server.js
```

Access the Pages

- Login Page:** Navigate to `http://localhost:3000/login.html` to login.
- Admin Page:** After logging in, you'll be redirected to `http://localhost:3000/admin.html`.
- User Page:** Navigate to `http://localhost:3000/index.html` to see the updated beers list.

This setup ensures that the `admin.html` page is password-protected, and the password is not visible in the client-side source code. Let me know if you encounter any issues or need further assistance!



I got this message when running server.js with node

```
"Uncaught Error Error: Cannot find module 'express-session'
Require stack:
- /Users/duncananderson/Desktop/Taplist-Project/server.js
  at Module._resolveFilename (internal/modules/cjs/loader:1145:15)
  at Module._load (internal/modules/cjs/loader:986:27)
  at Module.require (internal/modules/cjs/loader:1233:19)
  at require (internal/modules/helpers:179:18)
  at <anonymous> (/Users/duncananderson/Desktop/Taplist-Project/server.js:5:17)
  at Module._compile (internal/modules/cjs/loader:1358:14)
  at Module._extensions..js (internal/modules/cjs/loader:1416:10)
  at Module.load (internal/modules/cjs/loader:1208:32)
  at Module._load (internal/modules/cjs/loader:1024:12)
  at executeUserEntryPoint (internal/modules/run_main:174:12)
  at <anonymous> (internal/main/run_main_module:28:49)"
```



The error message indicates that the `express-session` module is not installed. You need to install this module using npm. Here's how you can do it:

Step-by-Step Guide

1. Install `express-session`

Run the following command in your project directory to install `express-session`:

```
bash
npm install express-session
```

2. Verify `package.json`

Ensure that your `package.json` file includes `express-session` as a dependency. It should look something like this:

```
json
{
  "name": "taplist-project",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
}
```

```

"dependencies": {
  "body-parser": "^1.19.0",
  "express": "^4.17.1",
  "express-session": "^1.17.1"
},
"author": "",
"license": "ISC"
}

```

3. Update `server.js`

Ensure `server.js` includes the necessary logic for session handling:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure
password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 'your_secret_key', // Replace with a secure key
  resave: false,
  saveUninitialized: true
}));

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/update-beers', (req, res) => {
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing beers file');
    }
  });
});

```

```

        } else {
            res.send('Beers updated successfully');
        }
    });

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

```

4. Run the Server

Start the server using npm:

```

bash

npm start

```

Or, if you prefer:

```

bash

node server.js

```

Access the Application

- Login Page:** Navigate to `http://localhost:3000/login.html` to login.
- Admin Page:** After logging in, you'll be redirected to `http://localhost:3000/admin.html`.
- User Page:** Navigate to `http://localhost:3000/index.html` to see the updated beers list.

This setup ensures that the `admin.html` page is password-protected, and the password is not visible in the client-side source code. Let me know if you encounter any issues or need further assistance!



Where is the password for the login page checked?



The password for the login page is checked in the `/login` route handler in `server.js`. When a user submits the login form, the password is sent to the server via a POST request. The server then compares the submitted password to the predefined `ADMIN_PASSWORD`. If the password is correct, the server sets a session variable to indicate that the user is logged in and redirects them to the `admin.html` page. If the password is incorrect, the server responds with an error message.

Here is the relevant part of `server.js` that handles the password check:

server.js

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/update-beers', (req, res) => {
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing beers file');
        } else {
            res.send('Beers updated successfully');
        }
    });
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

```

Explanation

- **Session Setup:** The `express-session` middleware is used to handle sessions. It stores session data on the server side and sets a cookie with the session ID on the client side.
- **Login Route (`/login`):** When a POST request is made to `/login`, the server extracts the `password` from the request body. It then compares this password to the predefined `ADMIN_PASSWORD`.
 - If the passwords match, the server sets `req.session.loggedIn = true` and redirects the user to `/admin.html`.
 - If the passwords do not match, the server responds with 'Incorrect password'.
- **Admin Route Protection (`/admin.html`):** When a GET request is made to `/admin.html`, the server checks if `req.session.loggedIn` is `true`.
 - If the user is logged in, the server allows the request to proceed by calling `next()`.
 - If the user is not logged in, the server redirects the user to `/login.html`.

Steps to Test

1. **Start the Server:** Run `node server.js` to start the server.
2. **Access the Login Page:** Navigate to `http://localhost:3000/login.html`.
3. **Login:** Enter the password you set in `ADMIN_PASSWORD` and submit the form.
4. **Access the Admin Page:** After logging in, you should be redirected to `http://localhost:3000/admin.html`.

This setup ensures secure password protection, with the password validation performed on the server side. If you encounter any issues or have further questions, please let me know!



It appears that I can access the admin.html page directly without first logging. Can we fix that so users must authenticate with login.html before they access admin



To ensure users must authenticate before accessing the `admin.html` page, we need to properly handle session checking in the server. Let's make sure our middleware for session validation is correctly configured and that the `admin.html` page is protected.

Step-by-Step Guide to Ensure Authentication

1. **Ensure Middleware is Correctly Set Up**
2. **Protect the `admin.html` Page**

1. Ensure Middleware is Correctly Set Up

We need to make sure that our middleware correctly checks for a logged-in session. We'll create middleware for session validation and apply it to the `admin.html` route.

Update `server.js`

Here is the updated `server.js`:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

app.use(express.static(path.join(__dirname, 'public')));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedin) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedin = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing beers file');
        } else {
            res.send('Beers updated successfully');
        }
    });
});

app.listen(PORT, () => {
```

```
    console.log(`Server is running on http://localhost:${PORT}`);
});
```

Explanation

- **Session Middleware:** `isAuthenticated` middleware checks if `req.session.loggedin` is `true`. If not, it redirects to the login page.
- **Login Route (`/login`):** Verifies the password and sets the session variable `loggedin`.
- **Protected Routes:** Apply `isAuthenticated` middleware to routes that need protection, like `/admin.html` and `/update-beers`.

2. Ensure File Structure and Content

Make sure your file structure looks like this:

```
java
taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── login.html
├── styles.css
├── script.js
└── admin-script.js
beers.json
```

Run the Server

Start the server:

```
bash
node server.js
```

Access the Application

1. **Login Page:** Navigate to `http://localhost:3000/login.html`.
2. **Login:** Enter the password you set in `ADMIN_PASSWORD` and submit the form.
3. **Admin Page:** After logging in, you'll be redirected to `http://localhost:3000/admin.html`.
4. **User Page:** Navigate to `http://localhost:3000/index.html` to see the updated beers list.

This setup ensures that the `admin.html` page is password-protected, and users must authenticate via `login.html` before accessing it. Let me know if you encounter any issues or need further assistance!



I can still just access the admin.html page directly without logging in



It seems like the session middleware is not correctly protecting the `admin.html` page. Let's ensure that the session middleware is correctly configured and properly protecting the `admin.html` route.

Ensure Middleware Configuration

1. Update Server Configuration to Protect Static Files
2. Explicitly Protect Admin Routes

1. Update Server Configuration to Protect Static Files

We need to make sure that the static files middleware does not serve `admin.html` directly without checking the session. We can do this by ensuring the middleware order is correct.

Update `server.js`

Here's the updated `server.js` with the correct middleware configuration:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedin) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

app.use(express.static(path.join(__dirname, 'public')));

// Protect the /admin.html route
app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});
```

```

});
```

```

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});
```

```

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});
```

```

// Protect the /update-beers route
app.post('/update-beers', isAuthenticated, (req, res) => {
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing beers file');
    } else {
      res.send('Beers updated successfully');
    }
  });
});
```

```

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Explanation

- **Order of Middleware:** `express.static` middleware is now called before the route protection middleware. This ensures that `/admin.html` is protected by `isAuthenticated`.
- **Protected Routes:** `/admin.html` and `/update-beers` routes are explicitly protected by `isAuthenticated` middleware.

Run the Server

Start the server again:

```

bash
node server.js
```

Test the Application

1. **Login Page:** Navigate to `http://localhost:3000/login.html`.
2. **Login:** Enter the password you set in `ADMIN_PASSWORD` and submit the form.
3. **Admin Page:** After logging in, you'll be redirected to `http://localhost:3000/admin.html`.
4. **Attempt Direct Access:** Try navigating directly to `http://localhost:3000/admin.html` without logging in to ensure it redirects to the login page.

This should now properly protect the `admin.html` page, ensuring users must log in before accessing it. Let me know if you encounter any issues or need further assistance!



I can still directly access admin.html without logging in



To ensure that the `admin.html` page is protected and cannot be accessed directly without logging in, let's adjust the middleware and static file serving setup more carefully.

Key Points to Ensure Proper Protection

1. **Static Middleware should not serve `admin.html` directly.**
2. **Protect `admin.html` route with authentication middleware.**

Updated `server.js`

Here is the complete and updated `server.js`:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});
```

```

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing beers file');
    } else {
      res.send('Beers updated successfully');
    }
  });
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

```

Explanation

- Middleware for Static Files:** A custom middleware checks if the requested path is `/admin.html`. If so, it calls `isAuthenticated` to check for a logged-in session.
- Static Middleware:** `express.static` is called after the custom middleware to serve other static files.
- Route Protection:** The `admin.html` route and the `/update-beers` endpoint are explicitly protected by `isAuthenticated`.

Run the Server

- Install Dependencies:** Ensure you have installed the required dependencies by running:

```

bash

npm install express body-parser express-session

```

- Start the Server:** Run the server using:

```
bash
```

```
node server.js
```

Access the Application

- Login Page:** Navigate to `http://localhost:3000/login.html`.
- Login:** Enter the password you set in `ADMIN_PASSWORD` and submit the form.
- Admin Page:** After logging in, you'll be redirected to `http://localhost:3000/admin.html`.
- Attempt Direct Access:** Try navigating directly to `http://localhost:3000/admin.html` without logging in to ensure it redirects to the login page.

This setup ensures that the `admin.html` page is properly protected and users must log in before accessing it. Let me know if you encounter any issues or need further assistance!



Works awesome, now can we have the beers displayed on index.html formatted with these font and alternating line colors? With a black background for the main page



Sure! To format the beers displayed on `index.html` with the specified styles, we will update the CSS to include the following:

- Black background for the main page.**
- Font styling for the beers.**
- Alternating line colors for the beer list.**

Updated `styles.css`

Here is the updated `styles.css` with the required styles:

```
css

body {
    font-family: Arial, sans-serif;
    background-color: #000; /* Black background */
    color: #fff; /* White text color */
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}
```

```
#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #444;
    padding: 10px 0;
    background-color: #111; /* Dark background for beer items */
}

.beer:nth-child(even) {
    background-color: #222; /* Alternate color for even items */
}

.beer h2 {
    margin: 0;
    font-size: 1.5em;
    color: #ffcc00; /* Custom font color for beer name */
}

.beer p {
    margin: 5px 0;
    color: #ddd; /* Lighter color for

```



- 1. Ekto Kooler by Eagle Park Brewing Co - Hard Seltzer - 5.0%**
- 2. Two Hearted by Bell's Brewery - IPA - 7.0%**
- 3. Gumballhead by 3 Floyds Brewing - Wheat APA - 5.6%**
- 4. 15-2 by 3 Sheeps Brewing - Stout - 7.2%**

Works awesome, now can we have the beers displayed on index.html formatted with these font and alternating line colors? With a black background for the main page



Sure! Based on the image you provided, we'll style the `index.html` to match the fonts and alternating line colors shown. Let's update the `styles.css` accordingly.

Updated `styles.css`

css

```
body {
    font-family: Arial, sans-serif;
    background-color: #000; /* Black background */
    color: #fff; /* White text color */
}
```

```

margin: 0;
padding: 0;
display: flex;
flex-direction: column;
align-items: center;
justify-content: start;
}

.banner {
  width: 100%;
  text-align: center;
  margin: 20px 0;
}

#business-name {
  max-width: 100%;
  height: auto;
}

.tap-list {
  width: 90%;
  margin: 0 auto;
}

.beer {
  border-bottom: 1px solid #444;
  padding: 10px 0;
  background-color: #ffeb7a; /* Light yellow background for odd items */
  color: #000; /* Black text color for better contrast */
}

.beer:nth-child(even) {
  background-color: #ffad88; /* Light red background for even items */
}

.beer h2 {
  margin: 0;
  font-size: 1.5em;
  color: #000; /* Black text color for beer name */
}

.beer p {
  margin: 5px 0;
  color: #000; /* Black text color for beer details */
}

.beer span {
  font-weight: bold;
  color: #000; /* Black text color for emphasized text */
}

```

Ensure `script.js` Uses the Correct HTML Structure

Ensure your `script.js` is correctly setting the structure for each beer item:

javascript

```

document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');
  let beers = [];

  const loadBeers = async () => {
    try {
      const response = await fetch('beers.json');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
    }
  }

```

```

        beers = await response.json();
        displayBeers();
    } catch (error) {
        console.error('Failed to load beers:', error);
    }
};

const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach(beer => {
        const beerElement = document.createElement('div');
        beerElement.classList.add('beer');
        beerElement.innerHTML =
            `

## ${beer.name}</h2> <p>Type: <span>${beer.type}</span></p> <p>ABV: <span>${beer.abv}</span></p> <p>${beer.description}</p> `; tapList.appendChild(beerElement); }); }; if (editButton) { editButton.addEventListener('click', () => { if (editButton.textContent === 'Edit') { editButton.textContent = 'Save'; const beerElements = document.querySelectorAll('.beer'); beerElements.forEach((beerElement, index) => { beerElement.innerHTML = `


```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display with the specified fonts, alternating line colors, and a black background for the main page. Let me know if you need any further adjustments!



1. Ekto Kooler by Eagle Park Brewing Co - Hard Seltzer - 5.0%

this is how I'd like the beers to be displayed. All one line, the line should contain: [beer name], by, [brewery], [abv]. With hyphens separating each info in the line



Got it! We'll update the `script.js` and `styles.css` to match the format you provided. The line should contain the beer name, "by", the brewery, and the ABV, all separated by hyphens. We'll also ensure it is all on one line.

Updated `script.js`

We'll format the inner HTML of each beer item to match the desired format.

```
javascript

document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');
  let beers = [];

  const loadBeers = async () => {
    try {
      const response = await fetch('beers.json');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      beers = await response.json();
      displayBeers();
    } catch (error) {
      console.error('Failed to load beers:', error);
    }
  };

  const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach((beer, index) => {
      const beerElement = document.createElement('div');
      beerElement.classList.add('beer');
      beerElement.innerHTML = `
        <p>
          <span class="beer-index">${index + 1}.</span>
          <span class="beer-name">${beer.name}</span> -
          <span class="brewery">${beer.brewery}</span> -
          <span class="abv">${beer.abv}</span>
        </p>
      `;
      tapList.appendChild(beerElement);
    });
  };

  if (editButton) {
    editButton.addEventListener('click', () => {
      if (editButton.textContent === 'Edit') {
        editButton.textContent = 'Save';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {

```

```

        beerElement.innerHTML = `
            <input type="text" value="${beers[index].name}" class="beer-name">
            <input type="text" value="${beers[index].brewery}" class="brewery">
            <input type="text" value="${beers[index].abv}" class="abv">
        `;
    } else {
        editButton.textContent = 'Edit';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach((beerElement, index) => {
            const name = beerElement.querySelector('.beer-name').value;
            const brewery = beerElement.querySelector('.brewery').value;
            const abv = beerElement.querySelector('.abv').value;
            beers[index] = { name, brewery, abv };
        });
        displayBeers();
    });
}
loadBeers();
);

```

Updated `styles.css`

We'll update the styles to ensure the beers are displayed on one line and with the correct colors.

```

css

body {
    font-family: Arial, sans-serif;
    background-color: #000; /* Black background */
    color: #fff; /* White text color */
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #444;
    padding: 10px 0;
    background-color: #ffeb7a; /* Light yellow background for odd items */
    color: #000; /* Black text color for better contrast */
}

.beer:nth-child(even) {
    background-color: #ffad88; /* Light red background for even items */
}

.beer p {

```

```
margin: 0;
padding: 10px;
display: flex;
align-items: center;
}

.beer .beer-index {
  font-weight: bold;
  margin-right: 10px;
}

.beer .beer-name {
  font-weight: bold;
  margin-right: 5px;
}

.beer .brewery,
.beer .abv {
  margin-left: 5px;
}
```

Ensure `beers.json` includes brewery information

Make sure your `beers.json` file includes the brewery information for each beer. Here's an example:

```
json

[
  { "name": "Ekto Kooler", "brewery": "Eagle Park Brewing Co", "abv": "5.0%" },
  { "name": "Two Hearted", "brewery": "Bell's Brewery", "abv": "7.0%" },
  { "name": "Gumballhead", "brewery": "3 Floyds Brewing", "abv": "5.6%" },
  { "name": "15-2", "brewery": "3 Sheeps Brewing", "abv": "7.2%" }
]
```

Run the Server and Test

- 1. Start the Server:** Ensure your server is running with `node server.js`.
- 2. Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display with the specified format, alternating line colors, and a black background for the main page. Let me know if you need any further adjustments!



1. **Ekto Kooler** by Eagle Park Brewing Co - Hard Seltzer - 5.0%
2. **Two Hearted** by Bell's Brewery - IPA - 7.0%
3. **Gumballhead** by 3 Floyds Brewing - Wheat APA - 5.6%
4. **15-2** by 3 Sheeps Brewing - Stout - 7.2%
5. **Mango Cart** by Golden Roard Brewing - Mango Wheat - 4.0%
6. **Gumballhead Blood Orange & Lemon** by 3 Floyds Brewing - Fruited Pale Ale - 4.8%
7. **Lavender Lemonade** by Third Space Brewing Co - Fruited Sour - 6.0%
8. **Lakefront IPA** by Lakefront Brewery - IPA - 6.7%
9. **Furious** by Surly Brewing Co - IPA - 6.7%
10. **Spotted Cow** by New Glarus Brewing - Farmhouse Ale - 4.8%
11. **Radiant Haze** by Toppling Goliath - IPA - 7.0%
12. **Warped Speed** by Lake Louie Brewing - Scotch Ale - 6.9%
13. **Fat Tire** by New Belgium Brewing - Ale - 5.2%
14. **Mortal Bloom** by Founders Brewing - Hazy IPA - 6.2%
15. **Wild Apple** by Wild State Cider - Cider - 6.3%
16. **Stone Hazy** by Stone Brewing - Hazy IPA - 6.7%
17. **Peach Mango** by Downeast Cider - Cider - 5.0%
18. **Farmer's Market: Cherry** by Saturday Beer Co. - Fruited Sour - 4.2%
19. **Cold Time** by Revolution Brewing - Lager - 4.8%
20. **Electric Sunshine** by Avery Brewing - Fruited Sour Ale - 5.1%

Can you make the beers.json file using this beer information?



It looks like there was an internal error while trying to create the `beers.json` file. Let me create the JSON content for you directly so you can manually copy it to your `beers.json` file.

beers.json

json

```
[
    { "name": "Ekto Kooler", "brewery": "Eagle Park Brewing Co", "type": "Hard Seltzer",
      "abv": "5.0%" },
      { "name": "Two Hearted", "brewery": "Bell's Brewery", "type": "IPA", "abv": "7.0%" },
      { "name": "Gumballhead", "brewery": "3 Floyds Brewing", "type": "Wheat APA", "abv":
      "5.6%" },
      { "name": "15-2", "brewery": "3 Sheeps Brewing", "type": "Stout", "abv": "7.2%" },
      { "name": "Mango Cart", "brewery": "Golden Road Brewing", "type": "Mango Wheat", "abv":
      "4.0%" },
      { "name": "Gumballhead Blood Orange & Lemon", "brewery": "3 Floyds Brewing", "type":
      "Fruited Pale Ale", "abv": "4.8%" },
      { "name": "Lavender Lemonade", "brewery": "Third Space Brewing Co", "type": "Fruited
      Sour", "abv": "6.0%" },
      { "name": "Lakefront IPA", "brewery": "Lakefront Brewery", "type": "IPA", "abv": "6.7%" },
    },
    { "name": "Furious", "brewery": "Surly Brewing Co", "type": "IPA", "abv": "6.7%" },
    { "name": "Spotted Cow", "brewery": "New Glarus Brewing", "type": "Farmhouse Ale",
      "abv": "4.8%" },
    { "name": "Radiant Haze", "brewery": "Toppling Goliath", "type": "IPA", "abv": "7.0%" },
    { "name": "Warped Speed", "brewery": "Lake Louie Brewing", "type": "Scotch Ale", "abv":
      "6.9%" },
    { "name": "Fat Tire", "brewery": "New Belgium Brewing", "type": "Ale", "abv": "5.2%" },
    { "name": "Mortal Bloom", "brewery": "Founders Brewing", "type": "Hazy IPA", "abv":
      "6.2%" },
    { "name": "Wild Apple", "brewery": "Wild State Cider", "type": "Cider", "abv": "6.3%" },
    { "name": "Stone Hazy", "brewery": "Stone Brewing", "type": "Hazy IPA", "abv": "6.7%" },
    { "name": "Peach Mango", "brewery": "Downeast Cider", "type": "Cider", "abv": "5.0%" },
    { "name": "Farmer's Market: Cherry", "brewery": "Saturday Beer Co.", "type": "Fruited
      Sour", "abv": "4.2%" },
    { "name": "Cold Time", "brewery": "Revolution Brewing", "type": "Lager", "abv": "4.8%" },
    { "name": "Electric Sunshine", "brewery": "Avery Brewing", "type": "Fruited Sour Ale",
      "abv": "5.1%" }
]
```

Instructions

- Create a new file** named `beers.json` in your `public` directory.
- Copy and paste** the above JSON content into the `beers.json` file.
- Save the file.**

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display with the updated information. Let me know if you need any further adjustments!



Here's a rough idea how to format the text in each line

```
<td><span style="font-size: 29pt;">1.&nbsp;&nbsp;</span><span style="font-size: 29pt; color: black; font-weight: bold">Ekto Kooler</span><span style="font-size: 21pt; color: black"> by<b> Eagle Park Brewing Co - Hard Seltzer - 5.0%</b></span></td>"
```



Sure! We'll adjust the `script.js` and `styles.css` to format each line according to the example you provided. We will use inline styles to match the specified formatting for the beer list.

Updated `script.js`

We will format each beer item to include the desired inline styles.

```
javascript

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const adminPanel = document.getElementById('admin-panel');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <p>
                    <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
                    <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                    <span style="font-size: 21pt; color: black;"> by <b>${beer.brewery} - ${beer.type} - ${beer.abv}</b></span>
                </p>
            `;
            tapList.appendChild(beerElement);
        });
    };

    if (editButton) {
        editButton.addEventListener('click', () => {
            if (editButton.textContent === 'Edit') {
                editButton.textContent = 'Save';
                const beerElements = document.querySelectorAll('.beer');
                beerElements.forEach((beerElement, index) => {
                    beerElement.innerHTML = `
                        <p>
                            <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
                            <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                            <span style="font-size: 21pt; color: black;"> by <b>${beer.brewery} - ${beer.type} - ${beer.abv}</b></span>
                        </p>
                    `;
                });
            }
        });
    }
});
```

```

        <input type="text" value="${beers[index].name}" class="beer-name">
        <input type="text" value="${beers[index].brewery}" class="brewery">
        <input type="text" value="${beers[index].type}" class="type">
        <input type="text" value="${beers[index].abv}" class="abv">
    `;
}
} else {
    editButton.textContent = 'Edit';
    const beerElements = document.querySelectorAll('.beer');
    beerElements.forEach((beerElement, index) => {
        const name = beerElement.querySelector('.beer-name').value;
        const brewery = beerElement.querySelector('.brewery').value;
        const type = beerElement.querySelector('.type').value;
        const abv = beerElement.querySelector('.abv').value;
        beers[index] = { name, brewery, type, abv };
    });
    displayBeers();
}
});
loadBeers();
);

```

Updated `styles.css`

We'll make minimal updates to ensure the general styles align with the black background and alternating row colors.

css

```

body {
    font-family: Arial, sans-serif;
    background-color: #000; /* Black background */
    color: #fff; /* White text color */
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #444;
    padding: 10px 0;
    background-color: #ffeb7a; /* Light yellow background for odd items */
    color: #000; /* Black text color for better contrast */
}

.beer:nth-child(even) {
    background-color: #ffad88; /* Light red background for even items */
}

```

```

        }
    .beer p {
        margin: 0;
        padding: 10px;
        display: flex;
        align-items: center;
    }
}

```

Ensure `beers.json` includes the required information

Make sure your `beers.json` file includes the required fields for each beer:

json

```
[
    { "name": "Ekto Kooler", "brewery": "Eagle Park Brewing Co", "type": "Hard Seltzer",
    "abv": "5.0%" },
    { "name": "Two Hearted", "brewery": "Bell's Brewery", "type": "IPA", "abv": "7.0%" },
    { "name": "Gumballhead", "brewery": "3 Floyds Brewing", "type": "Wheat APA", "abv":
    "5.6%" },
    { "name": "15-2", "brewery": "3 Sheeps Brewing", "type": "Stout", "abv": "7.2%" },
    { "name": "Mango Cart", "brewery": "Golden Road Brewing", "type": "Mango Wheat", "abv":
    "4.0%" },
    { "name": "Gumballhead Blood Orange & Lemon", "brewery": "3 Floyds Brewing", "type":
    "Fruited Pale Ale", "abv": "4.8%" },
    { "name": "Lavender Lemonade", "brewery": "Third Space Brewing Co", "type": "Fruited
    Sour", "abv": "6.0%" },
    { "name": "Lakefront IPA", "brewery": "Lakefront Brewery", "type": "IPA", "abv": "6.7%" },
    { "name": "Furious", "brewery": "Surly Brewing Co", "type": "IPA", "abv": "6.7%" },
    { "name": "Spotted Cow", "brewery": "New Glarus Brewing", "type": "Farmhouse Ale",
    "abv": "4.8%" },
    { "name": "Radiant Haze", "brewery": "Toppling Goliath", "type": "IPA", "abv": "7.0%" },
    { "name": "Warped Speed", "brewery": "Lake Louie Brewing", "type": "Scotch Ale", "abv":
    "6.9%" },
    { "name": "Fat Tire", "brewery": "New Belgium Brewing", "type": "Ale", "abv": "5.2%" },
    { "name": "Mortal Bloom", "brewery": "Founders Brewing", "type": "Hazy IPA", "abv":
    "6.2%" },
    { "name": "Wild Apple", "brewery": "Wild State Cider", "type": "Cider", "abv": "6.3%" },
    { "name": "Stone Hazy", "brewery": "Stone Brewing", "type": "Hazy IPA", "abv": "6.7%" },
    { "name": "Peach Mango", "brewery": "Downeast Cider", "type": "Cider", "abv": "5.0%" },
    { "name": "Farmer's Market: Cherry", "brewery": "Saturday Beer Co.", "type": "Fruited
    Sour", "abv": "4.2%" },
    { "name": "Cold Time", "brewery": "Revolution Brewing", "type": "Lager", "abv": "4.8%" },
    { "name": "Electric Sunshine", "brewery": "Avery Brewing", "type": "Fruited Sour Ale",
    "abv": "5.1%" }
]
```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display with the specified formatting. Let me know if you need any further adjustments!



server.js gave this message when I tried to run it

```
/usr/local/bin/node ./server.js
Process exited with code 1
Uncaught ReferenceError ReferenceError: document is not defined
  at <anonymous> (/Users/duncananderson/Desktop/Taplist-Project/server.js:1:1)
  at Module._compile (internal/modules/cjs/loader:1358:14)
  at Module._extensions..js (internal/modules/cjs/loader:1416:10)
  at Module.load (internal/modules/cjs/loader:1208:32)
  at Module._load (internal/modules/cjs/loader:1024:12)
  at executeUserEntryPoint (internal/modules/run_main:174:12)
  at <anonymous> (internal/main/run_main_module:28:49)
```



It looks like there's some confusion here: `document` is a part of the browser's DOM API and cannot be used directly in a Node.js environment. The error indicates that you might be trying to use client-side code on the server side.

Let's clarify the separation between the client-side JavaScript (which should be in a separate `script.js` file served to the browser) and the server-side code (which should be in `server.js`).

Ensure Client-Side Code is in a Separate File

Make sure the code that interacts with the DOM is in a file like `script.js` that is referenced by your HTML, not in `server.js`.

Correct File Setup

- Client-Side Script (`script.js`):** This script handles DOM manipulation and should be placed in the `public` directory.
- Server-Side Script (`server.js`):** This script handles serving files and the backend logic.

Client-Side Script (`script.js`)

```
javascript

document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');
  let beers = [];

  const loadBeers = async () => {
    try {
      const response = await fetch('beers.json');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      beers = await response.json();
    } catch (error) {
      console.error(error);
    }
  };

  loadBeers().then(() => {
    tapList.innerHTML = beers.map(beer => `- ${beer.name}
`).join('');
    editButton.addEventListener('click', () => {
      adminPanel.style.display = 'block';
    });
  });
});
```

```

        displayBeers();
    } catch (error) {
        console.error('Failed to load beers:', error);
    }
};

const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach((beer, index) => {
        const beerElement = document.createElement('div');
        beerElement.classList.add('beer');
        beerElement.innerHTML = `
            <p>
                <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
                <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                    <span style="font-size: 21pt; color: black;"> by <b>${beer.brewery} - ${beer.type} - ${beer.abv}</b></span>
                </p>
            `;
        tapList.appendChild(beerElement);
    });
}

if (editButton) {
    editButton.addEventListener('click', () => {
        if (editButton.textContent === 'Edit') {
            editButton.textContent = 'Save';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                beerElement.innerHTML = `
                    <input type="text" value="${beers[index].name}" class="beer-name">
                    <input type="text" value="${beers[index].brewery}" class="brewery">
                    <input type="text" value="${beers[index].type}" class="type">
                    <input type="text" value="${beers[index].abv}" class="abv">
                `;
            });
        } else {
            editButton.textContent = 'Edit';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                const name = beerElement.querySelector('.beer-name').value;
                const brewery = beerElement.querySelector('.brewery').value;
                const type = beerElement.querySelector('.type').value;
                const abv = beerElement.querySelector('.abv').value;
                beers[index] = { name, brewery, type, abv };
            });
            displayBeers();
        }
    });
}
loadBeers();
});

```

Server-Side Script (`server.js`)

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

```

```

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 'your_secret_key', // Replace with a secure key
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
};

// Serve static files from public directory, but protect admin.html
app.use((req, res, next) => {
  if (req.path === '/admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing beers file');
    } else {
      res.send('Beers updated successfully');
    }
  });
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

```

HTML File (^index.html^)

Ensure your `index.html` file references the `script.js` correctly:

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <div id="admin-panel">
        <button id="edit-button">Edit</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java
```

```
taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── login.html
├── styles.css
└── script.js
beers.json
```

Run the Server and Test

1. Install Dependencies: Ensure you have installed the required dependencies by running:

```
bash
```

```
npm install express body-parser express-session
```

2. Start the Server: Run the server using:

```
bash
```

```
node server.js
```

3. Open the Page: Navigate to `http://localhost:3000/index.html`.

This should properly separate the client-side and server-side logic, and allow the beers list to display correctly with the specified formatting. Let me know if you encounter any issues or need further assistance!



Can you give me that last response again with the code in boxes with a copy button



Certainly! Here is the response with the code in boxes and a copy button for easier copying.

Client-Side Script (`script.js`)

```
javascript

document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');
  let beers = [];

  const loadBeers = async () => {
    try {
      const response = await fetch('beers.json');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      beers = await response.json();
      displayBeers();
    } catch (error) {
      console.error('Failed to load beers:', error);
    }
  };

  const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach((beer, index) => {
      const beerElement = document.createElement('div');
      beerElement.classList.add('beer');
      beerElement.innerHTML = `
        <p>
          <span style="font-size: 29pt;">${index + 1}. &nbsp;</span>
          <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
          <span style="font-size: 21pt; color: black;"> by <b>${beer.brewery} - ${beer.type} - ${beer.abv}</b></span>
        </p>
      `;
      tapList.appendChild(beerElement);
    });
  };

  if (editButton) {
    editButton.addEventListener('click', () => {
      if (editButton.textContent === 'Edit') {
        editButton.textContent = 'Save';
        const beerElements = document.querySelectorAll('.beer');
        beerElements.forEach(beerElement, index) => {
          beerElement.innerHTML =
            <input type="text" value="${beers[index].name}" class="beer-name">
        };
      }
    });
  }
};
```

```

        <input type="text" value="${beers[index].brewery}" class="brewery">
        <input type="text" value="${beers[index].type}" class="type">
        <input type="text" value="${beers[index].abv}" class="abv">
    `;
}
} else {
    editButton.textContent = 'Edit';
    const beerElements = document.querySelectorAll('.beer');
    beerElements.forEach((beerElement, index) => {
        const name = beerElement.querySelector('.beer-name').value;
        const brewery = beerElement.querySelector('.brewery').value;
        const type = beerElement.querySelector('.type').value;
        const abv = beerElement.querySelector('.abv').value;
        beers[index] = { name, brewery, type, abv };
    });
    displayBeers();
}
});
```

loadBeers();

Server-Side Script (`server.js`)

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const
```



My mistake, the code was in boxes with copy buttons, it was a problem with my browser



No problem at all! Here's the response again with the code sections for easier copying.

Client-Side Script (`script.js`)

```
javascript

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const adminPanel = document.getElementById('admin-panel');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
```

```

        throw new Error(`HTTP error! status: ${response.status}`);
    }
    beers = await response.json();
    displayBeers();
} catch (error) {
    console.error('Failed to load beers:', error);
}
};

const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach((beer, index) => {
        const beerElement = document.createElement('div');
        beerElement.classList.add('beer');
        beerElement.innerHTML = `
            <p>
                <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
                <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                    <span style="font-size: 21pt; color: black;"> by <b>${beer.brewery} - ${beer.type} - ${beer.abv}</b></span>
                </p>
            `;
        tapList.appendChild(beerElement);
    });
}

if (editButton) {
    editButton.addEventListener('click', () => {
        if (editButton.textContent === 'Edit') {
            editButton.textContent = 'Save';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                beerElement.innerHTML =
                    <input type="text" value="${beers[index].name}" class="beer-name">
                    <input type="text" value="${beers[index].brewery}" class="brewery">
                    <input type="text" value="${beers[index].type}" class="type">
                    <input type="text" value="${beers[index].abv}" class="abv">
                ;
            });
        } else {
            editButton.textContent = 'Edit';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                const name = beerElement.querySelector('.beer-name').value;
                const brewery = beerElement.querySelector('.brewery').value;
                const type = beerElement.querySelector('.type').value;
                const abv = beerElement.querySelector('.abv').value;
                beers[index] = { name, brewery, type, abv };
            });
            displayBeers();
        }
    });
}

loadBeers();
});

```

Server-Side Script (`server.js`)

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;

```

```

const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing beers file');
        } else {
            res.send('Beers updated successfully');
        }
    });
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

```

HTML File (^index.html^)

Ensure your `index.html` file references the `script.js` correctly:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <div id="admin-panel">
        <button id="edit-button">Edit</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java

taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── login.html
├── styles.css
└── script.js
beers.json
```

Run the Server and Test

- 1. Install Dependencies:** Ensure you have installed the required dependencies by running:

```
bash

npm install express body-parser express-session
```

- 2. Start the Server:** Run the server using:

```
bash
```

```
node server.js
```

3. **Open the Page:** Navigate to `http://localhost:3000/index.html`.

This should properly separate the client-side and server-side logic, and allow the beers list to display correctly with the specified formatting. Let me know if you encounter any issues or need further assistance!



in styles.css, can you change the color of the Light red background for even items in #f58d72
and change the Light yellow background for odd items to #fcda81



Sure! Here is the updated `styles.css` file with the new background colors for the odd and even items:

Updated `styles.css`

```
css
```

```
body {
    font-family: Arial, sans-serif;
    background-color: #000; /* Black background */
    color: #fff; /* White text color */
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    border-bottom: 1px solid #444;
    padding: 10px 0;
    background-color: #fcda81; /* Light yellow background for odd items */
    color: #000; /* Black text color for better contrast */
}
```

```
.beer:nth-child(even) {
    background-color: #f58d72; /* Light red background for even items */
}

.beer p {
    margin: 0;
    padding: 10px;
    display: flex;
    align-items: center;
}
```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java

taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── login.html
├── styles.css
└── script.js
beers.json
```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display with the new background colors for odd and even items. Let me know if you need any further adjustments!



would you please remove the black line separating each item, also put a space between the beer name and the word by



Sure! Here's the updated `styles.css` to remove the black line separating each item and adjust the formatting to include a space between the beer name and the word "by":

Updated `styles.css`

```
css

body {
    font-family: Arial, sans-serif;
```

```

background-color: #000; /* Black background */
color: #fff; /* White text color */
margin: 0;
padding: 0;
display: flex;
flex-direction: column;
align-items: center;
justify-content: start;
}

.banner {
  width: 100%;
  text-align: center;
  margin: 20px 0;
}

#business-name {
  max-width: 100%;
  height: auto;
}

.tap-list {
  width: 90%;
  margin: 0 auto;
}

.beer {
  padding: 10px 0;
  background-color: #fcda81; /* Light yellow background for odd items */
  color: #000; /* Black text color for better contrast */
}

.beer:nth-child(even) {
  background-color: #f58d72; /* Light red background for even items */
}

.beer p {
  margin: 0;
  padding: 10px;
  display: flex;
  align-items: center;
}

```

Updated `script.js`

Ensure that the space between the beer name and the word "by" is included in the client-side script.

```

javascript

document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const adminPanel = document.getElementById('admin-panel');
  let beers = [];

  const loadBeers = async () => {
    try {
      const response = await fetch('beers.json');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      beers = await response.json();
      displayBeers();
    } catch (error) {
      console.error('Failed to load beers:', error);
    }
  };

  const displayBeers = () => {

```

```

tapList.innerHTML = '';
beers.forEach((beer, index) => {
    const beerElement = document.createElement('div');
    beerElement.classList.add('beer');
    beerElement.innerHTML = `
        <p>
            <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
            <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                <span style="font-size: 21pt; color: black;"> by <b>${beer.brewery} - ${beer.type} - ${beer.abv}</b></span>
        </p>
    `;
    tapList.appendChild(beerElement);
});
};

if (editButton) {
    editButton.addEventListener('click', () => {
        if (editButton.textContent === 'Edit') {
            editButton.textContent = 'Save';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                beerElement.innerHTML = `
                    <input type="text" value="${beers[index].name}" class="beer-name">
                    <input type="text" value="${beers[index].brewery}" class="brewery">
                    <input type="text" value="${beers[index].type}" class="type">
                    <input type="text" value="${beers[index].abv}" class="abv">
                `;
            });
        } else {
            editButton.textContent = 'Edit';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                const name = beerElement.querySelector('.beer-name').value;
                const brewery = beerElement.querySelector('.brewery').value;
                const type = beerElement.querySelector('.type').value;
                const abv = beerElement.querySelector('.abv').value;
                beers[index] = { name, brewery, type, abv };
            });
            displayBeers();
        }
    });
}
loadBeers();
});

```

Ensure File Structure

Make sure your project directory structure is as follows:

```

java
taplist/
├── package.json
└── package-lock.json
└── server.js
└── public/
    ├── index.html
    ├── admin.html
    ├── login.html
    ├── styles.css
    └── script.js
    └── beers.json

```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display without the black line separating each item, and with a space between the beer name and the word "by." Let me know if you need any further adjustments!



I need more of a space between the beer name and the word "by"



Sure! We can add some margin or padding to increase the space between the beer name and the word "by". Here's how you can do it:

Updated `script.js`

We'll add a non-breaking space (` `) for extra spacing between the beer name and the word "by".

```
javascript

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const adminPanel = document.getElementById('admin-panel');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <p>
                    <span style="font-size: 29pt;">${index + 1}.&ampnbsp&ampnbsp</span>
                    <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}&ampnbsp&ampnbsp</span>
                    <span style="font-size: 21pt; color: black;"> by <b>${beer.brewery} - ${beer.type} - ${beer.abv}</b></span>
                </p>
            `;
            tapList.appendChild(beerElement);
        });
    };
};
```

```

if (editButton) {
    editButton.addEventListener('click', () => {
        if (editButton.textContent === 'Edit') {
            editButton.textContent = 'Save';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                beerElement.innerHTML =
                    `>
                     <input type="text" value="${beers[index].brewery}" class="brewery">>
                     <input type="text" value="${beers[index].type}" class="type">>
                     <input type="text" value="${beers[index].abv}" class="abv">>
                `;
            });
        } else {
            editButton.textContent = 'Edit';
            const beerElements = document.querySelectorAll('.beer');
            beerElements.forEach((beerElement, index) => {
                const name = beerElement.querySelector('.beer-name').value;
                const brewery = beerElement.querySelector('.brewery').value;
                const type = beerElement.querySelector('.type').value;
                const abv = beerElement.querySelector('.abv').value;
                beers[index] = { name, brewery, type, abv };
            });
            displayBeers();
        }
    });
}
loadBeers();
});

```

Ensure File Structure

Make sure your project directory structure is as follows:

```

java
taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── login.html
├── styles.css
└── script.js
beers.json

```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display with more space between the beer name and the word "by." Let me know if you need any further adjustments!



That is too much space. I need the standard space amount, as if you were to push the spacebar on the keyboard



Got it. We'll use the standard space amount by ensuring there are spaces between the elements in the JavaScript string without using non-breaking spaces. Here's the updated `script.js`:

Updated `script.js`

javascript

```
document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const adminPanel = document.getElementById('admin-panel');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <p>
                    <span style="font-size: 29pt;">${index + 1}. &nbsp;</span>
                    <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                    <span style="font-size: 21pt; color: black;"> by <b>${beer.brewery} - ${beer.type} - ${beer.abv}</b></span>
                </p>
            `;
            tapList.appendChild(beerElement);
        });
    };

    if (editButton) {
        editButton.addEventListener('click', () => {
            if (editButton.textContent === 'Edit') {
                editButton.textContent = 'Save';
                const beerElements = document.querySelectorAll('.beer');
                beerElements.forEach((beerElement, index) => {
                    beerElement.innerHTML = `
                        <input type="text" value="${beers[index].name}" class="beer-name">
                        <input type="text" value="${beers[index].brewery}" class="brewery">
                        <input type="text" value="${beers[index].type}" class="type">
                        <input type="text" value="${beers[index].abv}" class="abv">
                    `;
                });
            }
        });
    }
};
```

```

    } else {
      editButton.textContent = 'Edit';
      const beerElements = document.querySelectorAll('.beer');
      beerElements.forEach((beerElement, index) => {
        const name = beerElement.querySelector('.beer-name').value;
        const brewery = beerElement.querySelector('.brewery').value;
        const type = beerElement.querySelector('.type').value;
        const abv = beerElement.querySelector('.abv').value;
        beers[index] = { name, brewery, type, abv };
      });
      displayBeers();
    });
  );
}

loadBeers();
);

```

Ensure File Structure

Make sure your project directory structure is as follows:

```

java

taplist/
├── package.json
├── package-lock.json
└── server.js
└── public/
    ├── index.html
    ├── admin.html
    ├── login.html
    ├── styles.css
    ├── script.js
    └── beers.json

```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display with the standard space amount between the beer name and the word "by." Let me know if you need any further adjustments!



for some reason there is still no space between the beer name and the word by



Let's ensure there is a space between the beer name and the word "by" by explicitly adding a space in the string.

Updated `script.js`

Here is the updated script to ensure the space is correctly added:

```
javascript

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const adminPanel = document.getElementById('admin-panel');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <p>
                    <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
                    <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                    <span style="font-size: 21pt; color: black;">&nbsp;by <b>${beer.brewery}</b>
                    - ${beer.type} - ${beer.abv}</span>
                </p>
            `;
            tapList.appendChild(beerElement);
        });
    };

    if (editButton) {
        editButton.addEventListener('click', () => {
            if (editButton.textContent === 'Edit') {
                editButton.textContent = 'Save';
                const beerElements = document.querySelectorAll('.beer');
                beerElements.forEach((beerElement, index) => {
                    beerElement.innerHTML = `
                        <input type="text" value="${beers[index].name}" class="beer-name">
                        <input type="text" value="${beers[index].brewery}" class="brewery">
                        <input type="text" value="${beers[index].type}" class="type">
                        <input type="text" value="${beers[index].abv}" class="abv">
                    `;
                });
            } else {
                editButton.textContent = 'Edit';
                const beerElements = document.querySelectorAll('.beer');
                beerElements.forEach((beerElement, index) => {
                    const name = beerElement.querySelector('.beer-name').value;
                    const brewery = beerElement.querySelector('.brewery').value;
                    const type = beerElement.querySelector('.type').value;
                    const abv = beerElement.querySelector('.abv').value;
                    beers[index] = { name, brewery, type, abv };
                });
                displayBeers();
            }
        });
    };
});
```

```
    loadBeers();
});
```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java

taplist/
└── package.json
└── package-lock.json
└── server.js
└── public/
    ├── index.html
    ├── admin.html
    ├── login.html
    ├── styles.css
    └── script.js
    └── beers.json
```

Run the Server and Test

1. **Start the Server:** Ensure your server is running with `node server.js`.
2. **Open the Page:** Navigate to `http://localhost:3000/index.html`.

The beers list should now display with the standard space amount between the beer name and the word "by." Let me know if you need any further adjustments!



Works great. Now we need to make our admin page reflect our update index.html layout, and make index.html editable via admin.html



Sure, let's update the `admin.html` page and its script to reflect the updated layout of `index.html` and make the `index.html` editable through `admin.html`.

Updated `admin.html`

This file will have a similar structure to `index.html`, but will include form fields to edit the beer entries.

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Tap List</title>
```

```

<link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="banner">
    
  </div>
  <div class="tap-list" id="tap-list">
    <!-- Beers will be dynamically inserted here -->
  </div>
  <div id="admin-panel">
    <button id="edit-button">Edit</button>
    <button id="save-button" style="display:none;">Save</button>
  </div>
  <script src="admin-script.js"></script>
</body>
</html>

```

New `admin-script.js`

This script will handle loading the beer list and providing edit functionality.

javascript

```

document.addEventListener('DOMContentLoaded', () => {
  const tapList = document.getElementById('tap-list');
  const editButton = document.getElementById('edit-button');
  const saveButton = document.getElementById('save-button');
  let beers = [];

  const loadBeers = async () => {
    try {
      const response = await fetch('beers.json');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      beers = await response.json();
      displayBeers();
    } catch (error) {
      console.error('Failed to load beers:', error);
    }
  };

  const displayBeers = () => {
    tapList.innerHTML = '';
    beers.forEach((beer, index) => {
      const beerElement = document.createElement('div');
      beerElement.classList.add('beer');
      beerElement.innerHTML = `
        <p>
          <span style="font-size: 29pt;">${index + 1}. &nbsp;</span>
          <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
          <span style="font-size: 21pt; color: black;">&nbsp;by <b>${beer.brewery}</b>
          - ${beer.type} - ${beer.abv}</span>
        </p>
      `;
      tapList.appendChild(beerElement);
    });
  };

  const enableEditing = () => {
    tapList.innerHTML = '';
    beers.forEach((beer, index) => {
      const beerElement = document.createElement('div');
      beerElement.classList.add('beer');
      beerElement.innerHTML = `
        <p>
          <span style="font-size: 29pt;">${index + 1}. &nbsp;</span>
          <input type="text" value="${beer.name}" class="beer-name" style="font-

```

```

size: 29pt; color: black; font-weight: bold;">>
    <span style="font-size: 21pt; color: black;">&nbsp;by </span>
    <input type="text" value="${beer.brewery}" class="brewery" style="font-
size: 21pt; color: black;">>
        <input type="text" value="${beer.type}" class="type" style="font-size:
21pt; color: black;">>
            <input type="text" value="${beer.abv}" class="abv" style="font-size:
21pt; color: black;">>
                </p>
            `;
        tapList.appendChild(beerElement);
    });
};

const saveBeers = async () => {
    const beerElements = document.querySelectorAll('.beer');
    beerElements.forEach((beerElement, index) => {
        const name = beerElement.querySelector('.beer-name').value;
        const brewery = beerElement.querySelector('.brewery').value;
        const type = beerElement.querySelector('.type').value;
        const abv = beerElement.querySelector('.abv').value;
        beers[index] = { name, brewery, type, abv };
    });
}

try {
    const response = await fetch('/update-beers', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(beers)
    });
    if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
    }
    alert('Beers updated successfully');
    displayBeers();
} catch (error) {
    console.error('Failed to save beers:', error);
    alert('Failed to save beers');
}
};

editButton.addEventListener('click', () => {
    editButton.style.display = 'none';
    saveButton.style.display = 'inline';
    enableEditing();
});

saveButton.addEventListener('click', () => {
    editButton.style.display = 'inline';
    saveButton.style.display = 'none';
    saveBeers();
});

loadBeers();
});

```

Updated `server.js`

Ensure the server is set up to serve the `beers.json` file and handle updates.

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');

```

```

const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 'your_secret_key', // Replace with a secure key
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedin) {
    next();
  } else {
    res.redirect('/login.html');
  }
};

// Serve static files from public directory, but protect admin.html
app.use((req, res, next) => {
  if (req.path === '/admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedin = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing beers file');
    } else {
      res.send('Beers updated successfully');
    }
  });
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java
taplist/
├── package.json
├── package-lock.json
└── server.js
└── public/
    ├── index.html
    ├── admin.html
    ├── login.html
    ├── styles.css
    ├── script.js
    └── admin-script.js
└── beers.json
```

Run the Server and Test

1. **Start the Server:** Ensure your server is running with `node server.js`.
2. **Open the Admin Page:** Navigate to `http://localhost:3000/login.html` and log in.
3. **Edit the Beers:** Make changes in `admin.html` and save them.
4. **Check the Index Page:** Navigate to `http://localhost:3000/index.html` to see the updated beer list.

This setup should allow you to edit the beer list in `admin.html` and see the updates in `index.html`. Let me know if you need any further adjustments!



It looks like when the admin.html page is updating index, it's also adding the edit button to index.html which obviously we don't want since it allows anyone to edit the index page



To ensure the `index.html` page does not display the edit button, we need to differentiate the admin and user functionalities in our scripts. We will also make sure that the edit button and related functionality are exclusive to `admin.html` .

Updated `index.html`

Remove any references to the edit button from `index.html` .

```
html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```

<title>Tap List</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script src="script.js"></script>
</body>
</html>

```

Updated `script.js` for `index.html`

Ensure the script for `index.html` only loads and displays the beers without any edit functionality.

javascript

```

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <p>
                    <span style="font-size: 29pt;">${index + 1}. &nbsp;</span>
                    <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                    <span style="font-size: 21pt; color: black;">&nbsp;by <b>${beer.brewery}</b>
                    - ${beer.type} - ${beer.abv}</span>
                </p>
            `;
            tapList.appendChild(beerElement);
        });
        loadBeers();
    };
});

```

Updated `admin.html`

Include the edit button and its related functionality in `admin.html`.

html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <div id="admin-panel">
        <button id="edit-button">Edit</button>
        <button id="save-button" style="display:none;">Save</button>
    </div>
    <script src="admin-script.js"></script>
</body>
</html>
```

New `admin-script.js` for `admin.html`

This script handles loading, editing, and saving the beer list.

javascript

```
document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const saveButton = document.getElementById('save-button');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <p>
                    <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
                    <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                    <span style="font-size: 21pt; color: black;">&nbsp;by <b>${beer.brewery}</b>
                    - ${beer.type} - ${beer.abv}</b></span>
                </p>
            `;
            tapList.appendChild(beerElement);
        });
    };
});
```

```

const enableEditing = () => {
    tapList.innerHTML = '';
    beers.forEach((beer, index) => {
        const beerElement = document.createElement('div');
        beerElement.classList.add('beer');
        beerElement.innerHTML = `
            <p>
                <span style="font-size: 29pt;">${index + 1} </span>
                <input type="text" value="${beer.name}" class="beer-name" style="font-size: 29pt; color: black; font-weight: bold;">
                <span style="font-size: 21pt; color: black;">by </span>
                <input type="text" value="${beer.brewery}" class="brewery" style="font-size: 21pt; color: black;">
                <input type="text" value="${beer.type}" class="type" style="font-size: 21pt; color: black;">
                <input type="text" value="${beer.abv}" class="abv" style="font-size: 21pt; color: black;">
            </p>
        `;
        tapList.appendChild(beerElement);
    });
};

const saveBeers = async () => {
    const beerElements = document.querySelectorAll('.beer');
    beerElements.forEach((beerElement, index) => {
        const name = beerElement.querySelector('.beer-name').value;
        const brewery = beerElement.querySelector('.brewery').value;
        const type = beerElement.querySelector('.type').value;
        const abv = beerElement.querySelector('.abv').value;
        beers[index] = { name, brewery, type, abv };
    });
}

try {
    const response = await fetch('/update-beers', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(beers)
    });
    if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
    }
    alert('Beers updated successfully');
    displayBeers();
} catch (error) {
    console.error('Failed to save beers:', error);
    alert('Failed to save beers');
}
};

editButton.addEventListener('click', () => {
    editButton.style.display = 'none';
    saveButton.style.display = 'inline';
    enableEditing();
});

saveButton.addEventListener('click', () => {
    editButton.style.display = 'inline';
    saveButton.style.display = 'none';
    saveBeers();
});

loadBeers();
});

```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java
```

```
taplist/
  package.json
  package-lock.json
  server.js
  public/
    index.html
    admin.html
    login.html
    styles.css
    script.js
    admin-script.js
    beers.json
```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Admin Page:** Navigate to `http://localhost:3000/login.html` and log in.
- Edit the Beers:** Make changes in `admin.html` and save them.
- Check the Index Page:** Navigate to `http://localhost:3000/index.html` to see the updated beer list.

This setup should ensure that the `index.html` page does not include the edit button, while allowing `admin.html` to edit and update the beer list. Let me know if you need any further adjustments!



On the admin.html page, can we move the edit button to the top of the page and on the left side instead of centered?



Sure! We'll adjust the positioning of the edit button in the `admin.html` page. We'll use CSS to position the button at the top left of the page.

Updated `admin.html`

Move the edit button to the top left:

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Admin Tap List</title>
  <link rel="stylesheet" href="styles.css">
  <style>
    #admin-panel {
      position: absolute;
      top: 10px;
      left: 10px;
```

```

        }
    </style>
</head>
<body>
    <div class="banner">
        
    </div>
    <div id="admin-panel">
        <button id="edit-button">Edit</button>
        <button id="save-button" style="display:none;">Save</button>
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script src="admin-script.js"></script>
</body>
</html>

```

Ensure File Structure

Make sure your project directory structure is as follows:

```

java
taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── login.html
├── styles.css
├── script.js
└── admin-script.js
beers.json

```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Admin Page:** Navigate to `http://localhost:3000/login.html` and log in.
- Edit the Beers:** Make changes in `admin.html` and save them.
- Check the Index Page:** Navigate to `http://localhost:3000/index.html` to see the updated beer list.

This setup should now place the edit button at the top left of the `admin.html` page. Let me know if you need any further adjustments!



Can we make a separate admin panel too called color-admin for modifying the line backgrounds for odd and even lines on index.html? I'd like to be able to enter a hex color code from the panel and it changes the line's background color to that



Sure! We can create a separate admin panel called `color-admin.html` for modifying the line background colors for odd and even lines on `index.html`. This will involve creating a new HTML file and a corresponding JavaScript file to handle the color changes. We will also need to update the server to handle saving the new colors.

Updated File Structure

Here is how the updated file structure will look:

```
java

taplist/
├── package.json
├── package-lock.json
└── server.js
└── public/
    ├── index.html
    ├── admin.html
    ├── login.html
    ├── color-admin.html
    ├── styles.css
    ├── script.js
    ├── admin-script.js
    ├── color-admin-script.js
    └── beers.json
        └── colors.json
```

Create `color-admin.html`

Create a new HTML file for the color admin panel:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Color Admin</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        #color-admin-panel {
            position: absolute;
            top: 10px;
            left: 10px;
        }
    </style>
</head>
<body>
    <div id="color-admin-panel">
        <label for="odd-color">Odd Line Color:</label>
        <input type="text" id="odd-color" placeholder="#fcda81">
        <label for="even-color">Even Line Color:</label>
        <input type="text" id="even-color" placeholder="#f58d72">
        <button id="save-colors-button">Save Colors</button>
    </div>
    <script src="color-admin-script.js"></script>
</body>
</html>
```

Create `color-admin-script.js`

Create a new JavaScript file for handling the color admin panel logic:

```
javascript

document.addEventListener('DOMContentLoaded', () => {
    const oddColorInput = document.getElementById('odd-color');
    const evenColorInput = document.getElementById('even-color');
    const saveColorsButton = document.getElementById('save-colors-button');

    const loadColors = async () => {
        try {
            const response = await fetch('colors.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            const colors = await response.json();
            oddColorInput.value = colors.odd;
            evenColorInput.value = colors.even;
        } catch (error) {
            console.error('Failed to load colors:', error);
        }
    };

    const saveColors = async () => {
        const colors = {
            odd: oddColorInput.value,
            even: evenColorInput.value
        };

        try {
            const response = await fetch('/update-colors', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(colors)
            });
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            alert('Colors updated successfully');
        } catch (error) {
            console.error('Failed to save colors:', error);
            alert('Failed to save colors');
        }
    };
    saveColorsButton.addEventListener('click', saveColors);
    loadColors();
});
});
```

Update `styles.css`

Update the CSS to apply the colors dynamically using CSS variables:

```
css

:root {
    --odd-line-color: #fcda81; /* Default color */
    --even-line-color: #f58d72; /* Default color */
}
```

```

body {
  font-family: Arial, sans-serif;
  background-color: #000; /* Black background */
  color: #fff; /* White text color */
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: start;
}

.banner {
  width: 100%;
  text-align: center;
  margin: 20px 0;
}

#business-name {
  max-width: 100%;
  height: auto;
}

.tap-list {
  width: 90%;
  margin: 0 auto;
}

.beer {
  padding: 10px 0;
  background-color: var(--odd-line-color); /* Light yellow background for odd items */
  color: #000; /* Black text color for better contrast */
}

.beer:nth-child(even) {
  background-color: var(--even-line-color); /* Light red background for even items */
}

.beer p {
  margin: 0;
  padding: 10px;
  display: flex;
  align-items: center;
}

```

Update `server.js`

Add a new endpoint to handle color updates and serve the `colors.json` file:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 'your_secret_key', // Replace with a secure key
  resave: false,
  saveUninitialized: true
})

```

```

});;

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing beers file');
        } else {
            res.send('Beers updated successfully');
        }
    });
});

app.post('/update-colors', isAuthenticated, (req, res) => {
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing colors file');
        } else {
    
```

```

        res.send('Colors updated successfully');
    });
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

```

Create `colors.json`

Create a new JSON file to store the colors:

json

```
{
    "odd": "#fcda81",
    "even": "#f58d72"
}
```

Update `script.js` for `index.html` to Load Colors

Ensure the script for `index.html` loads the colors from `colors.json`:

javascript

```

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const loadColors = async () => {
        try {
            const response = await fetch('colors.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            const colors = await response.json();
            document.documentElement.style.setProperty('--odd-line-color', colors.odd);
            document.documentElement.style.setProperty('--even-line-color', colors.even);
        } catch (error) {
            console.error('Failed to load colors:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `

```

```

        <p>
            <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
            <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
            <span style="font-size: 21pt; color: black;">&nbsp;by <b>${beer.brewery}</b>
            - ${beer.type} - ${beer.abv}</b></span>
        </p>
    `;
    tapList.appendChild(beerElement);
};

loadColors();
loadBeers();
});

```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Color Admin Page:** Navigate to `http://localhost:3000/color-admin.html` and log in.
- Change Colors:** Enter new hex color codes and save them.
- Check the Index Page:** Navigate to `http://localhost:3000/index.html` to see the updated line colors.

This setup should now allow you to modify the line background colors for odd and even lines on `index.html` using the `color-admin.html` page. Let me know if you need any further adjustments!



In the color admin panel can we make it so the text color is modifiable too via hex color code



Sure! We can extend the `color-admin.html` and `color-admin-script.js` to include an option for modifying the text color as well.

Updated `color-admin.html`

Add input fields for the text color:

```

html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Color Admin</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        #color-admin-panel {
            position: absolute;
            top: 10px;
            left: 10px;
        }
    </style>
</head>

```

```

<body>
  <div id="color-admin-panel">
    <label for="odd-color">Odd Line Color:</label>
    <input type="text" id="odd-color" placeholder="#fcda81">
    <label for="even-color">Even Line Color:</label>
    <input type="text" id="even-color" placeholder="#f58d72">
    <label for="text-color">Text Color:</label>
    <input type="text" id="text-color" placeholder="#000000">
    <button id="save-colors-button">Save Colors</button>
  </div>
  <script src="color-admin-script.js"></script>
</body>
</html>

```

Updated `color-admin-script.js`

Add functionality to save the text color:

```

javascript

document.addEventListener('DOMContentLoaded', () => {
  const oddColorInput = document.getElementById('odd-color');
  const evenColorInput = document.getElementById('even-color');
  const textColorInput = document.getElementById('text-color');
  const saveColorsButton = document.getElementById('save-colors-button');

  const loadColors = async () => {
    try {
      const response = await fetch('colors.json');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      const colors = await response.json();
      oddColorInput.value = colors.odd;
      evenColorInput.value = colors.even;
      textColorInput.value = colors.text;
    } catch (error) {
      console.error('Failed to load colors:', error);
    }
  };

  const saveColors = async () => {
    const colors = {
      odd: oddColorInput.value,
      even: evenColorInput.value,
      text: textColorInput.value
    };

    try {
      const response = await fetch('/update-colors', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(colors)
      });
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      alert('Colors updated successfully');
    } catch (error) {
      console.error('Failed to save colors:', error);
      alert('Failed to save colors');
    }
  };
}

saveColorsButton.addEventListener('click', saveColors);

```

```
    loadColors();
});
```

Updated `styles.css`

Update the CSS to apply the text color dynamically using CSS variables:

css

```
:root {
  --odd-line-color: #fcda81; /* Default color */
  --even-line-color: #f58d72; /* Default color */
  --text-color: #000000; /* Default text color */
}

body {
  font-family: Arial, sans-serif;
  background-color: #000; /* Black background */
  color: var(--text-color); /* Text color */
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: start;
}

.banner {
  width: 100%;
  text-align: center;
  margin: 20px 0;
}

#business-name {
  max-width: 100%;
  height: auto;
}

.tap-list {
  width: 90%;
  margin: 0 auto;
}

.beer {
  padding: 10px 0;
  background-color: var(--odd-line-color); /* Light yellow background for odd items */
  color: var(--text-color); /* Text color */
}

.beer:nth-child(even) {
  background-color: var(--even-line-color); /* Light red background for even items */
}

.beer p {
  margin: 0;
  padding: 10px;
  display: flex;
  align-items: center;
  color: var(--text-color); /* Text color */
}
```

Update `server.js`

Ensure the server can handle saving and serving the text color:

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {

```

```

        res.status(500).send('Error reading colors file');
    } else {
        res.json(JSON.parse(data));
    }
});

app.post('/update-beers', isAuthenticated, (req, res) => {
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing beers file');
        } else {
            res.send('Beers updated successfully');
        }
    });
});

app.post('/update-colors', isAuthenticated, (req, res) => {
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing colors file');
        } else {
            res.send('Colors updated successfully');
        }
    });
});

app.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

```

Updated `colors.json`

Ensure the `colors.json` file includes a field for the text color:

```
json

{
    "odd": "#fcda81",
    "even": "#f58d72",
    "text": "#000000"
}
```

Update `script.js` for `index.html` to Load Text Color

Ensure the script for `index.html` loads the text color from `colors.json`:

```
javascript

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    let beers = [];

    const loadBeers = async () => {
        try {
            const response = await fetch('beers.json');
            if (!response.ok) {
                throw new Error(`HTTP error! status: ${response.status}`);
            }
            beers = await response.json();
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };
    loadBeers();
});
```

```

    }
};

const loadColors = async () => {
  try {
    const response = await fetch('colors.json');
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    const colors = await response.json();
    document.documentElement.style.setProperty('--odd-line-color', colors.odd);
    document.documentElement.style.setProperty('--even-line-color', colors.even);
    document.documentElement.style.setProperty('--text-color', colors.text);
  } catch (error) {
    console.error('Failed to load colors:', error);
  }
};

const displayBeers = () => {
  tapList.innerHTML = '';
  beers.forEach((beer, index) => {
    const beerElement = document.createElement('div');
    beerElement.classList.add('beer');
    beerElement.innerHTML = `
      <p>
        <span style="font-size: 29pt;">${index + 1}.&nbsp;&nbsp;</span>
        <span style="font-size: 29pt; font-weight: bold;">${beer.name}</span>
        <span style="font-size: 21pt;">&nbsp;by <b>${beer.brewery} - 
${beer.type} - ${beer.abv}</b></span>
      </p>
    `;
    tapList.appendChild(beerElement);
  });
}

loadColors();
loadBeers();
});

```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Color Admin Page:** Navigate to `http://localhost:3000/color-admin.html` and log in.
- Change Colors:** Enter new hex color codes for the odd line color, even line color, and text color, and save them.
- Check the Index Page:** Navigate to `http://localhost:3000/index.html` to see the updated line colors and text color.

This setup should now allow you to modify the line background colors for odd and even lines as well as the text color on `index.html` using the `color-admin.html` page. Let me know if you need any further adjustments!



Would you please make the background on the color admin page gray and not black



Sure! We can update the CSS in the `color-admin.html` file to set the background color to gray.

Updated `color-admin.html`

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Color Admin</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        body {
            background-color: #808080; /* Gray background */
            color: #000; /* Black text color */
        }
        #color-admin-panel {
            position: absolute;
            top: 10px;
            left: 10px;
        }
    </style>
</head>
<body>
    <div id="color-admin-panel">
        <label for="odd-color">Odd Line Color:</label>
        <input type="text" id="odd-color" placeholder="#fcda81">
        <label for="even-color">Even Line Color:</label>
        <input type="text" id="even-color" placeholder="#f58d72">
        <label for="text-color">Text Color:</label>
        <input type="text" id="text-color" placeholder="#000000">
        <button id="save-colors-button">Save Colors</button>
    </div>
    <script src="color-admin-script.js"></script>
</body>
</html>
```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java

taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── login.html
├── color-admin.html
├── styles.css
├── script.js
├── admin-script.js
├── color-admin-script.js
└── beers.json
    └── colors.json
```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.

2. **Open the Color Admin Page:** Navigate to `http://localhost:3000/color-admin.html` and log in.
3. **Change Colors:** Enter new hex color codes for the odd line color, even line color, and text color, and save them.
4. **Check the Index Page:** Navigate to `http://localhost:3000/index.html` to see the updated line colors and text color.

This setup should now have a gray background on the `color-admin.html` page. Let me know if you need any further adjustments!



The question is now, how do we make this web page easily editable using the admin.html file from a mobile device that isn't hosting this web page, but is on the same LAN as the device hosting this web page



To make the webpage easily editable from a mobile device on the same LAN as the device hosting the webpage, follow these steps:

1. Ensure the Server is Accessible on the LAN

First, make sure your server is accessible on the local network. You need to know the IP address of the device hosting the server.

1. Find the Local IP Address:

- On Windows, you can use `ipconfig` in the Command Prompt.
- On Mac or Linux, you can use `ifconfig` in the Terminal.
- Look for the IP address in the output. It will usually be in the format `192.168.x.x` or `10.0.x.x`.

2. Update `server.js` to Listen on All Network Interfaces:

Ensure the server is set to listen on all network interfaces by specifying `0.0.0.0` as the host in the `listen` method.

Updated `server.js`

javascript

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const app = express();

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
```

```

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 'your_secret_key', // Replace with a secure key
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
  if (req.path === '/admin.html' || req.path === '/color-admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing beers file');
    } else {
      res.send('Beers updated successfully');
    }
  });
});

```

```

app.post('/update-colors', isAuthenticated, (req, res) => {
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing colors file');
    } else {
      res.send('Colors updated successfully');
    }
  });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

```

2. Access the Admin Pages from the Mobile Device

1. Start the Server:

Make sure the server is running on the host device:

```

bash

node server.js

```

2. Connect Mobile Device to the Same LAN:

Ensure that your mobile device is connected to the same Wi-Fi network as the server.

3. Open the Admin Page on Mobile Device:

Open a web browser on your mobile device and enter the IP address of the host device followed by the port number and the path to the admin page.

For example, if the host device's IP address is `192.168.1.100`, enter:

```

arduino

http://192.168.1.100:3000/login.html

```

4. Login to Access the Admin Pages:

- Use the login page to authenticate.
- Navigate to `admin.html` or `color-admin.html` to edit the beer list or colors.

3. Ensure Mobile-Friendly Design

Make sure the admin pages are mobile-friendly by using responsive design practices:

- Use a responsive meta tag in your HTML:

```
html
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

- Ensure input fields and buttons are easily tappable.

4. Example Admin Pages

Here's a recap of the admin HTML files with responsive design:

admin.html:

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Tap List</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        #admin-panel {
            position: absolute;
            top: 10px;
            left: 10px;
        }
    </style>
</head>
<body>
    <div class="banner">
        
    </div>
    <div id="admin-panel">
        <button id="edit-button">Edit</button>
        <button id="save-button" style="display:none;">Save</button>
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script src="admin-script.js"></script>
</body>
</html>
```

color-admin.html:

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Color Admin</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        body {
            background-color: #808080; /* Gray background */
            color: #000; /* Black text color */
        }
    </style>

```

```

#color-admin-panel {
    position: absolute;
    top: 10px;
    left: 10px;
}
</style>
</head>
<body>
    <div id="color-admin-panel">
        <label for="odd-color">Odd Line Color:</label>
        <input type="text" id="odd-color" placeholder="#fcda81">
        <label for="even-color">Even Line Color:</label>
        <input type="text" id="even-color" placeholder="#f58d72">
        <label for="text-color">Text Color:</label>
        <input type="text" id="text-color" placeholder="#000000">
        <button id="save-colors-button">Save Colors</button>
    </div>
    <script src="color-admin-script.js"></script>
</body>
</html>

```

By following these steps, you should be able to edit the webpage from a mobile device on the same LAN as the device hosting the server.



Is there any way to ensure connectivity even if the 1918 IP address of the hosting device changes



Yes, to ensure connectivity even if the local IP address of the hosting device changes, you have several options. Here are some common methods:

1. Dynamic DNS (DDNS)

Dynamic DNS (DDNS) allows you to map a domain name to a dynamic IP address, automatically updating the domain name with the current IP address of your device.

- 1. Sign up for a DDNS Service:** There are many free and paid DDNS services available (e.g., No-IP, DynDNS, DuckDNS).
- 2. Configure DDNS on Your Router:** Most routers support DDNS. You can find the DDNS configuration settings in your router's admin panel.
- 3. Access Your Device Using the Domain Name:** Once set up, you can access your server using a domain name (e.g., myserver.ddns.net), which will automatically resolve to your current IP address.

2. Static IP Address

Assign a static IP address to your hosting device to ensure it does not change. This is configured in your router's DHCP settings.

1. **Access Your Router's Admin Panel:** This is usually done by entering the router's IP address in a web browser.
2. **Find DHCP Settings:** Look for DHCP settings in your router's admin panel.
3. **Assign a Static IP:** Assign a static IP to your hosting device based on its MAC address.

3. mDNS (Multicast DNS)

mDNS allows you to access devices on your local network using a ` `.local` domain name. This is often used by devices like printers and is supported by most operating systems (e.g., Windows 10+, macOS, Linux).

1. **Ensure mDNS is Enabled:** Most modern operating systems have mDNS enabled by default.
2. **Access Your Device Using a .local Domain:** Access your device using its hostname followed by ` `.local` (e.g., ` http://myserver.local`).

4. Scripts to Automatically Update Configuration

You can create scripts to automatically update the configuration with the new IP address and notify you.

1. **Create a Script to Get the Current IP Address:** Use a script to get the current IP address of your hosting device.
2. **Update Configuration and Notify:** Use the script to update the configuration and notify you via email or another method.

Example of Using mDNS

If you want to use mDNS, you just need to ensure your device has a hostname configured. For example:

- **On Linux**, you can set the hostname using the `hostnamectl` command:

```
bash
sudo hostnamectl set-hostname myserver
```

- **On Windows 10+**, you can set the hostname in System Properties.

After setting the hostname, you can access your server using ` http://myserver.local` .

Example of a Script to Get the Current IP Address (Linux)

You can create a simple script that fetches the current IP address and updates a DNS record or sends an email:

bash

```
#!/bin/bash

# Get the current IP address
current_ip=$(hostname -I | awk '{print $1}')

# Specify the path to the configuration file
config_file="/path/to/config/file"

# Update the configuration file with the new IP address
sed -i "s/^server_ip=.*/server_ip=$current_ip/" $config_file

# Send an email notification (requires mailutils to be installed)
echo "The server IP address has changed to $current_ip" | mail -s "Server IP Updated" you@example.com
```

Conclusion

Using DDNS or mDNS is the most user-friendly and reliable method to ensure connectivity to your server even if the IP address changes. Setting a static IP address on your router is also a solid approach if your router supports it. Creating scripts for dynamic updates provides a more custom solution but requires additional setup and maintenance.

Choose the method that best suits your needs and infrastructure. Let me know if you need further assistance with any of these methods!



The webpage is now accessible on the network, but does not scale properly to a mobile device. Everything is very zoomed in



To make the webpage scale properly on mobile devices, we need to ensure that it uses responsive design principles. This typically involves using the viewport meta tag and applying CSS styles that adjust layout and sizing based on the screen size.

Add Viewport Meta Tag

First, ensure that the viewport meta tag is included in your HTML files (``index.html``, ``admin.html``, ``color-admin.html``):

```
html

<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Apply Responsive CSS

Next, update the CSS to ensure that elements scale properly on different screen sizes. We can use media queries to adjust styles for smaller screens.

Here's an updated version of `styles.css`:

```
css

:root {
    --odd-line-color: #fcda81; /* Default color */
    --even-line-color: #f58d72; /* Default color */
    --text-color: #000000; /* Default text color */
}

body {
    font-family: Arial, sans-serif;
    background-color: #000; /* Black background */
    color: var(--text-color); /* Text color */
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    padding: 10px 0;
    background-color: var(--odd-line-color); /* Light yellow background for odd items */
    color: var(--text-color); /* Text color */
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.beer:nth-child(even) {
    background-color: var(--even-line-color); /* Light red background for even items */
}

.beer p {
    margin: 0;
    padding: 10px;
    display: flex;
    align-items: center;
    color: var(--text-color); /* Text color */
    flex-wrap: wrap; /* Allow text to wrap on smaller screens */
}

/* Responsive styles */
@media (max-width: 600px) {
    .beer {
        flex-direction: column;
        align-items: flex-start;
    }
}
```

```
.beer p {
    font-size: 20px; /* Adjust font size for smaller screens */
}
```

Update HTML Files

Ensure each HTML file includes the viewport meta tag:

index.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script src="script.js"></script>
</body>
</html>
```

admin.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Tap List</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        #admin-panel {
            position: absolute;
            top: 10px;
            left: 10px;
        }
    </style>
</head>
<body>
    <div class="banner">
        
    </div>
    <div id="admin-panel">
        <button id="edit-button">Edit</button>
        <button id="save-button" style="display:none;">Save</button>
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script src="admin-script.js"></script>
```

```
</body>
</html>
```

color-admin.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Color Admin</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        body {
            background-color: #808080; /* Gray background */
            color: #000; /* Black text color */
        }
        #color-admin-panel {
            position: absolute;
            top: 10px;
            left: 10px;
        }
    </style>
</head>
<body>
    <div id="color-admin-panel">
        <label for="odd-color">Odd Line Color:</label>
        <input type="text" id="odd-color" placeholder="#fcda81">
        <label for="even-color">Even Line Color:</label>
        <input type="text" id="even-color" placeholder="#f58d72">
        <label for="text-color">Text Color:</label>
        <input type="text" id="text-color" placeholder="#000000">
        <button id="save-colors-button">Save Colors</button>
    </div>
    <script src="color-admin-script.js"></script>
</body>
</html>
```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Pages on Mobile Device:** Navigate to `http://your-server-ip:3000/index.html`, `admin.html`, and `color-admin.html` on your mobile device.
- Check the Layout:** Ensure the layout scales properly on the mobile device.

By including the viewport meta tag and applying responsive CSS styles, the webpage should now scale properly on mobile devices. Let me know if you need any further adjustments!



can we make the edit button 4 times bigger on the admin.html page?



Sure! We can increase the size of the edit button on the `admin.html` page by updating its CSS. Let's make the button 4 times bigger.

Updated `admin.html`

We'll add specific styles to increase the button size.

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Tap List</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        #admin-panel {
            position: absolute;
            top: 10px;
            left: 10px;
        }

        #admin-panel button {
            font-size: 2em; /* Increase font size */
            padding: 20px 40px; /* Increase padding */
        }
    </style>
</head>
<body>
    <div class="banner">
        
    </div>
    <div id="admin-panel">
        <button id="edit-button">Edit</button>
        <button id="save-button" style="display:none;">Save</button>
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script src="admin-script.js"></script>
</body>
</html>
```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java

taplist/
├── package.json
├── package-lock.json
└── server.js
public/
├── index.html
├── admin.html
├── login.html
├── color-admin.html
├── styles.css
├── script.js
├── admin-script.js
└── color-admin-script.js
```

```

└── beers.json
└── colors.json

```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Admin Page on Mobile Device:** Navigate to `http://your-server-ip:3000/admin.html` on your mobile device.
- Check the Button Size:** Ensure the edit button is now 4 times bigger.

This should make the edit button much larger and easier to tap on mobile devices. Let me know if you need any further adjustments!



On color-admin.html, can you add a page break after each of the text entry fields?



Sure! To add a page break (line break) after each of the text entry fields on `color-admin.html`, you can use the `
` tag. Here's the updated `color-admin.html`:

Updated `color-admin.html`

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Color Admin</title>
    <link rel="stylesheet" href="styles.css">
    <style>
        body {
            background-color: #808080; /* Gray background */
            color: #000; /* Black text color */
        }
        #color-admin-panel {
            position: absolute;
            top: 10px;
            left: 10px;
        }
    </style>
</head>
<body>
    <div id="color-admin-panel">
        <label for="odd-color">Odd Line Color:</label>
        <input type="text" id="odd-color" placeholder="#fcda81">
        <br>
        <label for="even-color">Even Line Color:</label>
        <input type="text" id="even-color" placeholder="#f58d72">
        <br>
        <label for="text-color">Text Color:</label>
        <input type="text" id="text-color" placeholder="#000000">
        <br>
        <button id="save-colors-button">Save Colors</button>
    </div>
</body>

```

```
<script src="color-admin-script.js"></script>
</body>
</html>
```

Ensure File Structure

Make sure your project directory structure is as follows:

```
java

taplist/
├── package.json
├── package-lock.json
└── server.js
└── public/
    ├── index.html
    ├── admin.html
    ├── login.html
    ├── color-admin.html
    ├── styles.css
    ├── script.js
    ├── admin-script.js
    └── color-admin-script.js
    └── beers.json
    └── colors.json
```

Run the Server and Test

- Start the Server:** Ensure your server is running with `node server.js`.
- Open the Color Admin Page:** Navigate to `http://your-server-ip:3000/color-admin.html` on your mobile device.
- Check the Layout:** Ensure there are line breaks after each text entry field.

This should add a line break after each text entry field, making the layout more readable on mobile devices. Let me know if you need any further adjustments!



Can we host this on github pages so it's both accessible via the internet but also editable via the admin.html page?



GitHub Pages is a great option for hosting static websites, but it doesn't support server-side code execution, which is necessary for handling the admin functionalities like editing the beer list and colors. However, we can use a different approach to achieve a similar outcome:

- Host the Static Site on GitHub Pages:** This will make the site accessible via the internet.
- Use a Backend Service:** Use a backend service like Firebase, AWS Lambda, or another cloud service to handle the admin functionalities and data storage.

Step 1: Host the Static Site on GitHub Pages

1. Create a GitHub Repository:

- Create a new repository on GitHub for your project.

2. Push Your Project to GitHub:

- Initialize a local Git repository if you haven't already:

```
bash
git init
```

- Add your files to the repository:

```
bash
git add .
```

- Commit your changes:

```
bash
git commit -m "Initial commit"
```

- Add the GitHub remote repository:

```
bash
git remote add origin https://github.com/your-username/your-repo-name.git
```

- Push your changes to GitHub:

```
bash
git push -u origin main
```

3. Enable GitHub Pages:

- Go to your repository settings on GitHub.
- Scroll down to the "GitHub Pages" section.
- Select the branch you want to use for GitHub Pages (e.g., `main`).
- Save the changes. Your site will be available at `https://your-username.github.io/your-repo-name`.

Step 2: Use a Backend Service for Admin Functionalities

To manage the admin functionalities, you'll need a backend service. For simplicity, let's use Firebase for this example.

1. Set Up Firebase:

- o Go to the [Firebase Console](#).
- o Create a new project.
- o In the project settings, add a web app to get your Firebase configuration.

2. Install Firebase CLI:

- o Install the Firebase CLI if you haven't already:

```
bash
npm install -g firebase-tools
```

3. Initialize Firebase in Your Project:

- o Log in to Firebase:

```
bash
firebase login
```

- o Initialize Firebase in your project directory:

```
bash
firebase init
```

- o Select the services you want to use (e.g., Firestore, Hosting).

4. Configure Firebase:

- o In your project directory, create a ``firebase-config.js`` file with your Firebase configuration:

```
javascript
// firebase-config.js
import { initializeApp } from 'firebase/app';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_AUTH_DOMAIN",
  projectId: "YOUR_PROJECT_ID",
  storageBucket: "YOUR_STORAGE_BUCKET",
  messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
  appId: "YOUR_APP_ID"
};
```

```
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);

export { db };
```

5. Update Your Admin Scripts:

- Use Firebase Firestore to store and retrieve the beers and colors data.

Example of admin-script.js using Firebase Firestore:

```
javascript

import { db } from './firebase-config';
import { collection, getDocs, updateDoc, doc } from 'firebase/firestore';

document.addEventListener('DOMContentLoaded', () => {
    const tapList = document.getElementById('tap-list');
    const editButton = document.getElementById('edit-button');
    const saveButton = document.getElementById('save-button');
    let beers = [];

    const loadBeers = async () => {
        try {
            const querySnapshot = await getDocs(collection(db, 'beers'));
            beers = querySnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
            displayBeers();
        } catch (error) {
            console.error('Failed to load beers:', error);
        }
    };

    const displayBeers = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <p>
                    <span style="font-size: 29pt;">${index + 1}.&ampnbsp&ampnbsp</span>
                    <span style="font-size: 29pt; color: black; font-weight: bold;">${beer.name}</span>
                    <span style="font-size: 21pt; color: black;">&ampnbspby <b>${beer.brewery}</b>
- ${beer.type} - ${beer.abv}</span>
                </p>
            `;
            tapList.appendChild(beerElement);
        });
    };

    const enableEditing = () => {
        tapList.innerHTML = '';
        beers.forEach((beer, index) => {
            const beerElement = document.createElement('div');
            beerElement.classList.add('beer');
            beerElement.innerHTML = `
                <p>
                    <span style="font-size: 29pt;">${index + 1}.&ampnbsp&ampnbsp</span>
                    <input type="text" value="${beer.name}" class="beer-name" style="font-size: 29pt; color: black; font-weight: bold;">
                    <span style="font-size: 21pt; color: black;">&ampnbspby </span>
                    <input type="text" value="${beer.brewery}" class="brewery" style="font-size: 21pt; color: black;">
                    <input type="text" value="${beer.type}" class="type" style="font-size: 21pt; color: black;">
                    <input type="text" value="${beer.abv}" class="abv" style="font-size: 21pt; color: black;">
                </p>
            `;
        });
    };
});
```

```

    `;
    tapList.appendChild(beerElement);
});

const saveBeers = async () => {
  const beerElements = document.querySelectorAll('.beer');
  beerElements.forEach(async (beerElement, index) => {
    const name = beerElement.querySelector('.beer-name').value;
    const brewery = beerElement.querySelector('.brewery').value;
    const type = beerElement.querySelector('.type').value;
    const abv = beerElement.querySelector('.abv').value;
    const beerRef = doc(db, 'beers', beers[index].id);
    await updateDoc(beerRef, { name, brewery, type, abv });
  });
  alert('Beers updated successfully');
  displayBeers();
};

editButton.addEventListener('click', () => {
  editButton.style.display = 'none';
  saveButton.style.display = 'inline';
  enableEditing();
});

saveButton.addEventListener('click', () => {
  editButton.style.display = 'inline';
  saveButton.style.display = 'none';
  saveBeers();
});

loadBeers();
});

```

Deploy to GitHub Pages

1. Build Your Project:

- o If using a build tool (like Webpack or a framework), build your project.
- o Ensure all necessary files (HTML, CSS, JavaScript) are in the `docs` folder or the root of the repository.

2. Push Changes to GitHub:

- o Push the updated project files to your GitHub repository.

3. Enable GitHub Pages:

- o Ensure GitHub Pages is enabled in the repository settings.
- o Check the URL provided by GitHub Pages to access your site.

Access and Edit from Mobile

1. Access the Site:

- o Visit your GitHub Pages URL from any device.

2. Edit Using Admin Pages:

- o Use the `admin.html` and `color-admin.html` pages to manage your content.

Conclusion

By combining GitHub Pages for hosting the static site and Firebase for backend functionalities, you can create a robust solution that allows you to edit your webpage via `admin.html` from any device on the internet. Let me know if you need further assistance with any part of this process!



No thank you, I think i'll continue to host this locally. Do you think we could make this print its 1918 address to a github pages web page though?



Sure, we can make the local server print its IP address to a GitHub Pages webpage. This will involve:

1. **Creating an Endpoint to Serve the IP Address:** This endpoint will return the server's local IP address.
2. **Fetching the IP Address from GitHub Pages:** A script on the GitHub Pages site will fetch and display the IP address.

Step 1: Create an Endpoint to Serve the IP Address

First, update your `server.js` to include an endpoint that returns the local IP address.

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os'); // Import the os module to get network interfaces

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};
```

```

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing beers file');
        } else {
            res.send('Beers updated successfully');
        }
    });
});

app.post('/update-colors', isAuthenticated, (req, res) => {
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
        if (err) {
            res.status(500).send('Error writing colors file');
        } else {
            res.send('Colors updated successfully');
        }
    });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];

```

```

        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

```

Step 2: Create the GitHub Pages Site

1. Create a GitHub Repository:

- o Create a new repository on GitHub for your project.

2. Add the HTML and JavaScript to Fetch the IP Address:

- o Create an `index.html` file that will fetch the IP address from the local server.

index.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Local Server IP</title>
    <script>
        async function fetchLocalIp() {
            try {
                const response = await fetch('http://localhost:3000/ip');
                const data = await response.json();
                document.getElementById('ip-address').innerText = `Local Server IP:
${data.ip}`;
            } catch (error) {
                console.error('Error fetching IP address:', error);
                document.getElementById('ip-address').innerText = 'Unable to fetch IP
address';
            }
        }

        window.onload = fetchLocalIp;
    </script>
</head>
<body>
    <h1>Local Server IP</h1>
    <p id="ip-address">Fetching IP address...</p>
</body>
</html>

```

3. Push the Code to GitHub:

- o Initialize a local Git repository if you haven't already:

```
bash
```

```
git init
```

- Add your files to the repository:

```
bash
```

```
git add .
```

- Commit your changes:

```
bash
```

```
git commit -m "Initial commit"
```

- Add the GitHub remote repository:

```
bash
```

```
git remote add origin https://github.com/your-username/your-repo-name.git
```

- Push your changes to GitHub:

```
bash
```

```
git push -u origin main
```

4. Enable GitHub Pages:

- Go to your repository settings on GitHub.
- Scroll down to the "GitHub Pages" section.
- Select the branch you want to use for GitHub Pages (e.g., `main`).
- Save the changes. Your site will be available at `https://your-username.github.io/your-repo-name`.

Accessing the GitHub Pages Site

1. Visit the GitHub Pages URL:

- Go to `https://your-username.github.io/your-repo-name`.

2. View the Local Server IP:

- The page will display the local server IP address.

Notes

- The GitHub Pages site will attempt to fetch the IP address from `http://localhost:3000/ip`, which only works when accessed from the same device hosting the server.
- For accessing the IP from different devices, you might need to use an intermediary service or a more sophisticated setup.

This setup provides a way to fetch and display the local IP address of the server, making it accessible via GitHub Pages. Let me know if you need further assistance!



You said "git add ." to add files to the newly made repo, is that right?



Yes, `git add .` is the correct command to add all the files in your current directory to the staging area in Git. Here's a step-by-step guide to push your project to GitHub and host it on GitHub Pages:

Step-by-Step Guide

1. Initialize the Git Repository (if not already initialized):

```
bash
git init
```

2. Add All Files to the Repository:

```
bash
git add .
```

This command stages all the changes in the current directory.

3. Commit the Changes:

```
bash
git commit -m "Initial commit"
```

4. Add the Remote Repository:

bash

```
git remote add origin https://github.com/your-username/your-repo-name.git
```

5. Push the Changes to GitHub:

bash

```
git push -u origin main
```

Ensure GitHub Pages is Enabled

1. Go to Your Repository Settings on GitHub:

- o Navigate to your repository on GitHub.
- o Click on the "Settings" tab.

2. Scroll Down to the GitHub Pages Section:

- o In the "GitHub Pages" section, select the branch you want to use for GitHub Pages (typically `main`).
- o Click "Save".

Example Project Structure

Ensure your project structure looks something like this:

perl

```
my-project/
└── index.html
└── README.md
```

Example `index.html`

Here is an example `index.html` that fetches the local server IP address:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Local Server IP</title>
    <script>
        async function fetchLocalIp() {
            try {
                const response = await fetch('http://localhost:3000/ip');
                const data = await response.json();
```

```

        document.getElementById('ip-address').innerText = `Local Server IP:
${data.ip}`;
    } catch (error) {
        console.error('Error fetching IP address:', error);
        document.getElementById('ip-address').innerText = 'Unable to fetch IP
address';
    }
}

window.onload = fetchLocalIp;
</script>
</head>
<body>
    <h1>Local Server IP</h1>
    <p id="ip-address">Fetching IP address...</p>
</body>
</html>

```

Testing the Setup

1. Run Your Local Server:

Ensure your local server (with the `server.js` changes to serve the IP address) is running:

```

bash
node server.js

```

2. Visit Your GitHub Pages URL:

Navigate to `https://your-username.github.io/your-repo-name` in your web browser.

3. Check the Output:

The page should attempt to fetch the local server's IP address. However, note that this will work when accessed from the same device hosting the server.

Conclusion

This setup allows you to host your static site on GitHub Pages and fetch the local server's IP address when accessed locally. If you need the site to be editable and accessible from other devices, you might need a more complex setup involving a backend service.

Let me know if you need further assistance!



I got this message in the terminal "cambro2:github-pages-website duncananderson\$ git push -u origin main
error: src refspec main does not match any.
error: failed to push some refs to 'https://github.com/taplists/taplists.github.io.git'"



The error message indicates that the branch `main` does not exist in your local repository. This is a common issue if your local repository is using a different branch name, such as `master`.

Here's how you can resolve this issue:

Step 1: Check Your Current Branch

First, check which branch you are currently on:

```
bash  
  
git branch
```

If you see `* master`, you are on the `master` branch.

Step 2: Create and Switch to `main` Branch (if necessary)

If you want to use the `main` branch instead of `master`, you can create and switch to the `main` branch:

```
bash  
  
git checkout -b main
```

Step 3: Add Files, Commit, and Push

Add your files, commit the changes, and push to the `main` branch:

```
bash  
  
git add .  
git commit -m "Initial commit"  
git push -u origin main
```

Step 4: Update GitHub Pages Settings (if necessary)

If you have configured GitHub Pages to use the `main` branch, make sure the branch is correctly set in your repository settings on GitHub.

Example Commands

Here are the commands all together:

bash

```
# Check the current branch
git branch

# If needed, create and switch to the main branch
git checkout -b main

# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"

# Push the changes to the main branch on GitHub
git push -u origin main
```

Alternative: Push to `master` Branch

If you prefer to use the `master` branch, you can push to `master` instead:

bash

```
# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"

# Push the changes to the master branch on GitHub
git push -u origin master
```

Then, make sure to configure GitHub Pages to use the `master` branch in your repository settings.

Summary

Ensure you are on the correct branch and push to that branch. Update your GitHub Pages settings to reflect the branch you are using.

Let me know if you encounter any further issues or need additional assistance!



So how will the website push the 1918 address to github?



To automatically push the local IP address (192.168.x.x) to a GitHub Pages site, you'll need a way to update a file in your GitHub repository with the current IP address. This can be done using a GitHub Action, which is a workflow automation tool provided by GitHub.

Here's an overview of how to accomplish this:

1. Set up your local server to expose the IP address.
2. Create a GitHub repository and host a static site on GitHub Pages.
3. Set up a GitHub Action to update the IP address on GitHub Pages.
4. Use a cron job or similar scheduling tool to periodically update the IP address.

Step-by-Step Guide

Step 1: Set Up Your Local Server

Make sure your server has an endpoint that exposes its IP address. You've already set up this part in your `server.js`:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os'); // Import the os module to get network interfaces

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});
```

```

});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/update-beers', isAuthenticated, (req, res) => {
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing beers file');
    } else {
      res.send('Beers updated successfully');
    }
  });
});

app.post('/update-colors', isAuthenticated, (req, res) => {
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(req.body, null, 2), err => {
    if (err) {
      res.status(500).send('Error writing colors file');
    } else {
      res.send('Colors updated successfully');
    }
  });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

```

Step 2: Create a GitHub Repository

Create a new repository on GitHub. This repository will host the static site on GitHub Pages.

Step 3: Create the GitHub Pages Site

Create an `index.html` file that will display the IP address.

index.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Local Server IP</title>
</head>
<body>
    <h1>Local Server IP</h1>
    <p id="ip-address">Fetching IP address...</p>

    <script>
        async function fetchLocalIp() {
            try {
                const response = await fetch('ip.json');
                const data = await response.json();
                document.getElementById('ip-address').innerText = `Local Server IP: ${data.ip}`;
            } catch (error) {
                console.error('Error fetching IP address:', error);
                document.getElementById('ip-address').innerText = 'Unable to fetch IP address';
            }
        }

        window.onload = fetchLocalIp;
    </script>
</body>
</html>
```

Step 4: Add the IP Fetching Script to Your Local Server

Create a script that fetches the IP address from your local server and commits it to the GitHub repository.

update-ip.js:

```
javascript

const fetch = require('node-fetch');
const fs = require('fs');
const { exec } = require('child_process');

const GITHUB_REPO = 'https://github.com/your-username/your-repo-name.git';
const GITHUB_BRANCH = 'main';
const IP_FILE_PATH = 'ip.json';

async function fetchLocalIp() {
    const response = await fetch('http://localhost:3000/ip');
    const data = await response.json();
    return data.ip;
}

async function updateIpFile(ip) {
```

```

const ipData = { ip };
fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));
}

function commitAndPushChanges() {
  exec(`git add ${IP_FILE_PATH}`, (err, stdout, stderr) => {
    if (err) {
      console.error('Error adding file:', stderr);
      return;
    }
    exec(`git commit -m "Update IP address"`, (err, stdout, stderr) => {
      if (err) {
        console.error('Error committing changes:', stderr);
        return;
      }
      exec(`git push ${GITHUB_REPO} ${GITHUB_BRANCH}`, (err, stdout, stderr) => {
        if (err) {
          console.error('Error pushing changes:', stderr);
        } else {
          console.log('IP address updated successfully.');
        }
      });
    });
  });
}

async function main() {
  const ip = await fetchLocalIp();
  await updateIpFile(ip);
  commitAndPushChanges();
}

main().catch(console.error);

```

Step 5: Schedule the Script to Run Periodically

You can use a cron job (Linux/macOS) or Task Scheduler (Windows) to run the script periodically.

Cron Job Example (run every hour):

```

bash

0 * * * * /usr/bin/node /path/to/update-ip.js

```

Conclusion

By following these steps, you can automatically update the local IP address of your server to a GitHub Pages site. The GitHub Pages site will fetch and display the IP address, ensuring that it is always up-to-date.

Let me know if you need further assistance!



Can we put it in server.js to call update-ip.js periodically



Yes, you can use Node.js to periodically run a function within your `server.js` to update the IP address. This avoids the need for a separate cron job and keeps everything within your Node.js application.

Here's how you can integrate the periodic update within `server.js`:

1. Install Necessary Packages:

You need the `node-fetch` package to fetch the IP address and the `simple-git` package to handle Git operations.

```
bash
npm install node-fetch simple-git
```

2. Update `server.js`:

Add the necessary code to periodically fetch the IP address and update the GitHub Pages site.

server.js:

```
javascript
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const fetch = require('node-fetch');
const simpleGit = require('simple-git');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/your-username/your-repo-name.git'; // Replace with your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.get('/admin.html', isAuthenticated);
app.get('/color-admin.html', isAuthenticated);
```

```

app.use((req, res, next) => {
  if (req.path === '/admin.html' || req.path === '/color-admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  startPeriodicIpUpdate();
});

const git = simpleGit();

// Function to fetch the local IP and update the IP file
async function fetchAndUpdateIp() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';

```

```

for (const interfaceName in networkInterfaces) {
  const interfaces = networkInterfaces[interfaceName];
  for (const i of interfaces) {
    if (i.family === 'IPv4' && !i.internal) {
      localIp = i.address;
      break;
    }
  }
}

const ipData = { ip: localIp };
fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

try {
  await git.add(IP_FILE_PATH);
  await git.commit('Update IP address');
  await git.push(GITHUB_REPO, GITHUB_BRANCH);
  console.log('IP address updated successfully.');
} catch (error) {
  console.error('Error updating IP address:', error);
}
}

// Function to start periodic IP updates
function startPeriodicIpUpdate() {
  fetchAndUpdateIp(); // Initial call to update immediately
  setInterval(fetchAndUpdateIp, 3600000); // Update every hour
}

```

Explanation

- 1. Install Dependencies:** Ensure `node-fetch` and `simple-git` are installed.
- 2. Update Server.js:** Add the `fetchAndUpdateIp` function to fetch the local IP and commit it to your GitHub repository.
- 3. Periodic Updates:** Use `setInterval` to run the `fetchAndUpdateIp` function every hour.

Run Your Server

- 1. Start the Server:**

```

bash

node server.js

```

- 2. Access Your GitHub Pages Site:**

- o Visit your GitHub Pages URL to see the updated IP address.

This setup will periodically update the IP address in your GitHub Pages site. Let me know if you need further assistance!



How do I unlink vscode from my github account



To unlink Visual Studio Code (VS Code) from your GitHub account, you need to remove any stored credentials and disconnect any extensions or integrations that use your GitHub account. Here's a step-by-step guide:

Step 1: Remove GitHub Authentication in VS Code

1. Open the Command Palette:

- Press `Ctrl+Shift+P` (Windows/Linux) or `Cmd+Shift+P` (Mac) to open the Command Palette.

2. Search for and Select `Settings Sync: Sign Out`:

- Type `Settings Sync: Sign Out` and select it to sign out of the Settings Sync feature if it's using your GitHub account.

3. Check for GitHub-related Extensions:

- Go to the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of the window or by pressing `Ctrl+Shift+X`.
- Look for any GitHub-related extensions (e.g., GitHub Pull Requests and Issues, GitLens, etc.) and disable or uninstall them if they are linked to your GitHub account.

Step 2: Remove Git Credentials from Git Credential Manager

1. Open Credential Manager:

- On Windows:
 - Open the Start menu, type `Credential Manager`, and press Enter.
 - Go to `Windows Credentials` and look for any entries related to Git or GitHub.
 - Remove those entries.
- On macOS:
 - Open the `Keychain Access` application.
 - Search for any items related to `GitHub` and delete them.
- On Linux:
 - If you use a credential manager like `gnome-keyring`, you can use the appropriate command to remove credentials.
 - Alternatively, check `~/.git-credentials` or `~/.config/git/credentials` and remove any GitHub entries.

Step 3: Remove Saved SSH Keys (if applicable)

If you use SSH keys to authenticate with GitHub, you might want to remove or unlink the keys:

1. Remove SSH Keys:

- Open the Terminal.
- Navigate to the `~/.ssh` directory:

```
bash
```

```
cd ~/.ssh
```

- o List the files to see your SSH keys:

```
bash
```

```
ls
```

- o Remove the SSH keys related to GitHub if you no longer need them:

```
bash
```

```
rm id_rsa_github
rm id_rsa_github.pub
```

2. Remove SSH Key Configuration:

- o Open the `~/.ssh/config` file and remove or comment out any configurations related to GitHub.

Step 4: Revoke GitHub Access Tokens

1. Go to GitHub Settings:

- o Log in to your GitHub account.
- o Go to `Settings` > `Applications` > `Authorized OAuth Apps` .
- o Find and revoke any tokens associated with VS Code or other applications you no longer want to access your GitHub account.

2. Regenerate or Remove Personal Access Tokens:

- o In `Settings` > `Developer settings` > `Personal access tokens` , regenerate or delete any tokens that are used by VS Code or other tools.

Step 5: Remove Git Configuration

1. Check Git Configuration:

- o Open the Terminal or Command Prompt.
- o Check your global Git configuration:

```
bash
```

```
git config --global --list
```

2. Remove GitHub-specific Configuration:

- Remove any configurations related to GitHub:

```
bash
git config --global --unset user.name
git config --global --unset user.email
git config --global --unset credential.helper
```

By following these steps, you should successfully unlink VS Code from your GitHub account and remove any stored credentials. Let me know if you need further assistance!



in update-ip.js, my username is taplists, the repo name is taplists.github.io and the branch is called main



Great! Let's update the `update-ip.js` script with your specific details:

1. Install Necessary Packages:

If you haven't already, make sure you have installed `node-fetch` and `simple-git`.

```
bash
npm install node-fetch simple-git
```

2. Update `update-ip.js` Script:

Here is the updated script with your specific details:

update-ip.js:

```
javascript
const fetch = require('node-fetch');
const fs = require('fs');
const simpleGit = require('simple-git');

const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git';
const GITHUB_BRANCH = 'main';
const IP_FILE_PATH = 'ip.json';

const git = simpleGit();

async function fetchLocalIp() {
    try {
        const response = await fetch('http://localhost:3000/ip');
        const data = await response.json();
        return data.ip;
    } catch (error) {
        console.error('Error fetching local IP:', error);
    }
}

git.fetch({ ref: GITHUB_BRANCH })
    .then(() => {
        return git.checkout(GITHUB_BRANCH);
    })
    .then(() => {
        return git.pull();
    })
    .then(() => {
        return git.status();
    })
    .then(status => {
        if (status.added || status.modified) {
            console.log(`Detected changes in repository. Updating IP file...`);
            const ipData = { ip: fetchLocalIp().then(ip => ip) };
            fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData));
            console.log(`IP file updated with local IP: ${ipData.ip}`);
        } else {
            console.log(`Repository is up-to-date. No changes detected.`);
        }
    })
    .catch(error => {
        console.error(`Error updating repository: ${error}`);
    });
}
```

```

    } catch (error) {
      console.error('Error fetching local IP address:', error);
      throw error;
    }
  }

async function updateIpFile(ip) {
  const ipData = { ip };
  fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));
}

async function commitAndPushChanges() {
  try {
    await git.add(IP_FILE_PATH);
    await git.commit('Update IP address');
    await git.push('origin', GITHUB_BRANCH);
    console.log('IP address updated successfully.');
  } catch (error) {
    console.error('Error pushing changes to GitHub:', error);
    throw error;
  }
}

async function main() {
  try {
    const ip = await fetchLocalIp();
    await updateIpFile(ip);
    await commitAndPushChanges();
  } catch (error) {
    console.error('Failed to update IP address:', error);
  }
}

main();

```

3. Ensure Your Local Git Repository is Properly Set Up:

Make sure your local git repository is configured correctly and has the correct remote repository URL. You can check and set this using the following commands:

bash

```
git remote set-url origin https://github.com/taplists/taplists.github.io.git
```

4. Test the Script:

Run the `update-ip.js` script manually to ensure it works:

bash

```
node update-ip.js
```

5. Integrate with Your Server:

You can integrate this script to run periodically within your `server.js`.

server.js:

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const fetch = require('node-fetch');
const simpleGit = require('simple-git');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git'; // Replace with your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

```

```

        });
    });

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    startPeriodicIpUpdate();
});

const git = simpleGit();

// Function to fetch the local IP and update the IP file
async function fetchAndUpdateIp() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const ipData = { ip: localIp };
    fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

    try {
        await git.add(IP_FILE_PATH);
        await git.commit('Update IP address');
        await git.push('origin', GITHUB_BRANCH);
        console.log('IP address updated successfully.');
    } catch (error) {
        console.error('Error updating IP address:', error);
    }
}

// Function to start periodic IP updates
function startPeriodicIpUpdate() {
    fetchAndUpdateIp(); // Initial call to update immediately
    setInterval(fetchAndUpdateIp, 3600000); // Update every hour
}

```

This script will fetch the local IP address, update the `ip.json` file, and commit and push the changes to your GitHub repository every hour.

By integrating these steps into your project, you ensure that the local IP address is periodically updated and pushed to your GitHub Pages site. Let me know if you need any further assistance!



I got this terminal message when I checked to see if the local git repo was properly set up

```
cambro2:Taplist-project duncananderson$ git remote set-url origin  
https://github.com/taplists/taplists.github.io.git  
fatal: No such remote 'origin'
```



The error message indicates that your local Git repository doesn't have a remote named `origin`. Let's fix this by adding the remote repository and ensuring everything is set up correctly.

Step-by-Step Guide to Add the Remote Repository

1. Initialize Git (if not already initialized):

If your repository is not already initialized, you need to initialize it:

```
bash
```

```
git init
```

2. Add the Remote Repository:

Add the remote repository URL to your local repository:

```
bash
```

```
git remote add origin https://github.com/taplists/taplists.github.io.git
```

3. Verify the Remote Repository:

Verify that the remote repository has been added correctly:

```
bash
```

```
git remote -v
```

You should see output similar to:

bash

```
origin https://github.com/taplists/taplists.github.io.git (fetch)
origin https://github.com/taplists/taplists.github.io.git (push)
```

4. Commit and Push Changes:

Ensure you have committed any changes before pushing:

bash

```
git add .
git commit -m "Initial commit"
git push -u origin main
```

Full Example of Commands

Here's the complete set of commands to initialize the repository, add the remote, and push the changes:

bash

```
# Initialize Git (if not already done)
git init

# Add the remote repository
git remote add origin https://github.com/taplists/taplists.github.io.git

# Verify the remote repository
git remote -v

# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"

# Push the changes to the main branch
git push -u origin main
```

Updating the `server.js` Script

Ensure your `server.js` script is set up to periodically update the IP address:

server.js:

javascript

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const fetch = require('node-fetch');
const simpleGit = require('simple-git');
```

```

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git'; // Replace with
your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 'your_secret_key', // Replace with a secure key
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
  if (req.path === '/admin.html' || req.path === '/color-admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

```

```

});;

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  startPeriodicIpUpdate();
});

const git = simpleGit();

// Function to fetch the local IP and update the IP file
async function fetchAndUpdateIp() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const ipData = { ip: localIp };
  fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

  try {
    await git.add(IP_FILE_PATH);
    await git.commit('Update IP address');
    await git.push('origin', GITHUB_BRANCH);
    console.log('IP address updated successfully.');
  } catch (error) {
    console.error('Error updating IP address:', error);
  }
}

// Function to start periodic IP updates
function startPeriodicIpUpdate() {
  fetchAndUpdateIp(); // Initial call to update immediately
  setInterval(fetchAndUpdateIp, 3600000); // Update every hour
}

```

By following these steps, you should be able to set up your local repository correctly, push changes to GitHub, and periodically update the IP address in your GitHub Pages site. Let me know if you encounter any further issues or need additional assistance!



received this terminal message when pushing the changes to the main branch

cambro2:Taplist-project duncananderson\$ git push -u origin main

```
error: src refspec main does not match any.  
error: failed to push some refs to 'https://github.com/taplists/taplists.github.io.git'
```



The error message indicates that there is no `main` branch in your local repository. By default, older Git repositories use `master` as the default branch. Here's how to resolve this issue:

Step 1: Check Current Branch

First, check which branch you are currently on:

```
bash  
  
git branch
```

If you see `* master`, you are on the `master` branch. In this case, you need to either push to the `master` branch or rename it to `main`.

Step 2: Rename `master` to `main` (if necessary)

If you want to use `main` instead of `master`, you can rename the branch:

```
bash  
  
git branch -m master main
```

Step 3: Push to GitHub

After renaming the branch (if necessary), add the remote repository and push the changes:

```
bash  
  
# Add the remote repository (if not already added)  
git remote add origin https://github.com/taplists/taplists.github.io.git  
  
# Push the changes to the main branch  
git push -u origin main
```

Complete Set of Commands

Here are the commands all together, including initialization, renaming, and pushing:

bash

```
# Initialize Git (if not already done)
git init

# Check the current branch
git branch

# If on 'master', rename it to 'main'
git branch -m master main

# Add the remote repository
git remote add origin https://github.com/taplists/taplists.github.io.git

# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"

# Push the changes to the main branch
git push -u origin main
```

Alternative: Push to `master` Branch

If you prefer to keep using the `master` branch, push to `master` instead of `main`:

bash

```
# Initialize Git (if not already done)
git init

# Add the remote repository
git remote add origin https://github.com/taplists/taplists.github.io.git

# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"

# Push the changes to the master branch
git push -u origin master
```

Update GitHub Pages Settings

Ensure that GitHub Pages is set to use the correct branch (`main` or `master`) in the repository settings:

1. Go to Your Repository Settings on GitHub:

- o Navigate to your repository on GitHub.
- o Click on the "Settings" tab.

2. Scroll Down to the GitHub Pages Section:

- o In the "GitHub Pages" section, select the branch you want to use (e.g., `main` or `master`).
- o Click "Save".

By following these steps, you should be able to resolve the issue and push your changes to GitHub. Let me know if you encounter any further issues or need additional assistance!



```
cambro2:Taplist-project duncananderson$ git push -u origin main
To https://github.com/taplists/taplists.github.io.git
! [rejected]      main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/taplists/taplists.github.io.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
cambro2:Taplist-project duncananderson$
```



The error message indicates that the remote branch has changes that your local branch does not have, so Git is preventing you from pushing your changes to avoid overwriting those remote changes.

Step-by-Step Guide to Resolve the Issue

1. Pull Remote Changes:

First, pull the remote changes to your local branch and resolve any merge conflicts if they occur.

```
bash
git pull origin main --rebase
```

2. Resolve Any Merge Conflicts:

If there are merge conflicts, Git will inform you which files are in conflict. Open those files, resolve the conflicts, and then add the resolved files:

```
bash
git add <file_with_conflict>
```

3. Continue the Rebase:

After resolving conflicts, continue the rebase:

```
bash
git rebase --continue
```

4. Push Changes:

After successfully rebasing, push your changes to the remote repository:

```
bash
git push -u origin main
```

Complete Example of Commands

Here are the commands all together:

```
bash
# Pull remote changes with rebase
git pull origin main --rebase

# Resolve any merge conflicts, if they occur
# Open the conflicting files, resolve conflicts, and then add them
git add <file_with_conflict>

# Continue the rebase after resolving conflicts
git rebase --continue

# Push changes to the remote repository
git push -u origin main
```

Explanation of Commands

- `git pull origin main --rebase`: Pulls the remote changes and rebases your local changes on top of them. This helps to keep a cleaner commit history.
- `git add <file_with_conflict>`: Adds the resolved files to the staging area after fixing merge conflicts.
- `git rebase --continue`: Continues the rebase process after conflicts are resolved.
- `git push -u origin main`: Pushes your changes to the remote repository.

By following these steps, you should be able to successfully pull the remote changes, resolve any conflicts, and push your local changes to the `main` branch on GitHub.

Let me know if you encounter any further issues or need additional assistance!



How can I change the directory that github is uploading from? I made it the wrong directory on accident



To change the directory that GitHub is uploading from, you need to update the remote repository URL or reinitialize the Git repository in the correct directory. Here's how you can do it:

Option 1: Change the Directory in Git Configuration

If you already have a Git repository initialized in the correct directory and just need to change the remote URL, you can update the remote URL using the following commands:

1. Navigate to the Correct Directory:

```
bash
cd /path/to/correct/directory
```

2. Initialize a New Git Repository (if not already initialized):

```
bash
git init
```

3. Set the Correct Remote URL:

```
bash
git remote set-url origin https://github.com/your-username/your-repo-name.git
```

4. Verify the Remote URL:

```
bash
git remote -v
```

5. Add, Commit, and Push Your Changes:

```
bash
git add .
git commit -m "Initial commit"
git push -u origin main
```

Option 2: Reinitialize the Git Repository in the Correct Directory

If you need to reinitialize the Git repository in the correct directory, follow these steps:

1. Navigate to the Correct Directory:

```
bash
```

```
cd /path/to/correct/directory
```

2. Initialize a New Git Repository:

```
bash
```

```
git init
```

3. Add the Remote Repository:

```
bash
```

```
git remote add origin https://github.com/your-username/your-repo-name.git
```

4. Verify the Remote URL:

```
bash
```

```
git remote -v
```

5. Add, Commit, and Push Your Changes:

```
bash
```

```
git add .
git commit -m "Initial commit"
git push -u origin main
```

Example Commands

Here's an example set of commands to reinitialize the Git repository in the correct directory:

```
bash
```

```
# Navigate to the correct directory
cd /path/to/correct/directory

# Initialize a new Git repository (if not already done)
git init

# Add the remote repository
git remote add origin https://github.com/taplists/taplists.github.io.git

# Verify the remote URL
```

```
git remote -v
# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"

# Push the changes to the main branch
git push -u origin main
```

By following these steps, you should be able to set up your Git repository to upload from the correct directory.

Let me know if you encounter any further issues or need additional assistance!



```
cambro2:github-taplist duncananderson$ git push -u origin main
error: src refspec main does not match any.
error: failed to push some refs to 'https://github.com/taplists/taplists.github.io.git'
cambro2:github-taplist duncananderson$
```



The error message indicates that there is no `main` branch in your local repository. By default, older Git repositories use `master` as the default branch. Let's resolve this issue by either creating a `main` branch or pushing to the `master` branch.

Step 1: Check Current Branch

First, check which branch you are currently on:

```
bash
git branch
```

If you see `* master`, you are on the `master` branch. In this case, you need to either push to the `master` branch or rename it to `main`.

Step 2: Rename `master` to `main` (if necessary)

If you want to use `main` instead of `master`, you can rename the branch:

```
bash
git branch -m master main
```

Step 3: Push to GitHub

After renaming the branch (if necessary), add the remote repository and push the changes:

```
bash

# Add the remote repository (if not already added)
git remote add origin https://github.com/taplists/taplists.github.io.git

# Push the changes to the main branch
git push -u origin main
```

Complete Set of Commands

Here are the commands all together, including initialization, renaming, and pushing:

```
bash

# Initialize Git (if not already done)
git init

# Check the current branch
git branch

# If on 'master', rename it to 'main'
git branch -m master main

# Add the remote repository
git remote add origin https://github.com/taplists/taplists.github.io.git

# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"

# Push the changes to the main branch
git push -u origin main
```

Alternative: Push to `master` Branch

If you prefer to keep using the `master` branch, push to `master` instead of `main`:

```
bash

# Initialize Git (if not already done)
git init

# Add the remote repository
git remote add origin https://github.com/taplists/taplists.github.io.git

# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"
```

```
# Push the changes to the master branch
git push -u origin master
```

Update GitHub Pages Settings

Ensure that GitHub Pages is set to use the correct branch (``main`` or ``master``) in the repository settings:

- 1. Go to Your Repository Settings on GitHub:**

- Navigate to your repository on GitHub.
- Click on the "Settings" tab.

- 2. Scroll Down to the GitHub Pages Section:**

- In the "GitHub Pages" section, select the branch you want to use (e.g., `main` or `master`).
- Click "Save".

By following these steps, you should be able to resolve the issue and push your changes to GitHub. Let me know if you encounter any further issues or need additional assistance!



```
cambro2:github-taplist duncananderson$ git push -u origin main
To https://github.com/taplists/taplists.github.io.git
 ! [rejected]      main -> main (fetch first)
error: failed to push some refs to 'https://github.com/taplists/taplists.github.io.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
cambro2:github-taplist duncananderson$
```



The error indicates that the remote repository has changes that your local repository doesn't have. You need to integrate those remote changes into your local repository before you can push your local changes. Here's how to do that:

Step-by-Step Guide

- 1. Pull Remote Changes:**

First, pull the remote changes to your local branch. This may cause merge conflicts, which you'll need to resolve.

```
bash
```

```
git pull origin main --rebase
```

2. Resolve Any Merge Conflicts:

If there are merge conflicts, Git will inform you which files are in conflict. Open those files, resolve the conflicts, and then add the resolved files:

```
bash
```

```
# Resolve conflicts in the conflicting files
git add <file_with_conflict>
```

3. Continue the Rebase:

After resolving conflicts, continue the rebase:

```
bash
```

```
git rebase --continue
```

4. Push Changes:

After successfully rebasing, push your changes to the remote repository:

```
bash
```

```
git push -u origin main
```

Complete Example of Commands

Here are the commands all together:

```
bash
```

```
# Pull remote changes with rebase
git pull origin main --rebase

# Resolve any merge conflicts, if they occur
# Open the conflicting files, resolve conflicts, and then add them
git add <file_with_conflict>

# Continue the rebase after resolving conflicts
git rebase --continue

# Push changes to the remote repository
git push -u origin main
```

Explanation of Commands

- `git pull origin main --rebase`: Pulls the remote changes and rebases your local changes on top of them. This helps to keep a cleaner commit history.
- `git add <file_with_conflict>`: Adds the resolved files to the staging area after fixing merge conflicts.
- `git rebase --continue`: Continues the rebase process after conflicts are resolved.
- `git push -u origin main`: Pushes your changes to the remote repository.

By following these steps, you should be able to successfully pull the remote changes, resolve any conflicts, and push your local changes to the `main` branch on GitHub.

Let me know if you encounter any further issues or need additional assistance!



I get this message when I attempt to run server.js

```
/usr/local/bin/node ./server.js
Process exited with code 1
Uncaught Error Error [ERR_REQUIRE_ESM]: require() of ES Module
/Users/duncananderson/Desktop/Taplist-Project/node_modules/node-fetch/src/index.js from
/Users/duncananderson/Desktop/Taplist-Project/server.js not supported.
Instead change the require of index.js in /Users/duncananderson/Desktop/Taplist-Project/server.js
to a dynamic import() which is available in all CommonJS modules.
    at Module._extensions..js (internal/modules/cjs/loader:1389:19)
    at Module.load (internal/modules/cjs/loader:1208:32)
    at Module._load (internal/modules/cjs/loader:1024:12)
    at Module.require (internal/modules/cjs/loader:1233:19)
    at require (internal/modules/helpers:179:18)
    at <anonymous> (/Users/duncananderson/Desktop/Taplist-Project/server.js:7:15)
    at Module._compile (internal/modules/cjs/loader:1358:14)
    at Module._extensions..js (internal/modules/cjs/loader:1416:10)
    at Module.load (internal/modules/cjs/loader:1208:32)
    at Module._load (internal/modules/cjs/loader:1024:12)
    at executeUserEntryPoint (internal/modules/run_main:174:12)
    at <anonymous> (internal/main/run_main_module:28:49)
server.js:7
```



The error message indicates that `node-fetch` is being imported using `require()`, but `node-fetch` is now an ES module, which means it must be imported using the `import` statement.

To fix this, you can either convert your `server.js` to an ES module by renaming it to `mjs` and using `import` statements, or you can use dynamic imports with `import()`.

Option 1: Convert to ES Module

1. Rename `server.js` to `server.mjs`:

Rename your `server.js` file to `server.mjs` to indicate it's an ES module.

2. Update Import Statements:

Use the `import` statement instead of `require`.

server.mjs:

```
javascript

import express from 'express';
import bodyParser from 'body-parser';
import path from 'path';
import { fileURLToPath } from 'url';
import fs from 'fs';
import session from 'express-session';
import os from 'os';
import fetch from 'node-fetch';
import simpleGit from 'simple-git';

// Fix __dirname and __filename in ES modules
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git'; // Replace with your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});
```

```

    });
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  startPeriodicIpUpdate();
});

const git = simpleGit();

// Function to fetch the local IP and update the IP file
async function fetchAndUpdateIp() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const ipData = { ip: localIp };
  fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

  try {
    await git.add(IP_FILE_PATH);
    await git.commit('Update IP address');
    await git.push('origin', GITHUB_BRANCH);
  }
}

```

```

        console.log('IP address updated successfully.');
    } catch (error) {
        console.error('Error updating IP address:', error);
    }
}

// Function to start periodic IP updates
function startPeriodicIpUpdate() {
    fetchAndUpdateIp(); // Initial call to update immediately
    setInterval(fetchAndUpdateIp, 3600000); // Update every hour
}

```

Option 2: Use Dynamic Imports

If you prefer to keep the file as a CommonJS module, you can use dynamic imports for `node-fetch`.

server.js:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const simpleGit = require('simple-git');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git'; // Replace with
your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;

```

```

if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
} else {
    res.send('Incorrect password');
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    startPeriodicIpUpdate();
});

const git = simpleGit();

// Function to fetch the local IP and update the IP file
async function fetchAndUpdateIp() {
    const fetch = await import('node-fetch').then(module => module.default);
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    const ipData = { ip: localIp };
}

```

```

fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

try {
    await git.add(IP_FILE_PATH);
    await git.commit('Update IP address');
    await git.push('origin', GITHUB_BRANCH);
    console.log('IP address updated successfully.');
} catch (error) {
    console.error('Error updating IP address:', error);
}

// Function to start periodic IP updates
function startPeriodicIpUpdate() {
    fetchAndUpdateIp(); // Initial call to update immediately
    setInterval(fetchAndUpdateIp, 3600000); // Update every hour
}

```

Run Your Server

1. Start the Server:

```

bash

node server.mjs

```

Or, if using the dynamic import method:

```

bash

node server.js

```

By following these steps, you should be able to resolve the issue and run your server correctly. Let me know if you encounter any further issues or need additional assistance!



This was the message when running server.js

Server is running on http://localhost:300



It looks like there might be a typo in your server configuration, as the port number `300` is



0



It looks like there was a typo in my previous message regarding the port number. The correct port number should be `3000`. Here's how you can fix it:

Update Your `server.js`

Ensure that your `server.js` has the correct port number `3000`:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const simpleGit = require('simple-git');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH =
```



This was the message when running server.js

Server is running on http://localhost:3000

Error updating IP address: Error: To https://github.com/taplists/taplists.github.io.git

! refs/heads/main:refs/heads/main [rejected] (non-fast-forward)

Done

Pushing to https://github.com/taplists/taplists.github.io.git

error: failed to push some refs to 'https://github.com/taplists/taplists.github.io.git'

hint: Updates were rejected because the tip of your current branch is behind

hint: its remote counterpart. Integrate the remote changes (e.g.

hint: 'git pull ...') before pushing again.

hint: See the 'Note about fast-forwards' in 'git push --help' for details.

at Object.action (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1445:25)

at PluginStore.exec (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-

```
git/dist/cjs/index.js:1494:29)
  at /Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-
git/dist/cjs/index.js:1921:43
  at new Promise (<anonymous>)
  at GitExecutorChain.handleTaskData (/Users/duncananderson/Desktop/Taplist-
Project/node_modules/simple-git/dist/cjs/index.js:1919:16)
  at GitExecutorChain.<anonymous> (/Users/duncananderson/Desktop/Taplist-
Project/node_modules/simple-git/dist/cjs/index.js:1903:44)
  at Generator.next (<anonymous>)
  at fulfilled (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-
git/dist/cjs/index.js:52:24) {task: {...}, stack: 'Error: To https://github.com/taplists/taplist...
e_modules/simple-git/dist/cjs/index.js:52:24)', message: "To
https://github.com/taplists/taplists.gith...forwards' in 'git push --help' for details.
"}
```



The error message indicates that the remote branch has changes that your local branch does not have, so Git is preventing you from pushing your changes to avoid overwriting those remote changes. To resolve this, you need to pull the remote changes and merge or rebase them with your local changes.

Here's how you can update your `server.js` to handle this situation by including a pull operation before pushing:

Step-by-Step Guide

1. Pull Remote Changes:

Ensure that you pull the remote changes before attempting to push.

2. Resolve Any Merge Conflicts:

If there are merge conflicts, resolve them as needed.

3. Push Changes After Pull:

After resolving conflicts, push your changes.

Updated `server.js` with Pull Before Push

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const simpleGit = require('simple-git');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
```

```

const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git'; // Replace with
your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {

```

```

const networkInterfaces = os.networkInterfaces();
let localIp = '';
for (const interfaceName in networkInterfaces) {
  const interfaces = networkInterfaces[interfaceName];
  for (const i of interfaces) {
    if (i.family === 'IPv4' && !i.internal) {
      localIp = i.address;
      break;
    }
  }
}
res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  startPeriodicIpUpdate();
});

const git = simpleGit();

// Function to fetch the local IP and update the IP file
async function fetchAndUpdateIp() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const ipData = { ip: localIp };
  fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

  try {
    await git.pull('origin', GITHUB_BRANCH); // Pull before pushing
    await git.add(IP_FILE_PATH);
    await git.commit('Update IP address');
    await git.push('origin', GITHUB_BRANCH);
    console.log('IP address updated successfully.');
  } catch (error) {
    console.error('Error updating IP address:', error);
  }
}

// Function to start periodic IP updates
function startPeriodicIpUpdate() {
  fetchAndUpdateIp(); // Initial call to update immediately
  setInterval(fetchAndUpdateIp, 3600000); // Update every hour
}

```

Run Your Server

1. Start the Server:

bash

node server.js

By pulling the remote changes before attempting to push, you should be able to integrate the remote changes and push your local updates successfully.

Let me know if you encounter any further issues or need additional assistance!



I got this message when running server.js

```
Error updating IP address: Error: From https://github.com/taplists/taplists.github.io
```

```
* branch      main    -> FETCH_HEAD
```

```
error: Your local changes to the following files would be overwritten by merge:
```

```
  .DS_Store
```

```
  server.js
```

```
Please commit your changes or stash them before you merge.
```

```
Aborting
```

```
  at Object.action (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1445:25)
```

```
  at PluginStore.exec (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1494:29)
```

```
  at /Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1921:43
```

```
  at new Promise (<anonymous>)
```

```
  at GitExecutorChain.handleTaskData (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1919:16)
```

```
  at GitExecutorChain.<anonymous> (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1903:44)
```

```
  at Generator.next (<anonymous>)
```

```
  at fulfilled (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:52:24)
```

```
  at process.processTicksAndRejections (node:internal/process/task_queues:95:5) {task: {...}, stack: 'Error: From https://github.com/taplists/taplists.github.io (node:internal/process/task_queues:95:5)', message: 'From https://github.com/taplists/taplists.github.io (node:internal/process/task_queues:95:5)',
```

```
  message: 'From https://github.com/taplists/taplists.github.io (node:internal/process/task_queues:95:5)',
```

```
  Aborting
```

```
}
```



The error message indicates that there are local changes in your repository that would be overwritten by the merge, specifically `.**DS_Store**` and `server.js`. To resolve this, you need to either commit or stash your local changes before pulling the remote changes.

Step-by-Step Guide

1. Commit Local Changes:

If you want to keep your local changes, commit them before pulling.

```
bash
```

```
git add .
```

```
git commit -m "Save local changes before pulling"
```

2. Stash Local Changes:

If you don't want to commit the changes but still need to save them temporarily, use `git stash` to stash the changes before pulling.

```
bash
```

```
git stash
```

3. Pull Remote Changes:

After committing or stashing your changes, pull the remote changes.

```
bash
```

```
git pull origin main --rebase
```

4. Apply Stashed Changes (if stashed):

If you used `git stash`, apply the stashed changes after pulling.

```
bash
```

```
git stash pop
```

5. Handle Conflicts:

If there are conflicts, resolve them, add the resolved files, and continue the rebase.

```
bash
```

```
git add <resolved_file>
git rebase --continue
```

Example Commands

Here are the commands all together:

```
bash
```

```
# If you want to commit local changes
git add .
git commit -m "Save local changes before pulling"

# If you want to stash local changes instead
# git stash
```

```
# Pull remote changes with rebase
git pull origin main --rebase

# Apply stashed changes if you used git stash
# git stash pop

# Resolve any merge conflicts, if they occur
# Open the conflicting files, resolve conflicts, and then add them
git add <resolved_file>
git rebase --continue

# Push changes to the remote repository
git push -u origin main
```

Update `server.js` to Handle the Pull and Push

Here's how you can update `server.js` to handle the situation more gracefully by stashing changes before pulling and applying the stash after pulling:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const simpleGit = require('simple-git');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git'; // Replace with your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});
```

```

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  startPeriodicIpUpdate();
});

const git = simpleGit();

// Function to fetch the local IP and update the IP file
async function fetchAndUpdateIp() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
}

```

```

        }

    const ipData = { ip: localIp };
    fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

    try {
        await git.stash(); // Stash changes before pulling
        await git.pull('origin', GITHUB_BRANCH); // Pull before pushing
        await git.stash(['pop']); // Apply stashed changes after pulling
        await git.add(IP_FILE_PATH);
        await git.commit('Update IP address');
        await git.push('origin', GITHUB_BRANCH);
        console.log('IP address updated successfully.');
    } catch (error) {
        console.error('Error updating IP address:', error);
    }
}

// Function to start periodic IP updates
function startPeriodicIpUpdate() {
    fetchAndUpdateIp(); // Initial call to update immediately
    setInterval(fetchAndUpdateIp, 3600000); // Update every hour
}

```

By following these steps and updating the `server.js` to handle stashing, pulling, and applying the stash, you should be able to avoid the issues related to local changes conflicting with remote changes.

Let me know if you encounter any further issues or need additional assistance!



Got this message when running server.js

Error updating IP address: Error: CONFLICT (modify/delete): server.js deleted in Updated upstream and modified in Stashed changes. Version Stashed changes of server.js left in tree.

Auto-merging .DS_Store

CONFLICT (content): Merge conflict in .DS_Store

warning: Cannot merge binary files: .DS_Store (Updated upstream vs. Stashed changes)

at Object.action (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1445:25)

at PluginStore.exec (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1494:29)

at /Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1921:43

at new Promise (<anonymous>)

at GitExecutorChain.handleTaskData (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1919:16)

at GitExecutorChain.<anonymous> (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:1903:44)

at Generator.next (<anonymous>)

at fulfilled (/Users/duncananderson/Desktop/Taplist-Project/node_modules/simple-git/dist/cjs/index.js:52:24) {task: {...}, stack: 'Error: CONFLICT (modify/delete): server.js de...e_modules/simple-git/dist/cjs/index.js:52:24', message: 'CONFLICT (modify/delete): server.js deleted'}

```
...tore (Updated upstream vs. Stashed changes)
'}
```



The error message indicates that there are conflicts between the changes in your local repository and the changes in the remote repository, specifically with the `.**DS_Store**` and `server.js` files.

To resolve this, you'll need to manually handle the merge conflicts. Here's how you can do that:

Step-by-Step Guide to Resolve Merge Conflicts

1. Check the Status:

Check the status of your repository to see which files are in conflict.

```
bash
git status
```

2. Resolve Conflicts:

Open the conflicting files and resolve the conflicts. For text files like `server.js`, you will need to manually merge the changes. For binary files like `.**DS_Store**`, you might need to decide which version to keep.

3. Add Resolved Files:

After resolving the conflicts, add the resolved files to the staging area.

```
bash
git add <resolved_file>
```

4. Continue the Rebase or Merge:

If you were in the middle of a rebase, continue the rebase. If you were merging, finalize the merge.

```
bash
git rebase --continue
```

or

```
bash
git merge --continue
```

5. Commit and Push Changes:

Commit the resolved changes and push them to the remote repository.

```
bash
```

```
git commit -m "Resolve merge conflicts"
git push origin main
```

Example Commands

Here's a complete set of commands to resolve the merge conflicts and push your changes:

```
bash
```

```
# Check the status to see which files are in conflict
git status

# Open the conflicting files and resolve the conflicts
# For example, resolve conflicts in server.js

# Add the resolved files to the staging area
git add server.js
git add .DS_Store

# Continue the rebase or merge
git rebase --continue

# If you were merging, use
# git merge --continue

# Commit the resolved changes
git commit -m "Resolve merge conflicts"

# Push the changes to the remote repository
git push origin main
```

Update `server.js` to Handle Conflicts

After resolving the conflicts manually, you might want to update your `server.js` script to avoid these conflicts in the future. You can add more robust error handling and ensure that your `.`DS_Store` file is ignored.

1. Add `.`DS_Store` to `.`gitignore`:

Create or update your `.`gitignore` file to ignore `.`DS_Store` files.

.gitignore:

```
.DS_Store
```

2. Update `server.js`:

Here's an updated version of `server.js` that includes more robust error handling:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const simpleGit = require('simple-git');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git'; // Replace with your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {

```

```

        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    startPeriodicIpUpdate();
});

const git = simpleGit();

// Function to fetch the local IP and update the IP file
async function fetchAndUpdateIp() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const ipData = { ip: localIp };
    fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

    try {
        await git.stash(); // Stash changes before pulling
        await git.pull('origin', GITHUB_BRANCH); // Pull before pushing
        await git.stash(['pop']); // Apply stashed changes after pulling
        await git.add(IP_FILE_PATH);
        await git.commit('Update IP address');
        await git.push('origin', GITHUB_BRANCH);
        console.log('IP address updated successfully.');
    } catch (error) {
        console.error('Error updating IP address:', error);
        await git.stash(['pop']); // Ensure stash is popped even if error occurs
    }
}

// Function to start periodic IP updates
function startPeriodicIpUpdate() {
    fetchAndUpdateIp(); // Initial call to update immediately
}

```

```

        setInterval(fetchAndUpdateIp, 3600000); // Update every hour
    }
}

```

By following these steps and updating the `server.js` to handle stashing and popping changes, you should be able to resolve the conflicts and push your local updates successfully.

Let me know if you encounter any further issues or need additional assistance!



Well unfortunately github has now deleted my project locally



I apologize for the trouble you're experiencing. Let's try to recover your project and make sure everything is set up correctly.

Recovering Your Project

1. Check for Local Backups:

Sometimes IDEs or operating systems keep temporary backups. Check if you have any local backups of your project directory.

2. Check Git Logs:

If you have a Git repository initialized, you can check the Git logs to see if your files are still recoverable.

```
bash
```

```
git reflog
```

This command shows you a history of changes in your repository, and you might be able to find a commit that includes your project files.

3. Revert to a Previous Commit:

If you find a commit that has your project files, you can reset your repository to that commit.

```
bash
```

```
git reset --hard <commit-hash>
```

Starting Fresh

If you're unable to recover your project, here's how you can set up your project again:

1. Clone the Remote Repository:

Start by cloning your GitHub repository to your local machine.

```
bash
```

```
git clone https://github.com/taplists/taplists.github.io.git
cd taplists.github.io
```

2. Recreate Your Project Files:

Recreate your `server.js`, `package.json`, and any other necessary files.

3. Install Dependencies:

Install the necessary Node.js dependencies.

```
bash
```

```
npm install express body-parser express-session simple-git node-fetch
```

4. Add Your Project Files to Git:

Add your project files to the Git repository and commit them.

```
bash
```

```
git add .
git commit -m "Recreate project files"
git push origin main
```

Ensuring GitHub and Local Sync

1. Configure ` .gitignore`:

Create a ` .gitignore` file to ignore unnecessary files like ` .DS_Store`.

.gitignore:

```
node_modules/
.DS_Store
```

2. Update `server.js`:

Here is an updated version of `server.js` with error handling and stashing changes:

server.js:

javascript

```

import express from 'express';
import bodyParser from 'body-parser';
import path from 'path';
import { fileURLToPath } from 'url';
import fs from 'fs';
import session from 'express-session';
import os from 'os';
import simpleGit from 'simple-git';

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password';
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git';
const GITHUB_BRANCH = 'main';

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key',
    resave: false,
    saveUninitialized: true
}));

const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

```

```

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  startPeriodicIpUpdate();
});

const git = simpleGit();

async function fetchAndUpdateIp() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const ipData = { ip: localIp };
  fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

  try {
    await git.stash();
    await git.pull('origin', GITHUB_BRANCH);
    await git.stash(['pop']);
    await git.add(IP_FILE_PATH);
    await git.commit('Update IP address');
    await git.push('origin', GITHUB_BRANCH);
    console.log('IP address updated successfully.');
  } catch (error) {
    console.error('Error updating IP address:', error);
    await git.stash(['pop']);
  }
}

function startPeriodicIpUpdate() {
  fetchAndUpdateIp();
  setInterval(fetchAndUpdateIp, 3600000);
}

```

3. Test the Setup:

Ensure everything is working as expected by running the server and verifying the changes.

```
bash
```

```
node server.js
```

By following these steps, you should be able to recover or recreate your project and set up the Git repository correctly. Let me know if you need any further assistance!



I had to revert to an old back up and lost all github functionality



I'm sorry to hear that you lost your GitHub functionality. Let's set up your project again with Git and GitHub from scratch. Here are the steps to initialize a new Git repository, link it to your GitHub repository, and push your changes:

Step-by-Step Guide

1. Initialize a New Git Repository:

Navigate to your project directory and initialize a new Git repository.

```
bash
```

```
cd /path/to/your/project  
git init
```

2. Add Your Files to the Repository:

Add all your project files to the staging area.

```
bash
```

```
git add .
```

3. Commit the Changes:

Commit the files you just added.

```
bash
```

```
git commit -m "Initial commit"
```

4. Add the Remote Repository:

Link your local repository to the remote repository on GitHub.

```
bash
```

```
git remote add origin https://github.com/taplists/taplists.github.io.git
```

5. Push Your Changes:

Push your changes to the remote repository. If your default branch is `main`, use `main`. If it's `master`, use `master`.

```
bash
```

```
git push -u origin main
```

If you encounter an error with `main`, try using `master` instead:

```
bash
```

```
git push -u origin master
```

Example Commands

Here's the complete set of commands:

```
bash
```

```
# Navigate to your project directory
cd /path/to/your/project

# Initialize a new Git repository
git init

# Add all files to the staging area
git add .

# Commit the changes
git commit -m "Initial commit"

# Add the remote repository
git remote add origin https://github.com/taplists/taplists.github.io.git

# Push changes to the remote repository on the main branch
git push -u origin main
```

Verifying GitHub Pages Configuration

Ensure that GitHub Pages is configured correctly in your repository settings on GitHub:

1. Go to Your Repository Settings on GitHub:

- Navigate to your repository on GitHub.
- Click on the "Settings" tab.

2. Scroll Down to the GitHub Pages Section:

- In the "GitHub Pages" section, select the branch you want to use for GitHub Pages (e.g., `main` or `master`).
- Click "Save".

Restoring Your Project Files

If you have restored your project files from an old backup, ensure all necessary files are in place, including:

- `server.js`
- `package.json`
- Any other project files

Example `server.js`

Here is an example `server.js` file to get you started:

javascript

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const simpleGit = require('simple-git');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const GITHUB_REPO = 'https://github.com/taplists/taplists.github.io.git'; // Replace with your GitHub repository
const GITHUB_BRANCH = 'main'; // Replace with your GitHub branch

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
}
```

```

};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedin = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    startPeriodicIpUpdate();
});

const git = simpleGit();

async function fetchAndUpdateIp() {

```

```

const networkInterfaces = os.networkInterfaces();
let localIp = '';
for (const interfaceName in networkInterfaces) {
  const interfaces = networkInterfaces[interfaceName];
  for (const i of interfaces) {
    if (i.family === 'IPv4' && !i.internal) {
      localIp = i.address;
      break;
    }
  }
}

const ipData = { ip: localIp };
fs.writeFileSync(IP_FILE_PATH, JSON.stringify(ipData, null, 2));

try {
  await git.stash();
  await git.pull('origin', GITHUB_BRANCH);
  await git.stash(['pop']);
  await git.add(IP_FILE_PATH);
  await git.commit('Update IP address');
  await git.push('origin', GITHUB_BRANCH);
  console.log('IP address updated successfully.');
} catch (error) {
  console.error('Error updating IP address:', error);
  await git.stash(['pop']);
}
}

function startPeriodicIpUpdate() {
  fetchAndUpdateIp();
  setInterval(fetchAndUpdateIp, 3600000);
}

```

By following these steps, you should be able to reinitialize your Git repository, link it to your GitHub repository, and push your changes. Let me know if you need any further assistance!



I think I want to scrap the github idea and use some sort of DDNS solution to keep from losing track of the 1918 address the taplist is hosted on



Using a Dynamic DNS (DDNS) service is a great way to keep track of your local IP address, especially if your network's IP address changes frequently. Here's how you can set up a DDNS service to keep your IP address updated.

Step-by-Step Guide to Set Up DDNS

1. Choose a DDNS Provider:

There are several free and paid DDNS providers available. Some popular options include:

- o No-IP (<https://www.noip.com/>)
- o DynDNS (<https://dyn.com/dns/>)
- o Duck DNS (<https://www.duckdns.org/>)

2. Sign Up and Create a Hostname:

- Sign up for an account on your chosen DDNS provider.
- Create a hostname that will be used to access your local server.

3. Install DDNS Update Client:

Many DDNS providers offer their own update clients that you can install on your local machine. These clients will automatically update the DDNS provider with your current IP address.

For example, if you're using No-IP, you can download and install their Dynamic Update Client.

Alternatively, you can use a router that supports DDNS updates directly.

4. Configure the Update Client:

Follow the instructions provided by your DDNS provider to configure the update client with your account details and the hostname you created.

Example Using No-IP

Here's an example of setting up DDNS using No-IP:

Step 1: Sign Up and Create a Hostname

1. Go to <https://www.noip.com/>.
2. Sign up for a free account.
3. Create a hostname (e.g., `mytaplist.ddns.net`).

Step 2: Install No-IP Update Client

1. Download the No-IP DUC (Dynamic Update Client) for your operating system from <https://www.noip.com/download>.
2. Install the client on your machine.

Step 3: Configure No-IP Update Client

1. Open the No-IP DUC.
2. Log in with your No-IP account credentials.
3. Select the hostname you created.
4. The client will now automatically update your IP address with No-IP.

Integrating DDNS with Your Server

You can configure your server to periodically print the DDNS hostname instead of the local IP address. Here's an example of how you might do this:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
```

```

const session = require('express-session');
const os = require('os');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DDNS_HOSTNAME = 'mytaplist.ddns.net'; // Replace with your DDNS hostname

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 'your_secret_key', // Replace with a secure key
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
  if (req.path === '/admin.html' || req.path === '/color-admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

```

```

// Endpoint to get DDNS hostname
app.get('/hostname', (req, res) => {
  res.json({ hostname: DDNS_HOSTNAME });
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

// Function to fetch the DDNS hostname and update the IP file
async function fetchAndUpdateHostname() {
  const hostnameData = { hostname: DDNS_HOSTNAME };
  fs.writeFileSync('hostname.json', JSON.stringify(hostnameData, null, 2));
}

// Function to start periodic hostname updates
function startPeriodicHostnameUpdate() {
  fetchAndUpdateHostname(); // Initial call to update immediately
  setInterval(fetchAndUpdateHostname, 3600000); // Update every hour
}

startPeriodicHostnameUpdate();

```

Accessing Your Server

After setting up DDNS, you can access your server using the hostname you created (e.g., `mytaplist.ddns.net:3000`).

Conclusion

Using DDNS is a reliable way to keep track of your local server's IP address even if it changes. It simplifies access and ensures you don't lose track of the server's location.

Let me know if you need any further assistance!



I don't think we'll even need to print the DDNS name says it wont be changing, we'll already know what it is



Great! Since you already know the DDNS hostname and it won't be changing, you can simply use the hostname directly without the need to fetch or update it periodically. Here's a streamlined version of your `server.js` file with DDNS setup:

Streamlined `server.js`

You can simplify the setup by removing unnecessary code related to fetching or updating the hostname and focus on serving your application.

server.js:

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DDNS_HOSTNAME = 'mytaplist.ddns.net'; // Replace with your DDNS hostname

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {

```

```

fs.readFile(COLORS_FILE_PATH, (err, data) => {
  if (err) {
    res.status(500).send('Error reading colors file');
  } else {
    res.json(JSON.parse(data));
  }
});

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  console.log(`Access your server at http://${DDNS_HOSTNAME}:${PORT}`);
});

```

Running Your Server

1. Start the Server:

```

bash

node server.js

```

2. Access Your Server:

Open your browser and navigate to your DDNS hostname:

```

url

http://mytaplist.ddns.net:3000

```

Using DDNS

By using DDNS, you ensure that even if your local IP changes, the hostname (``mytaplist.ddns.net``) will always point to your server, making it easier to access your server from different devices.

Conclusion

This setup allows you to host your taplist locally and access it using a consistent hostname provided by the DDNS service. Let me know if you need any further assistance!



So no-ip ended up not working because it can't function with 1918 addresses, only the public IP address. Can we try something like DDClient with the interface set to wlan0 rather than an IP checking external website caused the client to pull the private address for the update to the dynamic DNS provider



Sure, you can use `ddclient` to update your Dynamic DNS provider with your public IP address by configuring it to use the interface `wlan0`. Here's how you can set it up:

Step-by-Step Guide to Set Up `ddclient` with `wlan0`

Step 1: Install `ddclient`

1. For Debian/Ubuntu:

```
bash
sudo apt update
sudo apt install ddclient
```

2. For RedHat/CentOS:

```
bash
sudo yum install ddclient
```

Step 2: Configure `ddclient`

1. Edit the configuration file:

The configuration file is usually located at `/etc/ddclient.conf`. You might need to create it if it doesn't exist.

```
bash
sudo nano /etc/ddclient.conf
```

2. Add the following configuration:

Replace the placeholder values with your actual details.

```
bash
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=your-ddns-provider.com
protocol=dyndns2
login=your-ddns-login
password=your-ddns-password
your-ddns-hostname.ddns.net
# Check every 300 seconds
# Record PID in this file
# Use SSL to connect to the DDNS provider
# Get IP address from the specified interface
# Your DDNS provider
# Protocol used by your DDNS provider
# Your DDNS login
# Your DDNS password
# Your DDNS hostname
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

Step 3: Restart `ddclient`

1. Restart the `ddclient` service to apply the new configuration:

```
bash
sudo systemctl restart ddclient
```

2. Enable `ddclient` to start on boot:

```
bash
sudo systemctl enable ddclient
```

Verify `ddclient` is Working

1. Check the status of `ddclient`:

```
bash
sudo systemctl status ddclient
```

2. View the logs to ensure it's updating correctly:

```
bash
tail -f /var/log/syslog | grep ddclient
```

Example Configuration

Here is a sample configuration for `ddclient` with No-IP as the DDNS provider:

/etc/ddclient.conf:

```
bash
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=dynupdate.no-ip.com
protocol=dyndns2
```

```
login=your-noip-username
password=your-noip-password
your-hostname.ddns.net
```

Update Your Server Code

Since you will have a consistent DDNS hostname, you can use it directly in your server code. There is no need to fetch or update the hostname dynamically. Your `server.js` will look like this:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DDNS_HOSTNAME = 'your-hostname.ddns.net'; // Replace with your DDNS hostname

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
```

```

});  

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});  

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});  

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});  

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  console.log(`Access your server at http://${DDNS_HOSTNAME}:${PORT}`);
});

```

Accessing Your Server

You can now access your server using your DDNS hostname:

```

url  

http://your-hostname.ddns.net:3000

```

This setup ensures that your DDNS provider is updated with the public IP address of your router, and you can consistently access your server using the DDNS hostname.

Let me know if you need any further assistance!



The operation couldn't be completed. Unable to locate a Java Runtime.
Please visit <http://www.java.com> for information on installing Java.



It looks like the error you're encountering indicates that `ddclient` or another part of your setup might be requiring Java to run. Let's address this by installing Java and then ensuring that `ddclient` is correctly configured to work without needing an external IP check.

Step-by-Step Guide to Install Java

Step 1: Install Java Runtime Environment (JRE)

1. For Debian/Ubuntu:

```
bash
sudo apt update
sudo apt install default-jre
```

2. For RedHat/CentOS:

```
bash
sudo yum install java-1.8.0-openjdk
```

3. For macOS:

- Download and install Java from the official [Java website](#).

Step 2: Verify Java Installation

1. Check Java version:

```
bash
java -version
```

You should see output indicating the installed Java version.

Step-by-Step Guide to Configure `ddclient` with `wlan0`

Step 3: Install `ddclient`

1. For Debian/Ubuntu:

```
bash
sudo apt update
sudo apt install ddclient
```

2. For RedHat/CentOS:

```
bash
```

```
sudo yum install ddclient
```

Step 4: Configure `ddclient`

1. Edit the configuration file:

The configuration file is usually located at `/etc/ddclient.conf`. You might need to create it if it doesn't exist.

```
bash
```

```
sudo nano /etc/ddclient.conf
```

2. Add the following configuration:

Replace the placeholder values with your actual details. This configuration uses the `wlan0` interface to get the IP address.

```
bash
```

```
daemon=300                                     # Check every 300 seconds
pid=/var/run/ddclient.pid                       # Record PID in this file
ssl=yes                                         # Use SSL to connect to the DDNS provider
use=if, if=wlan0                                # Get IP address from the specified interface
server=your-ddns-provider.com                  # Your DDNS provider
protocol=dyndns2                                 # Protocol used by your DDNS provider
login=your-ddns-login                           # Your DDNS login
password=your-ddns-password                     # Your DDNS password
your-ddns-hostname.ddns.net                      # Your DDNS hostname
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

Step 5: Restart `ddclient`

1. Restart the `ddclient` service to apply the new configuration:

```
bash
```

```
sudo systemctl restart ddclient
```

2. Enable `ddclient` to start on boot:

```
bash
```

```
sudo systemctl enable ddclient
```

Verify `ddclient` is Working

1. Check the status of `ddclient`:

```
bash
```

```
sudo systemctl status ddclient
```

2. View the logs to ensure it's updating correctly:

```
bash
```

```
tail -f /var/log/syslog | grep ddclient
```

Example Configuration

Here is a sample configuration for `ddclient` with No-IP as the DDNS provider:

/etc/ddclient.conf:

```
bash
```

```
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=dynupdate.no-ip.com
protocol=dyndns2
login=your-noip-username
password=your-noip-password
your-hostname.ddns.net
```

Running Your Server

1. Start the Server:

```
bash
```

```
node server.js
```

2. Access Your Server:

Open your browser and navigate to your DDNS hostname:

```
url
http://your-hostname.ddns.net:3000
```

This setup ensures that your DDNS provider is updated with the public IP address of your router, and you can consistently access your server using the DDNS hostname.

Let me know if you need any further assistance!



I'd like to revisit this. Additional piece of info, I'm currently making this project in macOS "Can we try something like DDClient with the interface set to wlan0 rather than an IP checking external website caused the client to pull the private address for the update to the dynamic DNS provider"



Sure, let's set up `ddclient` on macOS to update your Dynamic DNS provider with the IP address from the `wlan0` interface. This will help ensure that the IP address used for updates is your public IP address, rather than relying on an external IP checking service.

Step-by-Step Guide to Set Up `ddclient` on macOS

Step 1: Install `ddclient`

First, you need to install `ddclient`. You can do this using Homebrew.

1. **Install Homebrew** (if you haven't already):

Open Terminal and run:

```
bash
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. **Install `ddclient` using Homebrew:**

```
bash
brew install ddclient
```

Step 2: Configure `ddclient`

1. Create/Edit the configuration file:

The configuration file should be created at `/usr/local/etc/ddclient/ddclient.conf`. If it doesn't exist, create it.

```
bash
sudo nano /usr/local/etc/ddclient/ddclient.conf
```

2. Add the following configuration:

Replace the placeholder values with your actual details. This configuration uses the `wlan0` interface to get the IP address.

```
bash

daemon=300                                # Check every 300 seconds
pid=/var/run/ddclient.pid                   # Record PID in this file
ssl=yes                                     # Use SSL to connect to the DDNS provider
use=if, if=wlan0                            # Get IP address from the specified interface
server=your-ddns-provider.com              # Your DDNS provider
protocol=dyndns2                            # Protocol used by your DDNS provider
login=your-ddns-login                      # Your DDNS login
password=your-ddns-password                # Your DDNS password
your-ddns-hostname.ddns.net                 # Your DDNS hostname
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

Step 3: Configure the `ddclient` Service

1. Create a LaunchDaemon plist for `ddclient`:

Create a file at `/Library/LaunchDaemons/org.ddclient.plist` to ensure `ddclient` runs as a service.

```
bash
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

2. Add the following plist configuration:

```
xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
```

```

<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient.conf</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>

```

3. Save and exit:

Press `Ctrl+X` , then `Y` , and then `Enter` to save the file and exit the editor.

4. Load the LaunchDaemon:

```

bash

sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist

```

Step 4: Verify `ddclient` is Working

1. Check the status of `ddclient`:

```

bash

sudo launchctl list | grep ddclient

```

2. View the logs to ensure it's updating correctly:

```

bash

tail -f /var/log/system.log | grep ddclient

```

Example Configuration

Here is a sample configuration for `ddclient` with No-IP as the DDNS provider:

/usr/local/etc/ddclient/ddclient.conf:

bash

```
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=dynupdate.no-ip.com
protocol=dyndns2
login=your-noip-username
password=your-noip-password
your-hostname.ddns.net
```

Running Your Server

1. Start the Server:

bash

```
node server.js
```

2. Access Your Server:

Open your browser and navigate to your DDNS hostname:

url

```
http://your-hostname.ddns.net:3000
```

This setup ensures that your DDNS provider is updated with the public IP address of your router, and you can consistently access your server using the DDNS hostname.

Let me know if you need any further assistance!



when I try to brew install ddclient, I get this response

```
"cambro2:Tplist-Project duncananderson$ brew install ddclient
Error: The current working directory must be readable to duncananderson to run brew."
```



The error message indicates that the current working directory does not have the appropriate read permissions for your user. To resolve this, you need to change the directory to one that has the correct permissions or fix the permissions of the current directory.

Step-by-Step Guide to Resolve the Issue

1. Change Directory to Home Directory

Change your current directory to your home directory or any other directory that has the correct permissions.

```
bash
```

```
cd ~
```

Then try running the `brew install` command again:

```
bash
```

```
brew install ddclient
```

2. Fix Permissions of the Current Directory

If you prefer to fix the permissions of the current directory, you can do so using the `chmod` command.

First, find out your current directory:

```
bash
```

```
pwd
```

Then change the permissions to make it readable for your user:

```
bash
```

```
sudo chmod 755 /path/to/current/directory
```

After fixing the permissions, try running the `brew install` command again:

```
bash
```

```
brew install ddclient
```

Complete Process for Installing and Configuring `ddclient`

After successfully installing `ddclient`, follow the previously outlined steps to configure it:

Step 1: Install `ddclient` Using Homebrew

```
bash
brew install ddclient
```

Step 2: Configure `ddclient`

Create or edit the configuration file at `/usr/local/etc/ddclient/ddclient.conf`:

```
bash
sudo nano /usr/local/etc/ddclient/ddclient.conf
```

Add the following configuration, replacing placeholder values with your actual details:

```
bash
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use;if, if=wlan0
server=dynamicupdate.no-ip.com
protocol=dyndns2
login=your-noip-username
password=your-noip-password
your-hostname.ddns.net
```

Save and exit the editor.

Step 3: Configure the `ddclient` Service

Create a LaunchDaemon plist for `ddclient`:

```
bash
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

Add the following plist configuration:

```
xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
```

```
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient.conf</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

Save and exit the editor.

Load the LaunchDaemon:

```
bash

sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist
```

Step 4: Verify `ddclient` is Working

Check the status of `ddclient`:

```
bash

sudo launchctl list | grep ddclient
```

View the logs to ensure it's updating correctly:

```
bash

tail -f /var/log/system.log | grep ddclient
```

Running Your Server

Start your server:

```
bash

node server.js
```

Access your server using the DDNS hostname:

```
url
```

```
http://your-hostname.ddns.net:3000
```

By following these steps, you should be able to set up `ddclient` on macOS and ensure your DDNS provider is updated correctly. Let me know if you encounter any further issues or need further assistance!



So how does ddclient work and what is my DDNS hostname?



How `ddclient` Works

`ddclient` is a Perl-based script that updates your dynamic DNS (DDNS) provider with the current IP address of your machine. It can detect your IP address using various methods and update your DDNS provider to ensure that your hostname points to the correct IP address, even if your IP address changes frequently.

How to Set Up `ddclient`

Here is a more detailed step-by-step guide to set up `ddclient` on macOS:

Step 1: Install `ddclient`

First, let's install `ddclient` using Homebrew.

1. **Install Homebrew** (if you haven't already):

Open Terminal and run:

```
bash
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. **Install `ddclient` using Homebrew:**

```
bash
cd ~ # Change to your home directory to avoid permission issues
brew install ddclient
```

Step 2: Configure `ddclient`

1. Create/Edit the configuration file:

Create or edit the configuration file at `/usr/local/etc/ddclient.conf`.

```
bash
sudo nano /usr/local/etc/ddclient.conf
```

2. Add the following configuration:

Replace the placeholder values with your actual details. This configuration uses the `wlan0` interface to get the IP address.

```
bash
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=dyndns.no-ip.com
protocol=dyndns2
login=your-noip-username
password=your-noip-password
your-hostname.ddns.net
# Check every 300 seconds
# Record PID in this file
# Use SSL to connect to the DDNS provider
# Get IP address from the specified interface
# Your DDNS provider
# Protocol used by your DDNS provider
# Your DDNS login
# Your DDNS password
# Your DDNS hostname
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

Step 3: Configure the `ddclient` Service

1. Create a LaunchDaemon plist for `ddclient`:

Create a file at `/Library/LaunchDaemons/org.ddclient.plist` to ensure `ddclient` runs as a service.

```
bash
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

2. Add the following plist configuration:

```
xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
```

```

<dict>
    <key>Label</key>
    <string>org.ddclient</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/local/bin/ddclient</string>
        <string>-file</string>
        <string>/usr/local/etc/ddclient.conf</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>KeepAlive</key>
    <true/>
</dict>
</plist>

```

3. Save and exit:

Press `Ctrl+X` , then `Y` , and then `Enter` to save the file and exit the editor.

4. Load the LaunchDaemon:

```

bash

sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist

```

Step 4: Verify `ddclient` is Working

1. Check the status of `ddclient`:

```

bash

sudo launchctl list | grep ddclient

```

2. View the logs to ensure it's updating correctly:

```

bash

tail -f /var/log/system.log | grep ddclient

```

Understanding Your DDNS Hostname

Your DDNS hostname is the address you created with your DDNS provider. This hostname will always point to your current public IP address, allowing you to access your server even if your IP address changes.

Example Configuration for No-IP

If you are using No-IP as your DDNS provider, here is a complete example of how your `ddclient` configuration might look:

/usr/local/etc/ddclient/ddclient.conf:

```
bash

daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=w
```



So my DDNS is no-ip, my login is anderwolfe@gmail.com, my password is ssjducks, the hostname is sample-taplist.ddns.net



Great! Let's set up your `ddclient` with the provided details for No-IP. Here are the steps to configure `ddclient` on your macOS system:

Step-by-Step Guide to Set Up `ddclient` on macOS with No-IP

Step 1: Install `ddclient`

1. Install Homebrew (if you haven't already):

Open Terminal and run:

```
bash

/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. Install `ddclient` using Homebrew:

```
bash

cd ~ # Change to your home directory to avoid permission issues
brew install ddclient
```

Step 2: Configure `ddclient`

1. Create/Edit the configuration file:

Create or edit the configuration file at `/usr/local/etc/ddclient/ddclient.conf`.

```
bash
```

```
sudo nano /usr/local/etc/ddclient/ddclient.conf
```

2. Add the following configuration:

Replace the placeholder values with your actual details. This configuration uses the `wlan0` interface to get the IP address.

```
bash
```

```
daemon=300                                     # Check every 300 seconds
pid=/var/run/ddclient.pid                      # Record PID in this file
ssl=yes                                         # Use SSL to connect to the DDNS provider
use=if, if=wlan0                                # Get IP address from the specified interface
server=dynupdate.no-ip.com                      # No-IP DDNS provider
protocol=dyndns2                                 # Protocol used by No-IP
login=anderwolfe@gmail.com                      # Your No-IP login
password=ssjducks                                # Your No-IP password
sample-taplist.ddns.net                          # Your No-IP hostname
```

3. Save and exit:

Press `Ctrl+X` , then `Y` , and then `Enter` to save the file and exit the editor.

Step 3: Configure the `ddclient` Service

1. Create a LaunchDaemon plist for `ddclient`:

Create a file at `/Library/LaunchDaemons/org.ddclient.plist` to ensure `ddclient` runs as a service.

```
bash
```

```
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

2. Add the following plist configuration:

```
xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>org.ddclient</string>
<key>ProgramArguments</key>
<array>
<string>/usr/local/bin/ddclient</string>
```

```

<string>-file</string>
<string>/usr/local/etc/ddclient.conf</string>
</array>
<key>RunAtLoad</key>
<true/>
<key>KeepAlive</key>
<true/>
</dict>
</plist>

```

3. Save and exit:

Press `Ctrl+X` , then `Y` , and then `Enter` to save the file and exit the editor.

4. Load the LaunchDaemon:

```

bash

sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist

```

Step 4: Verify `ddclient` is Working

1. Check the status of `ddclient`:

```

bash

sudo launchctl list | grep ddclient

```

2. View the logs to ensure it's updating correctly:

```

bash

tail -f /var/log/system.log | grep ddclient

```

Running Your Server

Once `ddclient` is set up and running, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

```

bash

node server.js

```

2. Access your server:

Open your browser and navigate to:

```
url
http://sample-taplist.ddns.net:3000
```

This setup ensures that your No-IP hostname is always up to date with your current public IP address, allowing you to access your server using the DDNS hostname.

Let me know if you need any further assistance!



when I try to run ddclient using sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist

It says this:

```
cambro2:~ duncananderson$ sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist
Load failed: 5: Input/output error
Try running `launchctl bootstrap` as root for richer errors.
```



The error you're encountering suggests that there might be an issue with the `plist` configuration or its permissions. Let's troubleshoot this step by step:

Step-by-Step Guide to Troubleshoot and Fix `ddclient` LaunchDaemon

1. Check the `plist` Configuration

Ensure that your `plist` configuration is correct. Open the `plist` file and verify the contents:

```
bash
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

Here is the correct configuration for `ddclient`:

```
xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
```

```

<key>ProgramArguments</key>
<array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient.conf</string>
</array>
<key>RunAtLoad</key>
<true/>
<key>KeepAlive</key>
<true/>
</dict>
</plist>

```

Save and exit by pressing `Ctrl+X` , then `Y` , and then `Enter` .

2. Verify File Permissions

Ensure that the `plist` file has the correct permissions:

```

bash

sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist

```

3. Check for Errors with `launchctl bootstrap`

Run the following command to get more detailed error information:

```

bash

sudo launchctl bootstrap system /Library/LaunchDaemons/org.ddclient.plist

```

4. Check System Logs

Check the system logs for any additional error messages related to `ddclient` :

```

bash

sudo log show --predicate 'process == "ddclient"' --info --debug --last 1h

```

Example Configuration for No-IP

Here is a complete example of your `ddclient` configuration:

/usr/local/etc/ddclient/ddclient.conf:

bash

```
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=dynupdate.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net
```

Verify `ddclient` Configuration

1. Test `ddclient` Manually

Before setting it up as a service, test `ddclient` manually to ensure it works:

bash

```
sudo ddclient -daemon=0 -debug -verbose -noquiet
```

This command will run `ddclient` in the foreground with debug output, which can help identify any issues with the configuration.

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

bash

```
node server.js
```

2. Access your server:

Open your browser and navigate to:

url

```
http://sample-taplist.ddns.net:3000
```

Additional Troubleshooting

If you still encounter issues, please provide the output of the following commands to help diagnose the problem:

1. Output of `launchctl bootstrap`:

```
bash
```

```
sudo launchctl bootstrap system /Library/LaunchDaemons/org.ddclient.plist
```

2. Relevant log entries:

```
bash
```

```
sudo log show --predicate 'process == "ddclient"' --info --debug --last 1h
```

This will help to provide more context and identify the root cause of the issue. Let me know if you need any further assistance!



```
cambro2:~ duncananderson$ sudo launchctl bootstrap system  
/Library/LaunchDaemons/org.ddclient.plist  
Bootstrap failed: 5: Input/output error  
cambro2:~ duncananderson$ sudo ddclient -daemon=0 -debug -verbose -noquiet  
sudo: ddclient: command not found
```



It seems that the `ddclient` binary might not be correctly installed or available in your system's PATH. Let's ensure that `ddclient` is correctly installed and accessible.

Reinstalling `ddclient` Using Homebrew

1. Uninstall any existing `ddclient` (if applicable):

```
bash
```

```
brew uninstall ddclient
```

2. Reinstall `ddclient` using Homebrew:

```
bash
```

```
brew install ddclient
```

Verify Installation

Ensure that `ddclient` is correctly installed and accessible:

- 1. Check if `ddclient` is installed:**

```
bash
```

```
which ddclient
```

This should return the path to `ddclient`, something like `/usr/local/bin/ddclient`.

Manually Running `ddclient`

- 1. Test `ddclient` manually:**

```
bash
```

```
sudo /usr/local/bin/ddclient -daemon=0 -debug -verbose -noquiet
```

If this command works, it means `ddclient` is correctly installed and configured.

Setting Up `ddclient` as a LaunchDaemon

If `ddclient` runs correctly manually, proceed with setting it up as a LaunchDaemon:

- 1. Ensure the `plist` configuration is correct:**

Recheck the contents of `/Library/LaunchDaemons/org.ddclient.plist`:

```
xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient/ddclient.conf</string>
  </array>
  <key>RunAtLoad</key>
```

```

<true/>
<key>KeepAlive</key>
<true/>
</dict>
</plist>

```

2. Ensure the correct file permissions:

bash

```

sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist

```

3. Load the LaunchDaemon:

bash

```

sudo launchctl bootstrap system /Library/LaunchDaemons/org.ddclient.plist

```

Additional Troubleshooting

If you still encounter issues, here are some additional steps you can take:

1. Check the system logs for more detailed error messages:

bash

```

sudo log show --predicate 'process == "ddclient"' --info --debug --last 1h

```

2. Manually specify the path when running commands:

Ensure that you are referencing the correct path to `ddclient` when running commands. If Homebrew installed `ddclient` in `/usr/local/bin`, make sure this path is used.

Example Configuration for No-IP

Here is the complete configuration for `ddclient` with No-IP as the DDNS provider:

/usr/local/etc/ddclient/ddclient.conf:

bash

```

daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=dynupdate.no-ip.com
protocol=dyndns2

```

```
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net
```

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

```
bash
node server.js
```

2. Access your server:

Open your browser and navigate to:

```
url
http://sample-taplist.ddns.net:3000
```

By following these steps, you should be able to set up `ddclient` on macOS and ensure your DDNS provider is updated correctly. Let me know if you encounter any further issues or need further assistance!



brew install ddclient returns this:

Error: Permission denied @ dir_s_mkdir - /usr/local/var/cache/ddclient



The permission denied error when installing `ddclient` indicates that the current user does not have the necessary permissions to create a directory in `/usr/local/var/cache/`. To resolve this, you can change the ownership of the directory or run the installation with sudo.

Step-by-Step Guide to Resolve Permission Issues

Step 1: Change Ownership of the Directory

1. Change ownership of `/usr/local/var/cache`:

```
bash
```

```
sudo chown -R $(whoami) /usr/local/var/cache/
```

2. Install `ddclient` again:

```
bash
```

```
brew install ddclient
```

Step 2: Verify Installation

Ensure that `ddclient` is correctly installed and accessible:

1. Check if `ddclient` is installed:

```
bash
```

```
which ddclient
```

This should return the path to `ddclient`, something like `/usr/local/bin/ddclient`.

Step 3: Manually Run `ddclient`

1. Test `ddclient` manually:

```
bash
```

```
sudo /usr/local/bin/ddclient -daemon=0 -debug -verbose -noquiet
```

If this command works, it means `ddclient` is correctly installed and configured.

Setting Up `ddclient` as a LaunchDaemon

If `ddclient` runs correctly manually, proceed with setting it up as a LaunchDaemon:

1. Ensure the `plist` configuration is correct:

Recheck the contents of `/Library/LaunchDaemons/org.ddclient.plist`:

```
xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
```

```

<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient.conf</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>

```

2. Ensure the correct file permissions:

bash

```

sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist

```

3. Load the LaunchDaemon:

bash

```

sudo launchctl bootstrap system /Library/LaunchDaemons/org.ddclient.plist

```

Example Configuration for No-IP

Here is the complete configuration for `ddclient` with No-IP as the DDNS provider:

/usr/local/etc/ddclient/ddclient.conf:

bash

```

daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=dynupdate.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net

```

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

```
bash
node server.js
```

2. Access your server:

Open your browser and navigate to:

```
url
http://sample-taplist.ddns.net:3000
```

By following these steps, you should be able to set up `ddclient` on macOS and ensure your DDNS provider is updated correctly. Let me know if you encounter any further issues or need further assistance!



```
cambro2:~ duncananderson$ sudo launchctl bootstrap system
/Library/LaunchDaemons/org.ddclient.plist
Bootstrap failed: 5: Input/output error
```



The "Input/output error" indicates that there might still be an issue with the `plist` configuration or permissions. Let's try to troubleshoot and fix this step by step.

Step-by-Step Troubleshooting

Step 1: Verify `ddclient` Installation

Ensure that `ddclient` is installed correctly.

1. Check `ddclient` installation:

```
bash
which ddclient
```

This should return the path to `ddclient`, such as `/usr/local/bin/ddclient`.

2. Run `ddclient` manually to ensure it works:

```
bash
```

```
sudo /usr/local/bin/ddclient -daemon=0 -debug -verbose -noquiet
```

If this command works, it means `ddclient` is correctly installed and configured.

Step 2: Verify `plist` Configuration

Ensure the `plist` configuration is correct.

1. Open the `plist` file:

```
bash
```

```
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

2. Verify the contents:

Ensure the file contains the following configuration:

```
xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient.conf</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

Step 3: Check File Permissions

Ensure the `plist` file has the correct permissions.

1. Change ownership and permissions:

```
bash
```

```
sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist  
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist
```

Step 4: Load the LaunchDaemon

1. Load the LaunchDaemon:

```
bash
```

```
sudo launchctl bootstrap system /Library/LaunchDaemons/org.ddclient.plist
```

Step 5: Check Logs for More Information

If the error persists, check the logs for more detailed information.

1. View system logs:

```
bash
```

```
sudo log show --predicate 'process == "ddclient"' --info --debug --last 1h
```

This command will provide more detailed logs that might help identify the issue.

Alternative Method: Using `launchd` with `launchctl`

If the above steps do not resolve the issue, try using the `launchd` and `launchctl` commands directly.

1. Unload the `plist` (if previously loaded):

```
bash
```

```
sudo launchctl unload /Library/LaunchDaemons/org.ddclient.plist
```

2. Load the `plist` using `launchd`:

```
bash
```

```
sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist
```

3. Check the status:

```
bash
```

```
sudo launchctl list | grep ddclient
```

Example Configuration for No-IP

Here is the complete configuration for `ddclient` with No-IP as the DDNS provider:

/usr/local/etc/ddclient/ddclient.conf:

```
bash
```

```
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=wlan0
server=dynupdate.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net
```

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

```
bash
```

```
node server.js
```

2. Access your server:

Open your browser and navigate to:

```
url
```

```
http://sample-taplist.ddns.net:3000
```

By following these steps, you should be able to set up `ddclient` on macOS and ensure your DDNS provider is updated correctly. Let me know if you encounter any further issues or need further assistance!



I feel like wlan0 may not be the name of the interface on my machine, how do I check that



To check the network interfaces available on your macOS machine, you can use the `ifconfig` command. This command will list all the network interfaces along with their details.

Step-by-Step Guide to Check Network Interfaces on macOS

1. Open Terminal:

You can open Terminal from your Applications folder or by searching for it in Spotlight.

2. Run the `ifconfig` command:

```
bash
ifconfig
```

3. Identify the Wi-Fi interface:

Look for the interface that corresponds to your Wi-Fi connection. On macOS, this is usually named `en0` or `en1`. You'll see details such as IP addresses associated with each interface.

Here's an example of what the output might look like:

```
bash
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      ether xx:xx:xx:xx:xx:xx
      inet 192.168.1.2 netmask 0xffffffff00 broadcast 192.168.1.255
      ...
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      ether xx:xx:xx:xx:xx:xx
      inet 192.168.1.3 netmask 0xffffffff00 broadcast 192.168.1.255
      ...
```

In this example, `en0` and `en1` are network interfaces. The interface with `inet` showing your local IP address is the one you're looking for.

Update `ddclient` Configuration with Correct Interface

Once you've identified the correct interface (let's assume it's `en0`), you need to update your `ddclient` configuration.

1. Edit the `ddclient` configuration file:

```
bash

sudo nano /usr/local/etc/ddclient/ddclient.conf
```

2. Update the `use=if` line to reflect the correct interface:

```
bash

daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=en0          # Update to the correct interface name
server=dynupdate.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

Retry Loading `ddclient` as a LaunchDaemon

1. Ensure the correct file permissions:

```
bash

sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist
```

2. Unload the `plist` (if previously loaded):

```
bash

sudo launchctl unload /Library/LaunchDaemons/org.ddclient.plist
```

3. Load the `plist` using `launchctl`:

```
bash

sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist
```

4. Check the status:

```
bash
```

```
sudo launchctl list | grep ddclient
```

Running `ddclient` Manually

If you encounter any issues, try running `ddclient` manually to ensure it works:

```
bash
```

```
sudo /usr/local/bin/ddclient -daemon=0 -debug -verbose -noquiet
```

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

```
bash
```

```
node server.js
```

2. Access your server:

Open your browser and navigate to:

```
url
```

```
http://sample-taplist.ddns.net:3000
```

By following these steps, you should be able to identify the correct network interface and set up `ddclient` on macOS to update your DDNS provider correctly. Let me know if you encounter any further issues or need further assistance!



Do we need to generate a DDNS key from no-ip?



You typically do not need a specific DDNS key from No-IP for basic DDNS functionality. The standard way to use No-IP DDNS is by providing your No-IP account username and password, along with the hostname you wish to update.

Step-by-Step Guide to Set Up No-IP DDNS with `ddclient`

Let's ensure everything is correctly set up using your No-IP credentials.

Step 1: Install `ddclient`

1. **Install Homebrew** (if you haven't already):

Open Terminal and run:

```
bash
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. **Install `ddclient` using Homebrew:**

```
bash
cd ~ # Change to your home directory to avoid permission issues
brew install ddclient
```

Step 2: Configure `ddclient`

1. **Create/Edit the configuration file:**

Create or edit the configuration file at `/usr/local/etc/ddclient/ddclient.conf`.

```
bash
sudo nano /usr/local/etc/ddclient/ddclient.conf
```

2. **Add the following configuration:**

Replace the placeholder values with your actual details. Ensure that you use the correct network interface (`en0`, `en1`, etc.) that you identified earlier.

```
bash
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use;if, if=en0 # Update to the correct interface name
```

```
server=dynupdate.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com          # Your No-IP login
password=ssjducks                   # Your No-IP password
sample-taplist.ddns.net             # Your No-IP hostname
```

3. Save and exit:

Press `Ctrl+X` , then `Y` , and then `Enter` to save the file and exit the editor.

Step 3: Verify `ddclient` Installation

Ensure that `ddclient` is installed correctly.

1. Check `ddclient` installation:

```
bash
which ddclient
```

This should return the path to `ddclient` , such as `/usr/local/bin/ddclient` .

2. Run `ddclient` manually to ensure it works:

```
bash
sudo /usr/local/bin/ddclient -daemon=0 -debug -verbose -noquiet
```

This will run `ddclient` in the foreground with debug output, which can help identify any issues with the configuration.

Step 4: Set Up `ddclient` as a LaunchDaemon

If `ddclient` runs correctly manually, proceed with setting it up as a LaunchDaemon:

1. Create a LaunchDaemon plist for `ddclient` :

Create a file at `/Library/LaunchDaemons/org.ddclient.plist` .

```
bash
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

2. Add the following plist configuration:

```
xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>Label</key>
<string>org.ddclient</string>
<key>ProgramArguments</key>
<array>
<string>/usr/local/bin/ddclient</string>
<string>-file</string>
<string>/usr/local/etc/ddclient/ddclient.conf</string>
</array>
<key>RunAtLoad</key>
<true/>
<key>KeepAlive</key>
<true/>
</dict>
</plist>
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

4. Ensure the correct file permissions:

```
bash
```

```
sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist
```

5. Unload the `plist` (if previously loaded):

```
bash
```

```
sudo launchctl unload /Library/LaunchDaemons/org.ddclient.plist
```

6. Load the `plist` using `launchctl`:

```
bash
```

```
sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist
```

7. Check the status:

```
bash
```

```
sudo launchctl list | grep ddclient
```

Step 5: Verify `ddclient` is Working

1. Check the status of `ddclient`:

```
bash
```

```
sudo launchctl list | grep ddclient
```

2. View the logs to ensure it's updating correctly:

```
bash
```

```
tail -f /var/log/system.log | grep ddclient
```

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

```
bash
```

```
node server.js
```

2. Access your server:

Open your browser and navigate to:

```
url
```

```
http://sample-taplist.ddns.net:3000
```

By following these steps, you should be able to set up `ddclient` on macOS and ensure your DDNS provider is updated correctly. Let me know if you encounter any further issues or need further assistance!



When I run "tail -f /var/log/system.log | grep ddclient" the terminal just returns to an empty line



If the `tail -f /var/log/system.log | grep ddclient` command returns nothing, it might indicate that `ddclient` is not running or not generating logs. Let's go through the steps to ensure that `ddclient` is correctly configured and running.

Step-by-Step Troubleshooting

Step 1: Verify `ddclient` Configuration

Ensure the `ddclient` configuration file is correctly set up.

1. Edit the `ddclient` configuration file:

```
bash
sudo nano /usr/local/etc/ddclient/ddclient.conf
```

2. Ensure the configuration is correct:

```
bash
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=en0 # Replace 'en0' with your actual network interface
server=dynupdate.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

Step 2: Manually Run `ddclient` for Debugging

Run `ddclient` manually in debug mode to see if there are any issues.

1. Run `ddclient` manually:

```
bash
```

```
sudo /usr/local/bin/ddclient -daemon=0 -debug -verbose -noquiet
```

2. Check the output:

The command should provide detailed output, including any errors or issues with the configuration.

Step 3: Verify `plist` Configuration

Ensure the `plist` file for the LaunchDaemon is correctly set up.

1. Edit the `plist` file:

```
bash
```

```
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

2. Ensure the `plist` file contents are correct:

```
xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient.conf</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

4. Ensure the correct file permissions:

bash

```
sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist
```

Step 4: Load the LaunchDaemon

1. **Unload the `plist` (if previously loaded):**

bash

```
sudo launchctl unload /Library/LaunchDaemons/org.ddclient.plist
```

2. **Load the `plist` using `launchctl`:**

bash

```
sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist
```

3. **Check the status:**

bash

```
sudo launchctl list | grep ddclient
```

Step 5: Verify Logs

1. **Check the system logs for `ddclient` activity:**

bash

```
sudo log show --predicate 'process == "ddclient"' --info --debug --last 1h
```

Example Configuration for No-IP

Here is the complete configuration for `ddclient` with No-IP as the DDNS provider:

/usr/local/etc/ddclient/ddclient.conf:

bash

```
daemon=300
pid=/var/run/ddclient.pid
```

```
ssl=yes
use=if, if=en0 # Replace 'en0' with your actual network interface
server=dynupdate.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net
```

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

```
bash
node server.js
```

2. Access your server:

Open your browser and navigate to:

```
url
http://sample-taplist.ddns.net:3000
```

By following these steps, you should be able to identify any issues with `ddclient` and ensure it is running correctly on macOS. Let me know if you encounter any further issues or need further assistance!



When I try to save ddclient.conf it says [Cannot open file for writing: No such file or directory]



It looks like the directory `/usr/local/etc/ddclient/` does not exist, which is why you're unable to save the configuration file. Let's create the necessary directories and then create the configuration file.

Step-by-Step Guide to Create Necessary Directories and Configuration File

Step 1: Create the Necessary Directories

1. Create the directory:

```
bash
```

```
sudo mkdir -p /usr/local/etc/ddclient
```

Step 2: Create and Edit the Configuration File

1. Create and edit the configuration file:

```
bash
```

```
sudo nano /usr/local/etc/ddclient/ddclient.conf
```

2. Add the following configuration:

Replace the placeholder values with your actual details. Ensure that you use the correct network interface (``en0``, ``en1``, etc.) that you identified earlier.

```
bash
```

```
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use=if, if=en0 # Replace 'en0' with your actual network interface
server=dynupdate.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

Step 3: Verify `ddclient` Installation

Ensure that `ddclient` is installed correctly.

1. Check `ddclient` installation:

```
bash
```

```
which ddclient
```

This should return the path to `ddclient`, such as `/usr/local/bin/ddclient`.

2. Run `ddclient` manually to ensure it works:

```
bash
```

```
sudo /usr/local/bin/ddclient -daemon=0 -debug -verbose -noquiet
```

This will run `ddclient` in the foreground with debug output, which can help identify any issues with the configuration.

Step 4: Set Up `ddclient` as a LaunchDaemon

If `ddclient` runs correctly manually, proceed with setting it up as a LaunchDaemon:

- 1. Create a LaunchDaemon plist for `ddclient`:**

Create a file at `/Library/LaunchDaemons/org.ddclient.plist`.

```
bash
```

```
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

- 2. Add the following plist configuration:**

```
xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient/ddclient.conf</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

- 3. Save and exit:**

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

- 4. Ensure the correct file permissions:**

```
bash
```

```
sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist  
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist
```

5. Unload the `plist` (if previously loaded):

```
bash
```

```
sudo launchctl unload /Library/LaunchDaemons/org.ddclient.plist
```

6. Load the `plist` using `launchctl`:

```
bash
```

```
sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist
```

7. Check the status:

```
bash
```

```
sudo launchctl list | grep ddclient
```

Step 5: Verify Logs

1. Check the system logs for `ddclient` activity:

```
bash
```

```
sudo log show --predicate 'process == "ddclient"' --info --debug --last 1h
```

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

```
bash
```

```
node server.js
```

2. Access your server:

Open your browser and navigate to:

```
url  
  
http://sample-taplist.ddns.net:3000
```

By following these steps, you should be able to set up `ddclient` on macOS and ensure your DDNS provider is updated correctly. Let me know if you encounter any further issues or need further assistance!



what directory should my terminal be in for all this?



To avoid permission issues and to ensure smooth installation and configuration, you should run the commands from your home directory or any directory where you have the necessary permissions. Here's a step-by-step guide ensuring you are in the correct directory for each step.

Step-by-Step Guide

Step 1: Open Terminal and Navigate to Home Directory

1. Open Terminal:

You can open Terminal from your Applications folder or by searching for it in Spotlight.

2. Navigate to your home directory:

```
bash  
  
cd ~
```

Step 2: Install `ddclient` Using Homebrew

1. Install Homebrew (if you haven't already):

```
bash  
  
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

2. Install `ddclient`:

```
bash
brew install ddclient
```

Step 3: Create Necessary Directories

1. Create the configuration directory:

```
bash
sudo mkdir -p /usr/local/etc/ddclient
```

Step 4: Create and Edit the Configuration File

1. Create and edit the `ddclient` configuration file:

```
bash
sudo nano /usr/local/etc/ddclient/ddclient.conf
```

2. Add the following configuration:

Replace the placeholder values with your actual details. Ensure that you use the correct network interface (``en0``, ``en1``, etc.) that you identified earlier.

```
bash
daemon=300
pid=/var/run/ddclient.pid
ssl=yes
use;if=en0 # Replace 'en0' with your actual network interface
server=dyndns.no-ip.com
protocol=dyndns2
login=anderwolfe@gmail.com
password=ssjducks
sample-taplist.ddns.net
```

3. Save and exit:

Press `Ctrl+X` , then `Y` , and then `Enter` to save the file and exit the editor.

Step 5: Verify `ddclient` Installation

Ensure that `ddclient` is installed correctly.

1. Check `ddclient` installation:

```
bash
which ddclient
```

This should return the path to `ddclient`, such as `/usr/local/bin/ddclient`.

2. Run `ddclient` manually to ensure it works:

```
bash
sudo /usr/local/bin/ddclient -daemon=0 -debug -verbose -noquiet
```

This will run `ddclient` in the foreground with debug output, which can help identify any issues with the configuration.

Step 6: Set Up `ddclient` as a LaunchDaemon

1. Create a LaunchDaemon plist for `ddclient`:

Create a file at `/Library/LaunchDaemons/org.ddclient.plist`.

```
bash
sudo nano /Library/LaunchDaemons/org.ddclient.plist
```

2. Add the following plist configuration:

```
xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>org.ddclient</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/ddclient</string>
    <string>-file</string>
    <string>/usr/local/etc/ddclient/ddclient.conf</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

3. Save and exit:

Press `Ctrl+X`, then `Y`, and then `Enter` to save the file and exit the editor.

4. Ensure the correct file permissions:

```
bash

sudo chown root:wheel /Library/LaunchDaemons/org.ddclient.plist
sudo chmod 644 /Library/LaunchDaemons/org.ddclient.plist
```

5. Unload the `plist` (if previously loaded):

```
bash

sudo launchctl unload /Library/LaunchDaemons/org.ddclient.plist
```

6. Load the `plist` using `launchctl`:

```
bash

sudo launchctl load /Library/LaunchDaemons/org.ddclient.plist
```

7. Check the status:

```
bash

sudo launchctl list | grep ddclient
```

Step 7: Verify Logs

1. Check the system logs for `ddclient` activity:

```
bash

sudo log show --predicate 'process == "ddclient"' --info --debug --last 1h
```

Running Your Server

Once `ddclient` is correctly configured and running as a service, it will keep your No-IP hostname (`sample-taplist.ddns.net`) updated with your current public IP address.

1. Start your server:

Navigate to your project directory:

```
bash  
  
cd /path/to/your/project
```

Then start your server:

```
bash  
  
node server.js
```

2. Access your server:

Open your browser and navigate to:

```
url  
  
http://sample-taplist.ddns.net:3000
```

By following these steps, you should be able to set up `ddclient` on macOS and ensure your DDNS provider is updated correctly. Let me know if you encounter any further issues or need further assistance!



So no-ip is getting the updated ip info, but for some reason when I try to go to sample-taplist.ddns.net:3000 in a browser it goes nowhere. The browser will go to the tap list website though with localhost:3000



It sounds like the No-IP configuration is correctly updating your public IP address, but there might be a few network configuration issues that are preventing access to your server from the outside. Here are the steps to troubleshoot and fix this issue:

Step-by-Step Guide to Troubleshoot External Access

1. Verify Public IP Address:

Ensure that the public IP address provided by No-IP is correct and matches your current public IP address.

You can check your public IP address by visiting a website like WhatIsMyIP.com.

2. Check Port Forwarding:

Ensure that your router is configured to forward port 3000 to your internal IP address where your server is running.

- **Access your router's settings:** Typically, you can do this by entering the router's IP address in your web browser (e.g., `192.168.1.1` or `192.168.0.1`).
- **Login to the router:** Use your router's username and password.
- **Navigate to the port forwarding section:** This might be under a section called "Port Forwarding", "Virtual Server", or similar.
- **Add a new port forwarding rule:**
 - **Name:** Something descriptive like `Taplist`.
 - **External Port:** 3000
 - **Internal Port:** 3000
 - **Internal IP Address:** The internal IP address of your server (e.g., `192.168.1.100`).
 - **Protocol:** TCP

3. Check Firewall Settings:

Ensure that your server's firewall (if any) is configured to allow connections on port 3000.

- **macOS Firewall:**
 - Go to `System Preferences` > `Security & Privacy` > `Firewall`.
 - Click the `Firewall Options...` button.
 - Ensure that the application or port `3000` is allowed.
- **Firewall on the router:**
 - Ensure that the firewall on your router is not blocking port 3000.

4. Test External Access:

- After setting up port forwarding, visit `http://sample-taplist.ddns.net:3000` from an external network (not connected to the same local network as your server) to see if it works.
- You can also use a tool like [CanYouSeeMe.org](https://canyouseeme.org) to check if port 3000 is open from the internet.

Example Port Forwarding Configuration

Here is an example of what the port forwarding configuration might look like:

- **Name:** Taplist
- **External Port:** 3000
- **Internal Port:** 3000
- **Internal IP Address:** 192.168.1.100 (replace with your server's internal IP address)
- **Protocol:** TCP

Summary of Steps

1. Verify your public IP address.
2. Set up port forwarding on your router to forward port 3000 to your server's internal IP address.
3. Ensure firewall settings allow traffic on port 3000.
4. Test external access using `http://sample-taplist.ddns.net:3000`.

Additional Troubleshooting

If you still cannot access your server externally, here are a few additional troubleshooting steps:

- **Restart your router:** Sometimes, changes to the router's configuration require a restart.
- **Check ISP Restrictions:** Some ISPs block certain ports. If port 3000 is blocked, try using a different port.
- **Check Local Server Status:** Ensure your server is running and accessible on your local network using `localhost:3000` or your internal IP address.

Let me know if you need any further assistance or if any specific issues arise during these steps!



A note that no-ip is getting a 1918 address, which is intended because this website is only meant to be accessible from the LAN. Could that be part of why <https://www.sample-taplist.ddns.net:3000/> isn't working?



If No-IP is receiving and updating a 1918 address (a private IP address like `192.168.x.x`), it means that your server is meant to be accessed only from within your local network (LAN). External access using a public DNS name pointing to a private IP address won't work because private IP addresses are not routable on the public internet.

To access your server using `sample-taplist.ddns.net:3000` from within your LAN, you'll need to ensure that DNS resolution within your LAN points to the correct local IP address. Here are a few steps to ensure this works correctly:

Step-by-Step Guide to Access Your Server on LAN

1. Ensure Local DNS Resolution:

Ensure that devices on your LAN can resolve `sample-taplist.ddns.net` to your server's local IP address.

2. Check Local Network Configuration:

- Make sure the hostname (`sample-taplist.ddns.net`) resolves to your server's local IP address (`192.168.x.x`).

3. Hosts File Configuration (Optional):

If DNS resolution isn't working correctly on your LAN, you can manually configure the hosts file on your local machines to point `sample-taplist.ddns.net` to your server's local IP address.

- **On macOS/Linux:**

- Edit the `/etc/hosts` file:

```
bash
```

```
sudo nano /etc/hosts
```

- Add the following line:

```
bash
```

```
192.168.x.x sample-taplist.ddns.net
```

- Save and exit (`**Ctrl+X**` , `Y` , `Enter`).

- **On Windows:**

- Edit the `C:\Windows\System32\drivers\etc\hosts` file as an administrator.
- Add the following line:

```
plaintext
```

```
192.168.x.x sample-taplist.ddns.net
```

- Save the file.

4. Check Browser Access:

After updating the hosts file, try accessing `http://sample-taplist.ddns.net:3000` from a browser on a device within your LAN.

5. Ensure Firewall Configuration:

Ensure that the firewall on your server (macOS) allows connections on port 3000.

- **macOS Firewall:**

- Go to `System Preferences` > `Security & Privacy` > `Firewall`.
- Click the `Firewall Options...` button.
- Ensure that the application or port `3000` is allowed.

Example Configuration for Local Access

Assuming your server's local IP address is `192.168.1.100`, your `hosts` file entry would be:

```
plaintext
```

```
192.168.1.100 sample-taplist.ddns.net
```

Testing Local Access

1. From a Browser on the Same LAN:

Open your browser and go to:

```
url  
  
http://sample-taplist.ddns.net:3000
```

Conclusion

By ensuring that `sample-taplist.ddns.net` resolves to the correct local IP address and configuring your local devices' hosts files if necessary, you should be able to access your server using the hostname within your LAN.

If you still encounter issues or need further assistance, please let me know!



Is there a way programmatically, that we can make server.js print the 1918 address of the local machine somewhere on the internet that isn't github



Yes, you can programmatically obtain the 1918 (private) IP address of the local machine and then send it to a remote server or service that can store and display it. One way to achieve this is by using a simple web service like Pastebin, a Google Sheets script, or any other custom web server that you control.

Step-by-Step Guide to Obtain and Send the Private IP Address

Step 1: Obtain the Private IP Address in `server.js`

First, let's update your `server.js` to obtain the private IP address of the local machine.

server.js:

```
javascript  
  
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const fetch = require('node-fetch');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
```

```

const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {

```

```

        if (i.family === 'IPv4' && !i.internal) {
            localIp = i.address;
            break;
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to a remote service
async function sendIpToRemoteService() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    const remoteServiceUrl = 'https://your-remote-service.com/your-endpoint'; // Replace
    with your actual remote service URL
    const payload = {
        ip: localIp,
        timestamp: new Date().toISOString()
    };
    try {
        const response = await fetch(remoteServiceUrl, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(payload)
        });
        if (!response.ok) {
            throw new Error(`Error: ${response.statusText}`);
        }
        const responseData = await response.json();
        console.log('IP address sent successfully:', responseData);
    } catch (error) {
        console.error('Failed to send IP address:', error);
    }
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToRemoteService(); // Send IP address at startup
    setInterval(sendIpToRemoteService, 3600000); // Send IP address every hour
});

```

Step 2: Set Up a Remote Service to Receive the IP Address

For demonstration purposes, let's use Pastebin as the remote service. You can create a free account on Pastebin and get an API key.

1. Create a Pastebin account and get an API key:

- o Sign up at <https://pastebin.com/signup>.
- o Once logged in, go to <https://pastebin.com/api> and get your API key.

2. Modify the `sendIpToRemoteService` function to send the IP address to Pastebin:

Replace the `sendIpToRemoteService` function in `server.js` with the following code:

javascript

```

async function sendIpToPastebin() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const pastebinApiKey = 'your_pastebin_api_key'; // Replace with your Pastebin API
    key
    const payload = {
        api_dev_key: pastebinApiKey,
        api_option: 'paste',
        api_paste_code: `Local IP: ${localIp}\nTimestamp: ${new Date().toISOString()}`,
        api_paste_private: '1',
        api_paste_name: 'Local IP Address',
        api_paste_expire_date: '10M'
    };

    try {
        const response = await fetch('https://pastebin.com/api/api_post.php', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/x-www-form-urlencoded'
            },
            body: new URLSearchParams(payload)
        });

        if (!response.ok) {
            throw new Error(`Error: ${response.statusText}`);
        }

        const responseData = await response.text();
        console.log('IP address sent to Pastebin successfully:', responseData);
    } catch (error) {
        console.error('Failed to send IP address to Pastebin:', error);
    }
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToPastebin(); // Send IP address at startup
    setInterval(sendIpToPastebin, 3600000); // Send IP address every hour
});

```

Replace `your_pastebin_api_key` with your actual Pastebin API key.

Running Your Server

1. Start your server:

bash

```
node server.js
```

2. Check Pastebin:

After starting your server, check Pastebin to see if the IP address has been posted. You should see a new paste with the local IP address and timestamp.

This setup will ensure that your local IP address is sent to Pastebin (or any other remote service) and can be accessed from there. If you prefer another service or have your own server, you can modify the `sendIpToRemoteService` function accordingly. Let me know if you need further assistance!



Instead of pastebin could it send it to a google sheets page?



Yes, you can send the local IP address to a Google Sheets page using the Google Sheets API. Below is a step-by-step guide on how to set this up.

Step-by-Step Guide to Send Local IP Address to Google Sheets

Step 1: Set Up Google Sheets API

1. Create a new Google Cloud project:

- Go to the [Google Cloud Console](#).
- Click on the project dropdown and select "New Project".
- Give your project a name and click "Create".

2. Enable Google Sheets API:

- In the Google Cloud Console, go to the "APIs & Services" > "Library".
- Search for "Google Sheets API" and click on it.
- Click "Enable" to enable the API for your project.

3. Create service account credentials:

- Go to the "APIs & Services" > "Credentials".
- Click "Create Credentials" and select "Service Account".
- Fill in the details and click "Create".
- Grant the service account "Editor" role.
- Click "Continue" and then "Done".
- In the "Credentials" tab, click on your service account email.
- Click "Add Key" > "Create New Key" and select "JSON". This will download a JSON file with your service account credentials.

4. Share your Google Sheet with the service account:

- Create a new Google Sheet.
- Share the Google Sheet with the service account email (found in the JSON file).

Step 2: Install Node.js Dependencies

1. Navigate to your project directory:

```
bash
cd /path/to/your/project
```

2. Install necessary packages:

```
bash
npm install googleapis
```

Step 3: Update `server.js` to Send IP to Google Sheets

1. Replace the `sendIpToRemoteService` function with the following code:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const { authenticate } = require('@google-cloud/local-auth');
const credentials = require('./path/to/your/credentials.json'); // Replace with the
path to your downloaded JSON credentials file

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure
password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const IP_FILE_PATH = path.join(__dirname, 'public', 'ip.json');
const SPREADSHEET_ID = 'your_spreadsheet_id'; // Replace with your Google Sheets ID
const SHEET_NAME = 'Sheet1'; // Replace with your sheet name

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};
```

```

// Serve static files from public directory, but protect admin.html and color-
admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Sheets
async function sendIpToGoogleSheets() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
            }
        }
    }
}

```

```

        break;
    }
}

const auth = new google.auth.GoogleAuth({
    keyFile: './path/to/your/credentials.json', // Replace with the path to your
downloaded JSON credentials file
    scopes: ['https://www.googleapis.com/auth/spreadsheets']
});

const sheets = google.sheets({ version: 'v4', auth });
const now = new Date();
const values = [
    [
        localIp,
        now.toISOString()
    ]
];

const resource = {
    values,
};

try {
    const response = await sheets.spreadsheets.values.append({
        spreadsheetId: SPREADSHEET_ID,
        range: `${SHEET_NAME}!A1`,
        valueInputOption: 'RAW',
        resource,
    });
    console.log('IP address sent to Google Sheets successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Sheets:', error);
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleSheets(); // Send IP address at startup
    setInterval(sendIpToGoogleSheets, 3600000); // Send IP address every hour
});

```

Replace:

- `./path/to/your/credentials.json` with the actual path to your downloaded JSON credentials file.
- `your_spreadsheet_id` with your actual Google Sheets ID.
- `Sheet1` with the name of the sheet where you want to store the IP addresses.

Running Your Server

1. Start your server:

```

bash

node server.js

```

2. Check Google Sheets:

After starting your server, check your Google Sheets to see if the IP address and timestamp have been added.

By following these steps, you should be able to send the local IP address of your server to a Google Sheets page programmatically. Let me know if you encounter any further issues or need further assistance!



Could it be a google docs page instead of a google sheets page?



Yes, you can send the local IP address to a Google Docs page instead of a Google Sheets page. To achieve this, you need to use the Google Docs API. Below is a step-by-step guide on how to set this up.

Step-by-Step Guide to Send Local IP Address to Google Docs

Step 1: Set Up Google Docs API

1. Create a new Google Cloud project:

- Go to the [Google Cloud Console](#).
- Click on the project dropdown and select "New Project".
- Give your project a name and click "Create".

2. Enable Google Docs API:

- In the Google Cloud Console, go to the "APIs & Services" > "Library".
- Search for "Google Docs API" and click on it.
- Click "Enable" to enable the API for your project.

3. Create service account credentials:

- Go to the "APIs & Services" > "Credentials".
- Click "Create Credentials" and select "Service Account".
- Fill in the details and click "Create".
- Grant the service account "Editor" role.
- Click "Continue" and then "Done".
- In the "Credentials" tab, click on your service account email.
- Click "Add Key" > "Create New Key" and select "JSON". This will download a JSON file with your service account credentials.

4. Share your Google Doc with the service account:

- Create a new Google Doc.
- Share the Google Doc with the service account email (found in the JSON file).

Step 2: Install Node.js Dependencies

1. Navigate to your project directory:

```
bash
```

```
cd /path/to/your/project
```

2. Install necessary packages:

```
bash
```

```
npm install googleapis
```

Step 3: Update `server.js` to Send IP to Google Docs

1. Replace the `sendIpToRemoteService` function with the following code:

server.js:

```
javascript
```

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./path/to/your/credentials.json'); // Replace with the
path to your downloaded JSON credentials file

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure
password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = 'your_doc_id'; // Replace with your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-
admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});
```

```

});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: './path/to/your/credentials.json', // Replace with the path to your
downloaded JSON credentials file
    scopes: ['https://www.googleapis.com/auth/documents']
}

```

```

});;

const docs = google.docs({ version: 'v1', auth });
const now = new Date();
const content = `Local IP: ${localIp}\nTimestamp: ${now.toISOString()}\n\n`;

try {
    const response = await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: {
            requests: [
                {
                    insertText: {
                        location: {
                            index: 1,
                        },
                        text: content,
                    },
                },
            ],
        },
    });
    console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Replace:

- `./path/to/your/credentials.json` with the actual path to your downloaded JSON credentials file.
- `your_doc_id` with your actual Google Docs ID.

Running Your Server

1. Start your server:

```

bash

node server.js

```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the IP address and timestamp have been added.

By following these steps, you should be able to send the local IP address of your server to a Google Docs page programmatically. Let me know if you encounter any further issues or need further assistance!



the downloaded json credentials file will live in the same folder are server.js



Great! If the downloaded JSON credentials file is in the same folder as `server.js`, you can reference it directly. Here's the updated `server.js` script, assuming the JSON credentials file is named `credentials.json` and is located in the same directory as `server.js`.

Updated `server.js` to Send IP to Google Docs

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Replace with your actual credentials file

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = 'your_doc_id'; // Replace with your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
```

```

if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
} else {
    res.send('Incorrect password');
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const content = `Local IP: ${localIp}\nTimestamp: ${now.toISOString()}\n\n`;
}

```

```

try {
  const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
      requests: [
        {
          insertText: {
            location: {
              index: 1,
            },
            text: content,
          },
        },
      ],
    },
  });
  console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
  console.error('Failed to send IP address to Google Docs:', error);
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Replace:

- `./credentials.json` with the actual path to your downloaded JSON credentials file.
- `your_doc_id` with your actual Google Docs ID.

Running Your Server

1. Start your server:

```

bash

node server.js

```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the IP address and timestamp have been added.

By following these steps, you should be able to send the local IP address of your server to a Google Docs page programmatically. Let me know if you encounter any further issues or need further assistance!



I got this message when running server.js

Failed to send IP address to Google Docs: Error: Requested entity was not found.
at Gaxios._request (/Users/duncananderson/Desktop/Taplist-Project/node_modules/gaxios/build/src/gaxios.js:142:23)

```
at process.processTicksAndRejections (/Users/duncananderson/Desktop/Taplist-Project/lib/internal/process/task_queues.js:95:5)
  at async JWT.requestAsync (/Users/duncananderson/Desktop/Taplist-Project/node_modules/google-auth-library/build/src/auth/oauth2client.js:408:18)
  at async sendIpToGoogleDocs (/Users/duncananderson/Desktop/Taplist-Project/server.js:122:26)
{config: {...}, response: {...}, error: undefined, status: 404, code: 404, ...}
```



The error message "Requested entity was not found" typically means that the Google Docs document ID provided is incorrect or the service account does not have the necessary permissions to access the document. Let's troubleshoot and ensure everything is set up correctly.

Step-by-Step Guide to Troubleshoot the Issue

Step 1: Verify the Document ID

1. **Open the Google Docs document in your browser.**
2. **Check the URL:** The document ID is the long string in the URL. For example, in the URL `https://docs.google.com/document/d/1A2B3C4D5E6F7G8H9I0J/edit`, the document ID is `1A2B3C4D5E6F7G8H9I0J`.
3. **Ensure the document ID in your code matches the one in the URL:**

```
javascript
```

```
const DOC_ID = 'your_correct_doc_id'; // Replace with your actual Google Doc ID
```

Step 2: Verify Permissions

1. **Share the Google Doc with the service account:**
 - o Open the Google Docs document.
 - o Click the "Share" button.
 - o Share the document with the service account email (found in the JSON credentials file).

Step 3: Ensure Correct Path to Credentials File

1. **Ensure the path to the JSON credentials file is correct:**

```
javascript
```

```
const auth = new google.auth.GoogleAuth({
  keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
  file path
  scopes: ['https://www.googleapis.com/auth/documents']
});
```

Step 4: Update the Code and Retry

Here is the updated `server.js` script for clarity:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = 'your_correct_doc_id'; // Replace with your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 'your_secret_key', // Replace with a secure key
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});
```

```

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    file path
    scopes: ['https://www.googleapis.com/auth/documents']
  });

  const docs = google.docs({ version: 'v1', auth });
  const now = new Date();
  const content = `Local IP: ${localIp}\nTimestamp: ${now.toISOString()}\n\n`;

  try {
    const response = await docs.documents.batchUpdate({
      documentId: DOC_ID,
      requestBody: {
        requests: [
          {
            insertText: {
              location: {
                index: 1,
              },
              text: content,
            },
          },
        ],
      },
    });
  }
}

```

```

        console.log('IP address sent to Google Docs successfully:', response.data);
    } catch (error) {
        console.error('Failed to send IP address to Google Docs:', error);
    }
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Step 5: Running Your Server

1. Start your server:

bash

node server.js

2. Check Google Docs:

After starting your server, check your Google Docs to see if the IP address and timestamp have been added.

Summary of Steps

1. Verify that the document ID in the code matches the one in the Google Docs URL.
2. Ensure that the Google Docs document is shared with the service account.
3. Ensure that the path to the JSON credentials file is correct.
4. Start your server and check for updates in Google Docs.

By following these steps, you should be able to resolve the "Requested entity was not found" error and successfully send the IP address to Google Docs. Let me know if you encounter any further issues or need further assistance!



this is the doc ID: 1T1HqTNp-VH3ZTIEbi4O7NUJltII9LNGZc8yXNz6v3NQ

Also what is your_secret_key supposed to be replace with in server.js?



In `server.js`, `your_secret_key` should be replaced with a secure string that is used to sign the session ID cookie. This string should be kept secret and should be difficult to guess to enhance security. You can generate a secure random string using various online tools or libraries.

For example, you can replace `your_secret_key` with something like `s3cur3R@nd0mStr1ng`. Here's the updated code:

Updated `server.js`

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's3cur3R@nd0mStr1ng', // Replace with a secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
```

```

fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
        res.status(500).send('Error reading beers file');
    } else {
        res.json(JSON.parse(data));
    }
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const content = `Local IP: ${localIp}\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                            text: content,
                        },
                    },
                ],
            },
        });
        console.log('IP address sent to Google Docs successfully:', response.data);
    }
}

```

```

    } catch (error) {
      console.error('Failed to send IP address to Google Docs:', error);
    }
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Summary

1. Replace `your_secret_key` with a secure random string.
2. Replace `your_secure_password` with your actual secure password.
3. Replace `your_doc_id` with your Google Doc ID.
4. Ensure that the credentials file is correctly referenced.

Running Your Server

1. Start your server:

```

bash

node server.js

```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the IP address and timestamp have been added.

By following these steps, you should be able to send the local IP address of your server to a Google Docs page programmatically. Let me know if you encounter any further issues or need further assistance!



Would you please generate an actual secure random string to replace that place holder?



Sure! Here's a secure random string that you can use to replace the placeholder:

```

plaintext

s2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ

```

This string is a combination of letters, numbers, and special characters, making it difficult to guess.

Updated `server.js`

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {

```

```

        res.status(500).send('Error reading beers file');
    } else {
        res.json(JSON.parse(data));
    }
});
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const content = `Local IP: ${localIp}\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                            text: content,
                        },
                    },
                ],
            },
        });
        console.log('IP address sent to Google Docs successfully:', response.data);
    } catch (error) {
        console.error('Failed to send IP address to Google Docs:', error);
    }
}

```

```

    }
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running Your Server

1. Start your server:

```

bash
node server.js

```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the IP address and timestamp have been added.

By following these steps, you should be able to send the local IP address of your server to a Google Docs page programmatically. Let me know if you encounter any further issues or need further assistance!



Can we make it so when it writes in the google doc, it writes a hyperlink with the words, "Click to edit taplist" and it links to the sent local IP address plus port 3000 on the end of said IP



Sure! You can modify the `sendIpToGoogleDocs` function to insert a hyperlink instead of plain text. Below is the updated code to achieve this.

Updated `server.js`

server.js:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();

```

```

const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
  if (req.path === '/admin.html' || req.path === '/color-admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';

```

```

for (const interfaceName in networkInterfaces) {
  const interfaces = networkInterfaces[interfaceName];
  for (const i of interfaces) {
    if (i.family === 'IPv4' && !i.internal) {
      localIp = i.address;
      break;
    }
  }
  res.json({ ip: localIp });
};

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    file path
    scopes: ['https://www.googleapis.com/auth/documents']
  });

  const docs = google.docs({ version: 'v1', auth });
  const now = new Date();
  const url = `http://${localIp}:3000`;
  const hyperlink = {
    textRun: {
      content: 'Click to edit taplist\n',
      textStyle: {
        link: {
          url: url
        }
      }
    }
  };
}

try {
  const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
      requests: [
        {
          insertText: {
            location: {
              index: 1,
            },
            text: `Timestamp: ${now.toISOString()}\n`
          }
        },
        {
          insertText: {
            location: {
              index: 1,
            },
            text: hyperlink.textRun.content,
            textStyle: hyperlink.textRun.textStyle
          }
        }
      ],
    }
  });
  console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
  console.error('Failed to send IP address to Google Docs:', error);
}

```

```

    }

    app.listen(PORT, '0.0.0.0', () => {
      console.log(`Server is running on http://localhost:${PORT}`);
      sendIpToGoogleDocs(); // Send IP address at startup
      setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
    });
  
```

Running Your Server

1. Start your server:

```

bash

node server.js
  
```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the hyperlink "Click to edit taplist" and the timestamp have been added.

This updated script ensures that the Google Doc will have a hyperlink pointing to the local IP address with port 3000. Let me know if you encounter any further issues or need further assistance!



I got this message when trying to run server.js

```

Failed to send IP address to Google Docs: Error: Invalid JSON payload received. Unknown name
"textStyle" at 'requests[1].insert_text': Cannot find field.
  at Gaxios._request (/Users/duncananderson/Desktop/Taplist-
Project/node_modules/gaxios/build/src/gaxios.js:142:23)
  at process.processTicksAndRejections (/Users/duncananderson/Desktop/Taplist-
Project/lib/internal/process/task_queues.js:95:5)
  at async JWT.requestAsync (/Users/duncananderson/Desktop/Taplist-
Project/node_modules/google-auth-library/build/src/auth/oauth2client.js:408:18)
  at async sendIpToGoogleDocs (/Users/duncananderson/Desktop/Taplist-Project/server.js:132:26)
{config: {...}, response: {...}, error: undefined, status: 400, code: 400, ...}
  
```



The error message indicates that there is an issue with the JSON payload for the `batchUpdate` request, specifically with the `insertText` request not recognizing the `textStyle` field. To insert a hyperlink in a Google Doc using the API, you need to use the `insertText` request to insert the text and then a `updateTextStyle` request to apply the hyperlink style.

Here is the corrected version of the `sendIpToGoogleDocs` function:

Updated `sendIpToGoogleDocs` Function

```

javascript

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const url = `http://${localIp}:3000`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                            text: textToInsert,
                        },
                    },
                    {
                        updateTextStyle: {
                            range: {
                                startIndex: 1,
                                endIndex: 18, // Adjust this range to match the length of
                                "Click to edit taplist"
                            },
                            textStyle: {
                                link: {
                                    url: url,
                                },
                            },
                            fields: 'link',
                        },
                    },
                ],
            },
        });
        console.log('IP address sent to Google Docs successfully:', response.data);
    } catch (error) {
        console.error('Failed to send IP address to Google Docs:', error);
    }
}

```

Full Updated `server.js`

Here is the full updated `server.js` file including the corrected `sendIpToGoogleDocs` function:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {

```

```

        res.status(500).send('Error reading beers file');
    } else {
        res.json(JSON.parse(data));
    }
});
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const url = `http://${localIp}:3000`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                            text: textToInsert,
                        },
                    },
                    {
                        updateTextStyle: {
                            range: {
                                startIndex: 1,
                                endIndex: 18, // Adjust this range to match the length of
                            }
                        }
                    }
                ]
            }
        });
    }
}

```

```

    "Click to edit taplist"
        },
        textStyle: {
            link: {
                url: url,
            },
            fields: 'link',
        },
    ],
},
}),
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running Your Server

1. Start your server:

```

bash

node server.js

```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the hyperlink "Click to edit taplist" and the timestamp have been added.

This updated script should ensure that the Google Doc will have a hyperlink pointing to the local IP address with port 3000. Let me know if you encounter any further issues or need further assistance!



can we make the Click to edit taplist font size twice as a big and link to the local ip address, with port 3000, plus /admin.html

Can we also make it so the google doc overwrites it self instead of adding new text every time the check is called to write new text



Sure! Let's make the necessary adjustments to the `sendIpToGoogleDocs` function to format the text, include the `/admin.html` path, and overwrite the existing content in the Google Doc.

Updated `sendIpToGoogleDocs` Function

Here is the updated function to achieve your requirements:

1. **Change the hyperlink to include `/admin.html`.
2. **Adjust the font size for the "Click to edit taplist" text.
3. **Overwrite the existing content instead of appending new text.**

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});
```

```

});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    file path
    scopes: ['https://www.googleapis.com/auth/documents']
  });

  const docs = google.docs({ version: 'v1', auth });
  const now = new Date();
  const url = `http://${localIp}:3000/admin.html`;
  const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

  try {
    // Clear existing content
    await docs.documents.batchUpdate({
      documentId: DOC_ID,
      requestBody: {
        requests: [
          {
            deleteContentRange: {
              range: {

```

```

        startIndex: 1,
        endIndex: 1000000, // Large number to ensure all content is
      deleted
      },
    },
  ],
},
});

// Insert new content
const response = await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        insertText: {
          location: {
            index: 1,
          },
          text: textToInsert,
        },
      },
      {
        updateTextStyle: {
          range: {
            startIndex: 1,
            endIndex: 21, // Adjust this range to match the length of
            "Click to edit taplist"
          },
          textStyle: {
            link: {
              url: url,
            },
            fontSize: {
              magnitude: 16,
              unit: 'PT',
            },
            fields: 'link,fontSize',
          },
        },
      },
      ],
    },
  });
  console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
  console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running Your Server

1. Start your server:

bash

```
node server.js
```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the hyperlink "Click to edit taplist" (with the specified font size) and the timestamp have been added. The content should overwrite itself each time the check is called.

This updated script ensures that the Google Doc will have a hyperlink pointing to the local IP address with port 3000 and `admin.html`, with the specified font size, and will overwrite the existing content instead of appending new text each time. Let me know if you encounter any further issues or need further assistance!



I got this message when trying to run server.js

Failed to send IP address to Google Docs: Error: Invalid requests[0].deleteContentRange: Index 999999 must be less than the end index of the referenced segment, 122.

```
at Gaxios._request (/Users/duncananderson/Desktop/Taplist-Project/node_modules/gaxios/build/src/gaxios.js:142:23)
  at process.processTicksAndRejections (/Users/duncananderson/Desktop/Taplist-Project/lib/internal/process/task_queues.js:95:5)
    at async JWT.requestAsync (/Users/duncananderson/Desktop/Taplist-Project/node_modules/google-auth-library/build/src/auth/oauth2client.js:408:18)
      at async sendIpToGoogleDocs (/Users/duncananderson/Desktop/Taplist-Project/server.js:124:9)
{config: {...}, response: {...}, error: undefined, status: 400, code: 400, ...}
```



The error message indicates that the range specified for `deleteContentRange` is out of bounds. To fix this, we need to first determine the actual end index of the document content and then clear the document within the valid range. Here's how you can achieve this:

Updated `sendIpToGoogleDocs` Function

1. **Determine the end index of the document.**
2. **Clear the document content within the valid range.**
3. **Insert the new content.**

server.js:

javascript

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
```

```

const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        ...
    }
});

```

```

        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,
                                endIndex: endIndex,
                            },
                        },
                    },
                    ],
                },
            });
        }

        // Insert new content
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                            text: textToInsert,
                        },
                    },
                    {
                        updateTextStyle: {
                            range: {

```

```

        startIndex: 1,
        endIndex: 21, // Adjust this range to match the length of
    "Click to edit taplist"
    },
    textStyle: {
        link: {
            url: url,
        },
        fontSize: {
            magnitude: 16,
            unit: 'PT',
        },
        fields: 'link,fontSize',
    },
},
],
},
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Explanation

- Get Document End Index:** Retrieve the document to determine its actual end index.
- Delete Existing Content:** Use the valid range based on the document's end index to delete the content.
- Insert New Content:** Insert the new text and apply the hyperlink and font size.

Running Your Server

1. Start your server:

```

bash

node server.js

```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the hyperlink "Click to edit taplist" (with the specified font size) and the timestamp have been added. The content should overwrite itself each time the check is called.

This updated script should resolve the range issue and ensure the Google Doc content is overwritten properly. Let me know if you encounter any further issues or need further assistance!



I got this message when trying to run server.js

```
Failed to send IP address to Google Docs: Error: Invalid requests[0].deleteContentRange: The range
cannot include the newline character at the end of the segment.
  at Gaxios._request (/Users/duncananderson/Desktop/Taplist-
Project/node_modules/gaxios/build/src/gaxios.js:142:23)
  at process.processTicksAndRejections (/Users/duncananderson/Desktop/Taplist-
Project/lib/internal/process/task_queues.js:95:5)
  at async JWT.requestAsync (/Users/duncananderson/Desktop/Taplist-
Project/node_modules/google-auth-library/build/src/auth/oauth2client.js:408:18)
  at async sendIpToGoogleDocs (/Users/duncananderson/Desktop/Taplist-Project/server.js:130:9)
{config: {...}, response: {...}, error: undefined, status: 400, code: 400, ...}
```



The error message indicates that the ``deleteContentRange`` request cannot include the newline character at the end of the segment. To avoid this issue, we'll adjust the end index of the range to exclude the newline character. Here's the updated ``sendIpToGoogleDocs`` function to handle this:

Updated `sendIpToGoogleDocs` Function

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedin) {
    next();
  } else {
    res.redirect('/login.html');
  }
}
```

```

    }

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
}

```

```

        localIp = i.address;
        break;
    }
}

const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    file path
    scopes: ['https://www.googleapis.com/auth/documents']
});

const docs = google.docs({ version: 'v1', auth });
const now = new Date();
const url = `http://${localIp}:3000/admin.html`;
const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
    const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
        return element.endIndex > maxIndex ? element.endIndex : maxIndex;
    }, 1);

    // Adjust the endIndex to exclude the newline character
    const adjustedEndIndex = endIndex - 1;

    // Clear existing content
    await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: {
            requests: [
                {
                    deleteContentRange: {
                        range: {
                            startIndex: 1,
                            endIndex: adjustedEndIndex,
                        },
                    },
                },
            ],
        },
    });
}

// Insert new content
const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
        requests: [
            {
                insertText: {
                    location: {
                        index: 1,
                    },
                    text: textToInsert,
                },
            },
            {
                updateTextStyle: {
                    range: {
                        startIndex: 1,
                        endIndex: 21, // Adjust this range to match the length of
                        "Click to edit taplist"
                    },
                    textStyle: {
                        link: {
                            url: url,
                        },
                        fontSize: {
                            magnitude: 16,
                            unit: 'PT',
                        },
                    },
                    fields: 'link,fontSize',
                },
            },
        ],
    },
});

```

```

        },
      ],
    });
    console.log('IP address sent to Google Docs successfully:', response.data);
  } catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
  }
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Explanation

- Determine the End Index:** Retrieve the document to determine its actual end index.
- Adjust End Index:** Subtract 1 from the end index to exclude the newline character.
- Delete Existing Content:** Use the valid range based on the adjusted end index to delete the content.
- Insert New Content:** Insert the new text and apply the hyperlink and font size.

Running Your Server

1. Start your server:

```

bash
node server.js

```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the hyperlink "Click to edit taplist" (with the specified font size) and the timestamp have been added. The content should overwrite itself each time the check is called.

This updated script should resolve the range issue and ensure the Google Doc content is overwritten properly. Let me know if you encounter any further issues or need further assistance!



Works great except the last letter in "Click to edit taplist" is not a part of the hyperlink for some reason



The issue with the last letter not being part of the hyperlink is likely due to the range used in the `updateTextStyle` request. To ensure the entire phrase "Click to edit taplist" is hyperlinked, let's adjust the range to cover the entire text.

Updated `sendIpToGoogleDocs` Function

Here's the adjusted function to ensure the full text "Click to edit taplist" is included in the hyperlink:

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});
```

```

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    file path
    scopes: ['https://www.googleapis.com/auth/documents']
  });

  const docs = google.docs({ version: 'v1', auth });
  const now = new Date();
  const url = `http://${localIp}:3000/admin.html`;
  const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

  try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
    const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
      return element.endIndex > maxIndex ? element.endIndex : maxIndex;
    }, 1);

    // Adjust the endIndex to exclude the newline character
    const adjustedEndIndex = endIndex - 1;

    // Clear existing content
    await docs.documents.batchUpdate({
      documentId: DOC_ID,
      requestBody: {
        requests: [

```

```

        {
            deleteContentRange: {
                range: {
                    startIndex: 1,
                    endIndex: adjustedEndIndex,
                },
            },
        ],
    },
});

// Insert new content
const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: [
        requests: [
            {
                insertText: {
                    location: {
                        index: 1,
                    },
                    text: textToInsert,
                },
            },
            {
                updateTextStyle: {
                    range: {
                        startIndex: 1,
                        endIndex: 21, // Adjust this range to match the length of
"Click to edit taplist"
                    },
                    textStyle: {
                        link: {
                            url: url,
                        },
                        fontSize: {
                            magnitude: 32, // Twice the original font size (16 * 2)
                            unit: 'PT',
                        },
                    },
                    fields: 'link,fontSize',
                },
            },
        ],
    },
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Explanation

- Determine the End Index:** Retrieve the document to determine its actual end index.
- Adjust End Index:** Subtract 1 from the end index to exclude the newline character.
- Delete Existing Content:** Use the valid range based on the adjusted end index to delete the content.
- Insert New Content:** Insert the new text and apply the hyperlink and font size. Note that the `endIndex` for the `updateTextStyle` request has been set to 21 to cover the full text "Click to edit taplist".

Running Your Server

1. Start your server:

```
bash
node server.js
```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the hyperlink "Click to edit taplist" (with the specified font size) and the timestamp have been added. The content should overwrite itself each time the check is called.

This updated script should ensure that the entire text "Click to edit taplist" is hyperlinked and that the content is overwritten properly each time. Let me know if you encounter any further issues or need further assistance!



the hyperlink font size is much bigger but the t at the end is still not a part of the hyperlink, it is just plain text at the moment. I'd like it to be part of the hyperlink please



It looks like the ``endIndex`` might not be correctly covering the entire length of the text "Click to edit taplist". Let's adjust the ``endIndex`` to ensure it includes the full text. The length of "Click to edit taplist" is 21 characters, so the ``endIndex`` should be 22 (since indices are zero-based).

Updated `sendIpToGoogleDocs` Function

`server.js`:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
```

```

app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
});

```

```

        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
        const adjustedEndIndex = endIndex - 1;

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,
                                endIndex: adjustedEndIndex,
                            },
                        },
                    },
                ],
            },
        });

        // Insert new content
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                            text: textToInsert,
                        },
                    },
                    {
                        updateTextStyle: {
                            range: {
                                startIndex: 1,
                                endIndex: 22, // Adjust this range to match the length of
                            }
                        }
                    }
                ]
            }
        });
    }
}

```

```

    "Click to edit taplist" (21 characters + 1 to cover the full length)
        },
        textStyle: {
            link: {
                url: url,
            },
            fontSize: {
                magnitude: 32, // Twice the original font size (16 * 2)
                unit: 'PT',
            },
            fields: 'link,fontSize',
        },
    },
],
},
);
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running Your Server

1. Start your server:

```

bash
node server.js

```

2. Check Google Docs:

After starting your server, check your Google Docs to see if the hyperlink "Click to edit taplist" (with the specified font size) and the timestamp have been added. The content should overwrite itself each time the check is called.

This updated script should ensure that the entire text "Click to edit taplist" is hyperlinked and that the content is overwritten properly each time. Let me know if you encounter any further issues or need further assistance!



Wonderful. Now I've noticed that admin.html is not behaving correctly. When you use the edit button to edit beers in the tap list, it gives you a "failed to save beers" message and the editable text fields persist even after clicking save



Let's address the issue with `admin.html` not saving the beers correctly. We need to ensure the backend route handling the save operation is correctly implemented and the front-end is correctly interacting with it.

Backend Changes

Ensure that you have an endpoint to handle the saving of beers. This should be a POST endpoint that receives the updated list of beers and saves it to the JSON file.

server.js:

```
javascript

app.post('/save-beers', isAuthenticated, (req, res) => {
  const updatedBeers = req.body;
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save beers');
    } else {
      res.send('Beers saved successfully');
    }
  });
});
```

Frontend Changes

Ensure that your `admin.html` page correctly sends the updated beer list to the backend endpoint.

admin.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Admin - Edit Tap List</title>
  <style>
    .editable {
      border: 1px solid #000;
      padding: 5px;
    }
  </style>
</head>
<body>
  <h1>Edit Tap List</h1>
  <button id="editButton">Edit</button>
  <button id="saveButton" style="display: none;">Save</button>
  <div id="tapList">
    <!-- Beer list will be populated here -->
  </div>
  <script>
    document.addEventListener('DOMContentLoaded', () => {
      const tapListDiv = document.getElementById('tapList');
      const editButton = document.getElementById('editButton');
      const saveButton = document.getElementById('saveButton');

      fetch('/beers')
        .then(response => response.json())
```

```

        .then(beers => {
            beers.forEach((beer, index) => {
                const beerDiv = document.createElement('div');
                beerDiv.textContent = `${index + 1}. ${beer.name} by ${beer.brewery} - ${beer.abv}`;
                tapListDiv.appendChild(beerDiv);
            });
        });

        editButton.addEventListener('click', () => {
            const beerDivs = tapListDiv.querySelectorAll('div');
            beerDivs.forEach(div => {
                const index = div.dataset.index;
                const beerText = div.textContent;
                const [name, rest] = beerText.split(' by ');
                const [brewery, abv] = rest.split(' - ');

                div.innerHTML =
                    `` +
                    `` +
                    ``;
                div.classList.add('editable');
            });
            editButton.style.display = 'none';
            saveButton.style.display = 'block';
        });

        saveButton.addEventListener('click', () => {
            const updatedBeers = [];
            const beerDivs = tapListDiv.querySelectorAll('div');
            beerDivs.forEach(div => {
                const inputs = div.querySelectorAll('input');
                const beer = {};
                inputs.forEach(input => {
                    const field = input.dataset.field;
                    beer[field] = input.value;
                });
                updatedBeers.push(beer);
            });

            fetch('/save-beers', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(updatedBeers)
            })
            .then(response => response.text())
            .then(result => {
                if (result === 'Beers saved successfully') {
                    beerDivs.forEach((div, index) => {
                        const beer = updatedBeers[index];
                        div.innerHTML = `${index + 1}. ${beer.name} by ${beer.brewery} - ${beer.abv}`;
                        div.classList.remove('editable');
                    });
                    editButton.style.display = 'block';
                    saveButton.style.display = 'none';
                } else {
                    alert(result);
                }
            })
            .catch(error => {
                alert('Failed to save beers');
                console.error(error);
            });
        });
    });

```

Explanation

1. Backend:

- The `/save-beers` endpoint is added to handle saving the beers. It writes the updated beer list to `beers.json`.

2. Frontend:

- The `admin.html` page fetches the current list of beers and populates the `tapListDiv`.
- When the "Edit" button is clicked, each beer is replaced with editable input fields.
- When the "Save" button is clicked, the updated beer data is sent to the `/save-beers` endpoint.
- If the save operation is successful, the page updates the displayed beer list and removes the editable fields.

This should resolve the issues with the edit and save functionality on the `admin.html` page. Let me know if you encounter any further issues or need further assistance!



Would you please show me what the code for server.js should look like in its entirety



Sure! Below is the complete `server.js` file, incorporating all the necessary endpoints and functionalities to support editing the tap list from `admin.html`.

Complete `server.js`

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
  resave: false,
  saveUninitialized: true
}));
```

```

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {

```

```

        if (i.family === 'IPv4' && !i.internal) {
            localIp = i.address;
            break;
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
        const adjustedEndIndex = endIndex - 1;

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,
                                endIndex: adjustedEndIndex,
                            },
                        },
                    ],
                ],
            },
        });

        // Insert new content
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                            text: textToInsert,
                        },
                    },
                ],
            },
        });
    }
}

```

```

        updateTextStyle: {
            range: {
                startIndex: 1,
                endIndex: 22, // Adjust this range to match the length of
                "Click to edit taplist" (21 characters + 1 to cover the full length)
            },
            textStyle: {
                link: {
                    url: url,
                },
                fontSize: {
                    magnitude: 32, // Twice the original font size (16 * 2)
                    unit: 'PT',
                },
                fields: 'link,fontSize',
            },
        },
    ],
},
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Explanation

- Session Management:** The server uses `express-session` to manage user sessions.
- Authentication:** The middleware function `isAuthenticated` ensures that only logged-in users can access `admin.html` and `color-admin.html`.
- Endpoints:**
 - `/login`: Handles user login and sets the session.
 - `/admin.html` and `/color-admin.html`: Serves the respective HTML files if the user is authenticated.
 - `/beers`: Serves the beer data from the JSON file.
 - `/save-beers`: Saves the updated beer data to the JSON file.
 - `/colors`: Serves the color data from the JSON file.
 - `/ip`: Returns the local IP address.
- Google Docs Integration:** The function `sendIpToGoogleDocs` updates a Google Docs document with a hyperlink to the local server and a timestamp.

Frontend Changes

Ensure that your `admin.html` page correctly sends the updated beer list to the backend endpoint.

admin.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>

```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Admin - Edit Tap List</title>
<style>
    .editable {
        border: 1px solid #000;
        padding: 5px;
    }
</style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here --&gt;
    &lt;/div&gt;
    &lt;script&gt;
        document.addEventListener('DOMContentLoaded', () =&gt; {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response =&gt; response.json())
                .then(beers =&gt; {
                    beers.forEach((beer, index) =&gt; {
                        const beerDiv = document.createElement('div');
                        beerDiv.textContent = `${index + 1}. ${beer.name} by ${beer.brewery}
- ${beer.abv}`;
                        beerDiv.dataset.index = index;
                        tapListDiv.appendChild(beerDiv);
                    });
                });

            editButton.addEventListener('click', () =&gt; {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div =&gt; {
                    const index = div.dataset.index;
                    const beerText = div.textContent;
                    const [name, rest] = beerText.split(' by ');
                    const [brewery, abv] = rest.split(' - ');

                    div.innerHTML =
                        &lt;input type="text" value="${name}" data-field="name"&gt;
                        &lt;input type="text" value="${brewery}" data-field="brewery"&gt;
                        &lt;input type="text" value="${abv}" data-field="abv"&gt;
                    ;
                    div.classList.add('editable');
                });
                editButton.style.display = 'none';
                saveButton.style.display = 'block';
            });

            saveButton.addEventListener('click', () =&gt; {
                const updatedBeers = [];
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div =&gt; {
                    const inputs = div.querySelectorAll('input');
                    const beer = {};
                    inputs.forEach(input =&gt; {
                        const field = input.dataset.field;
                        beer[field] = input.value;
                    });
                    updatedBeers.push(beer);
                });

                fetch('/save-beers', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify(updatedBeers)
                })
                    .then(response =&gt; response.text())
            });
        });
    &lt;/script&gt;
&lt;/body&gt;
</pre>

```

```

        .then(result => {
            if (result === 'Beers saved successfully') {
                beerDivs.forEach((div, index) => {
                    const beer = updatedBeers[index];
                    div.innerHTML = `${index + 1}. ${beer.name} by ${beer.brewery} - 
${beer.abv}`;
                    div.classList.remove('editable');
                });
                editButton.style.display = 'block';
                saveButton.style.display = 'none';
            } else {
                alert(result);
            }
        })
        .catch(error => {
            alert('Failed to save beers');
            console.error(error);
        });
    });
});
</script>
</body>
</html>

```

Running Your Server

1. Start your server:

```

bash
node server.js

```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

This complete setup should ensure that the `admin.html` page correctly interacts with the backend to edit and save the beer list. Let me know if you encounter any further issues or need further assistance!



beers.json has lost the info for the beer "type". It's just not there anymore. Also when you edit and save the list, it adds the line number again each time. So you end up with 1. 1. 1. Ekto Kooler if you save twice



Let's address both issues:

1. Ensure the beer type is included in the JSON data and handled correctly.
2. Prevent the line number from duplicating each time you save.

Backend Changes

Update the `save-beers` endpoint to ensure it saves all the necessary fields, including the beer type.

Frontend Changes

Ensure the `admin.html` handles all beer fields and prevents the duplication of line numbers.

Updated `server.js`

Ensure the backend can handle saving all fields including the beer type.

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});
```

```

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });
}

```

```

const docs = google.docs({ version: 'v1', auth });
const now = new Date();
const url = `http://${localIp}:3000/admin.html`;
const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
    const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
        return element.endIndex > maxIndex ? element.endIndex : maxIndex;
    }, 1);

    // Adjust the endIndex to exclude the newline character
    const adjustedEndIndex = endIndex - 1;

    // Clear existing content
    await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: {
            requests: [
                {
                    deleteContentRange: {
                        range: {
                            startIndex: 1,
                            endIndex: adjustedEndIndex,
                        },
                    },
                },
                ],
            },
        });
    }

    // Insert new content
    const response = await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: [
            requests: [
                {
                    insertText: {
                        location: {
                            index: 1,
                        },
                        text: textToInsert,
                    },
                },
                {
                    updateTextStyle: {
                        range: {
                            startIndex: 1,
                            endIndex: 22, // Adjust this range to match the length of
                            "Click to edit taplist" (21 characters + 1 to cover the full length)
                        },
                        textStyle: {
                            link: {
                                url: url,
                            },
                            fontSize: {
                                magnitude: 32, // Twice the original font size (16 * 2)
                                unit: 'PT',
                            },
                        },
                        fields: 'link,fontSize',
                    },
                },
            ],
        ],
    });
    console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
}

```

```

sendIpToGoogleDocs(); // Send IP address at startup
setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Updated `admin.html`

Ensure the frontend handles all beer fields and prevents duplication of line numbers.

admin.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
        .editable {
            border: 1px solid #000;
            padding: 5px;
        }
    </style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML = `${index + 1}. <span>${beer.name}</span> by
$beer.brewery - <span>${beer.type}</span> - <span>${beer.abv}</span>`;
                        beerDiv.dataset.index = index;
                        tapListDiv.appendChild(beerDiv);
                    });
                });
        });

        editButton.addEventListener('click', () => {
            const beerDivs = tapListDiv.querySelectorAll('div');
            beerDivs.forEach(div => {
                const index = div.dataset.index;
                const spans = div.querySelectorAll('span');
                const name = spans[0].textContent;
                const brewery = spans[1].textContent;
                const type = spans[2].textContent;
                const abv = spans[3].textContent;

                div.innerHTML =
                    `${index + 1}.
                    <input type="text" value="${name}" data-field="name">
                    by <input type="text" value="${brewery}" data-field="brewery">
                    - <input type="text" value="${type}" data-field="type">
                    - <input type="text" value="${abv}" data-field="abv">
`;
                div.classList.add('editable');
            });
        });
    </script>

```

```

    });
    editButton.style.display = 'none';
    saveButton.style.display = 'block';
});

saveButton.addEventListener('click', () => {
    const updatedBeers = [];
    const beerDivs = tapListDiv.querySelectorAll('div');
    beerDivs.forEach(div => {
        const inputs = div.querySelectorAll('input');
        const beer = {};
        inputs.forEach(input => {
            const field = input.dataset.field;
            beer[field] = input.value;
        });
        updatedBeers.push(beer);
    });

    fetch('/save-beers', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(updatedBeers)
    })
    .then(response => response.text())
    .then(result => {
        if (result === 'Beers saved successfully') {
            beerDivs.forEach((div, index) => {
                const beer = updatedBeers[index];
                div.innerHTML = `${index + 1}. <span>${beer.name}</span> by
<span>${beer.brewery}</span> - <span>${beer.type}</span> - <span>${beer.abv}</span>`;
                div.classList.remove('editable');
            });
            editButton.style.display = 'block';
            saveButton.style.display = 'none';
        } else {
            alert(result);
        }
    })
    .catch(error => {
        alert('Failed to save beers');
        console.error(error);
    });
});
});
</script>
</body>
</html>

```

Explanation

1. Backend:

- The `/save-beers` endpoint now correctly handles all beer fields, including the beer type.

2. Frontend:

- The `admin.html` page now correctly handles all beer fields, including the beer type.
- The line numbers are correctly handled and won't duplicate each time you save.

Running Your Server

1. Start your server:

```
bash
```

```
node server.js
```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

This setup should ensure that the `admin.html` page correctly interacts with the backend to edit and save the beer list, including the beer type, and prevent duplication of line numbers. Let me know if you encounter any further issues or need further assistance!



Can the edit button on admin.html be like 4 times as big.

Also can we add some functionality where one of the beer lines can be faded out or something in the event one of the tap lines being empty. Maybe we use some sort of check box on or something on the admin.html page when they're editing to make a line "empty"



Sure! Let's make the following updates to the `admin.html` page:

1. Increase the size of the edit button.
2. Add functionality to fade out a line when it's empty, using a checkbox in the admin panel.

Updated `admin.html`

Here's how you can update `admin.html` to incorporate these changes:

admin.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
        .editable {
            border: 1px solid #000;
            padding: 5px;
        }
        .faded {
            opacity: 0.5;
        }
        #editButton, #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
    </style>

```

```

        }
    </style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML = `${index + 1}. <span>${beer.name}</span> by
$beer.brewery - <span>${beer.type}</span> - <span>${beer.abv}</span> <input
type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled>;
                        beerDiv.dataset.index = index;
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        tapListDiv.appendChild(beerDiv);
                    });
                });

            editButton.addEventListener('click', () => {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div => {
                    const index = div.dataset.index;
                    const spans = div.querySelectorAll('span');
                    const name = spans[0].textContent;
                    const brewery = spans[1].textContent;
                    const type = spans[2].textContent;
                    const abv = spans[3].textContent;
                    const empty = div.querySelector('input[data-field="empty"]').checked;

                    div.innerHTML =
                        `${index + 1}.
                        <input type="text" value="${name}" data-field="name">
                        by <input type="text" value="${brewery}" data-field="brewery">
                        - <input type="text" value="${type}" data-field="type">
                        - <input type="text" value="${abv}" data-field="abv">
                        <label>Empty <input type="checkbox" data-field="empty" ${empty ?
'checked' : ''}></label>
                    `;
                    div.classList.add('editable');
                    if (empty) {
                        div.classList.add('faded');
                    }
                });
                editButton.style.display = 'none';
                saveButton.style.display = 'block';
            });
        });

        saveButton.addEventListener('click', () => {
            const updatedBeers = [];
            const beerDivs = tapListDiv.querySelectorAll('div');
            beerDivs.forEach(div => {
                const inputs = div.querySelectorAll('input[type="text"]');
                const checkbox = div.querySelector('input[type="checkbox"]');
                const beer = {};
                inputs.forEach(input =>
                    const field = input.dataset.field;
                    beer[field] = input.value
                );
                beer.empty = checkbox.checked;
                updatedBeers.push(beer);
            });
        });
    </script>

```

```

        fetch('/save-beers', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(updatedBeers)
        })
        .then(response => response.text())
        .then(result => {
            if (result === 'Beers saved successfully') {
                beerDivs.forEach((div, index) => {
                    const beer = updatedBeers[index];
                    div.innerHTML = `${index + 1}. <span>${beer.name}</span> by
<span>${beer.brewery}</span> - <span>${beer.type}</span> - <span>${beer.abv}</span> <input
type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled>;
                    div.classList.remove('editable');
                    if (beer.empty) {
                        div.classList.add('faded');
                    } else {
                        div.classList.remove('faded');
                    }
                });
                editButton.style.display = 'block';
                saveButton.style.display = 'none';
            } else {
                alert(result);
            }
        })
        .catch(error => {
            alert('Failed to save beers');
            console.error(error);
        });
    });
}
</script>
</body>
</html>

```

Explanation

1. Increased Edit Button Size:

- The `#editButton` and `#saveButton` styles have been updated to increase the button size.

2. Fading Out Empty Tap Lines:

- Added a checkbox for each beer line to mark it as empty.
- When the checkbox is checked, the line is faded out using the `faded` class.
- When editing, the checkbox is displayed, allowing the user to mark a line as empty.
- On saving, the `empty` state is saved and the line is updated accordingly.

Updated `server.js`

Ensure the backend can handle the `empty` field for each beer.

server.js:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');

```

```

const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        }
    });
});

```

```

        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
        const adjustedEndIndex = endIndex - 1;

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,

```

```

        endIndex: adjustedEndIndex,
    },
},
],
},
}),
};

// Insert new content
const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
        requests: [
            {
                insertText: {
                    location: {
                        index: 1,
                    },
                    text: textToInsert,
                },
            },
            {
                updateTextStyle: {
                    range: {
                        startIndex: 1,
                        endIndex: 22, // Adjust this range to match the length of
"Click to edit taplist" (21 characters + 1 to cover the full length)
                    },
                    textStyle: {
                        link: {
                            url: url,
                        },
                        fontSize: {
                            magnitude: 32, // Twice the original font size (16 * 2)
                            unit: 'PT',
                        },
                        fields: 'link,fontSize',
                    },
                },
            },
        ],
    },
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running Your Server

1. Start your server:

bash

node server.js

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

This updated setup should ensure that the `admin.html` page correctly handles editing, saving, and marking lines as empty, as well as increasing the size of the edit button. Let me know if you encounter any further issues or need further assistance!



When the empty box is checked for an item on admin.html, it needs to make the corresponding item on index.html opaque. And for the opaque text on index.html, can that be almost entirely opaque. I want it almost barely visible when the items are "empty"



Sure! Let's update both `admin.html` and `index.html` to handle the "empty" state, and adjust the opacity of the "empty" items on `index.html`.

Updated `index.html`

Update `index.html` to apply the opacity to "empty" items.

index.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <style>
        body {
            background-color: black;
            color: white;
            font-family: Arial, sans-serif;
        }
        .tap-item {
            font-size: 29pt;
        }
        .tap-item span {
            font-size: 29pt;
            color: white;
        }
        .tap-item .brewery {
            font-size: 21pt;
            color: white;
        }
        .faded {
            opacity: 0.1;
        }
    </style>
</head>
<body>
    <h1 style="text-align: center;">Tap List</h1>
    <div id="tapList"></div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/beers')
```

```

        .then(response => response.json())
        .then(beers => {
            const tapListDiv = document.getElementById('tapList');
            beers.forEach((beer, index) => {
                const beerDiv = document.createElement('div');
                beerDiv.classList.add('tap-item');
                if (beer.empty) {
                    beerDiv.classList.add('faded');
                }
                beerDiv.innerHTML = `${index + 1}. <span>${beer.name}</span> by
<span class="brewery">${beer.brewery}</span> - ${beer.type} - ${beer.abv}`;
                tapListDiv.appendChild(beerDiv);
            });
        });
    </script>
</body>
</html>

```

Updated `admin.html`

Update `admin.html` to handle the "empty" state and save it to the backend.

admin.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
        .editable {
            border: 1px solid #000;
            padding: 5px;
        }
        .faded {
            opacity: 0.5;
        }
        #editButton, #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
    </style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here --&gt;
    &lt;/div&gt;
    &lt;script&gt;
        document.addEventListener('DOMContentLoaded', () =&gt; {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response =&gt; response.json())
                .then(beers =&gt; {
                    beers.forEach((beer, index) =&gt; {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML = `${index + 1}. &lt;span&gt;${beer.name}&lt;/span&gt; by
${beer.brewery}&lt;/span&gt; - &lt;span&gt;${beer.type}&lt;/span&gt; - &lt;span&gt;${beer.abv}&lt;/span&gt; &lt;input
type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled&gt;`;
                    });
                });
        });
    &lt;/script&gt;
&lt;/body&gt;
</pre>

```

```

        beerDiv.dataset.index = index;
        if (beer.empty) {
            beerDiv.classList.add('faded');
        }
        tapListDiv.appendChild(beerDiv);
    });
});

editButton.addEventListener('click', () => {
    const beerDivs = tapListDiv.querySelectorAll('div');
    beerDivs.forEach(div => {
        const index = div.dataset.index;
        const spans = div.querySelectorAll('span');
        const name = spans[0].textContent;
        const brewery = spans[1].textContent;
        const type = spans[2].textContent;
        const abv = spans[3].textContent;
        const empty = div.querySelector('input[data-field="empty"]').checked;

        div.innerHTML =
            `${index + 1}.
            <input type="text" value="${name}" data-field="name">
            by <input type="text" value="${brewery}" data-field="brewery">
            - <input type="text" value="${type}" data-field="type">
            - <input type="text" value="${abv}" data-field="abv">
            <label>Empty <input type="checkbox" data-field="empty" ${empty ?
'checked' : ''}></label>
            `;
        div.classList.add('editable');
        if (empty) {
            div.classList.add('faded');
        }
    });
    editButton.style.display = 'none';
    saveButton.style.display = 'block';
});

saveButton.addEventListener('click', () => {
    const updatedBeers = [];
    const beerDivs = tapListDiv.querySelectorAll('div');
    beerDivs.forEach(div => {
        const inputs = div.querySelectorAll('input[type="text"]');
        const checkbox = div.querySelector('input[type="checkbox"]');
        const beer = {};
        inputs.forEach(input => {
            const field = input.dataset.field;
            beer[field] = input.value;
        });
        beer.empty = checkbox.checked;
        updatedBeers.push(beer);
    });

    fetch('/save-beers', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(updatedBeers)
    })
    .then(response => response.text())
    .then(result => {
        if (result === 'Beers saved successfully') {
            beerDivs.forEach((div, index) => {
                const beer = updatedBeers[index];
                div.innerHTML = `${index + 1}. <span>${beer.name}</span> by
<span>${beer.brewery}</span> - <span>${beer.type}</span> - <span>${beer.abv}</span> <input
type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled>;
                div.classList.remove('editable');
                if (beer.empty) {
                    div.classList.add('faded');
                } else {
                    div.classList.remove('faded');
                }
            });
            editButton.style.display = 'block';
        }
    });
});

```

```

        saveButton.style.display = 'none';
    } else {
        alert(result);
    }
})
.catch(error => {
    alert('Failed to save beers');
    console.error(error);
});
});
</script>
</body>
</html>

```

Updated `server.js`

Ensure the backend can handle the `empty` field for each beer.

server.js:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));
app.post('/login', (req, res) => {

```

```

const { password } = req.body;
if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
} else {
    res.send('Incorrect password');
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
}

```

```

}

const auth = new google.auth.GoogleAuth({
  keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
  file path
  scopes: ['https://www.googleapis.com/auth/documents']
});

const docs = google.docs({ version: 'v1', auth });
const now = new Date();
const url = `http://${localIp}:3000/admin.html`;
const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

try {
  // Get the document to determine its end index
  const doc = await docs.documents.get({ documentId: DOC_ID });
  const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
    return element.endIndex > maxIndex ? element.endIndex : maxIndex;
  }, 1);

  // Adjust the endIndex to exclude the newline character
  const adjustedEndIndex = endIndex - 1;

  // Clear existing content
  await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
      requests: [
        {
          deleteContentRange: {
            range: {
              startIndex: 1,
              endIndex: adjustedEndIndex,
            },
          },
        },
      ],
    },
  });

  // Insert new content
  const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
      requests: [
        {
          insertText: {
            location: {
              index: 1,
            },
            text: textToInsert,
          },
        },
        {
          updateTextStyle: {
            range: {
              startIndex: 1,
              endIndex: 22, // Adjust this range to match the length of
              "Click to edit taplist" (21 characters + 1 to cover the full length)
            },
            textStyle: {
              link: {
                url: url,
              },
              fontSize: {
                magnitude: 32, // Twice the original font size (16 * 2)
                unit: 'PT',
              },
            },
            fields: 'link,fontSize',
          },
        },
      ],
    },
  });
}

```

```

        console.log('IP address sent to Google Docs successfully:', response.data);
    } catch (error) {
        console.error('Failed to send IP address to Google Docs:', error);
    }
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running Your Server

1. Start your server:

```

bash

node server.js

```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

This updated setup should ensure that the `admin.html` page correctly handles editing, saving, and marking lines as empty, and that `index.html` correctly displays the opacity for empty items. Let me know if you encounter any further issues or need further assistance!



Oh no, the index.html lost all of its color formatting. I want it to retain its even and odd line color formatting like it had in this example:

```

"<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->

```

```

</div>
<script src="script.js"></script>
</body>
</html>
"
```



Let's combine the color formatting from the previous example with the new functionality for handling empty lines.

Updated `index.html`

Update `index.html` to retain its even and odd line color formatting and handle empty lines.

index.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    const tapListDiv = document.getElementById('tap-list');
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.classList.add('tap-item');
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        beerDiv.innerHTML = `${index + 1}. <span>${beer.name}</span> by
<span class="brewery">${beer.brewery}</span> - ${beer.type} - ${beer.abv}`;
                        beerDiv.style.backgroundColor = index % 2 === 0 ? '#f58d72' :
                        '#fcda81';
                        tapListDiv.appendChild(beerDiv);
                    });
                });
        });
    </script>
</body>
</html>
```

Updated `styles.css`

Ensure that `styles.css` includes the necessary styles for the color formatting and faded effect.

styles.css:

css

```
body {
    background-color: black;
    color: white;
    font-family: Arial, sans-serif;
}

.banner {
    text-align: center;
    margin-bottom: 20px;
}

#business-name {
    max-width: 100%;
}

.tap-list .tap-item {
    padding: 10px;
    font-size: 24px;
    border-bottom: 1px solid white;
}

.tap-list .tap-item.faded {
    opacity: 0.1;
}

.tap-list .tap-item:nth-child(even) {
    background-color: #f58d72;
}

.tap-list .tap-item:nth-child(odd) {
    background-color: #fcda81;
}

.tap-item span {
    font-size: 29pt;
    color: white;
}

.tap-item .brewery {
    font-size: 21pt;
    color: white;
}
```

Updated `admin.html`

Ensure the admin page retains its functionality and correctly handles the `empty` state.

admin.html:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
```

```

    .editable {
        border: 1px solid #000;
        padding: 5px;
    }
    .faded {
        opacity: 0.5;
    }
    #editButton, #saveButton {
        font-size: 2em; /* Increase font size */
        padding: 10px 20px;
    }
</style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML = `${index + 1}. <span>${beer.name}</span> by
<span>${beer.brewery}</span> - <span>${beer.type}</span> - <span>${beer.abv}</span> <input
type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled>`;
                        beerDiv.dataset.index = index;
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        tapListDiv.appendChild(beerDiv);
                    });
                });

            editButton.addEventListener('click', () => {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div => {
                    const index = div.dataset.index;
                    const spans = div.querySelectorAll('span');
                    const name = spans[0].textContent;
                    const brewery = spans[1].textContent;
                    const type = spans[2].textContent;
                    const abv = spans[3].textContent;
                    const empty = div.querySelector('input[data-field="empty"]').checked;

                    div.innerHTML =
                        `${index + 1}.
                        <input type="text" value="${name}" data-field="name">
                        by <input type="text" value="${brewery}" data-field="brewery">
                        - <input type="text" value="${type}" data-field="type">
                        - <input type="text" value="${abv}" data-field="abv">
                        <label>Empty <input type="checkbox" data-field="empty" ${empty ?
'checked' : ''}></label>
                    `;
                    div.classList.add('editable');
                    if (empty) {
                        div.classList.add('faded');
                    }
                });
                editButton.style.display = 'none';
                saveButton.style.display = 'block';
            });

            saveButton.addEventListener('click', () => {
                const updatedBeers = [];
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div => {

```

```

const inputs = div.querySelectorAll('input[type="text"]');
const checkbox = div.querySelector('input[type="checkbox"]');
const beer = {};
inputs.forEach(input => {
    const field = input.dataset.field;
    beer[field] = input.value;
});
beer.empty = checkbox.checked;
updatedBeers.push(beer);
});

fetch('/save-beers', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(updatedBeers)
})
.then(response => response.text())
.then(result => {
    if (result === 'Beers saved successfully') {
        beerDivs.forEach((div, index) => {
            const beer = updatedBeers[index];
            div.innerHTML = `${index + 1}. <span>${beer.name}</span> by
<span>${beer.brewery}</span> - <span>${beer.type}</span> - <span>${beer.abv}</span> <input
type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled>;
            div.classList.remove('editable');
            if (beer.empty) {
                div.classList.add('faded');
            } else {
                div.classList.remove('faded');
            }
        });
        editButton.style.display = 'block';
        saveButton.style.display = 'none';
    } else {
        alert(result);
    }
})
.catch(error => {
    alert('Failed to save beers');
    console.error(error);
});
});
});
</script>
</body>
</html>

```

Updated `server.js`

Ensure the backend can handle the `empty` field for each beer.

server.js:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;

```

```

const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {

```

```

        res.status(500).send('Error reading colors file');
    } else {
        res.json(JSON.parse(data));
    }
});
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
        const adjustedEndIndex = endIndex - 1;

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,
                                endIndex: adjustedEndIndex,
                            },
                        },
                    ],
                ],
            },
        });
    }

    // Insert new content
}

```

```

const response = await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        insertText: {
          location: {
            index: 1,
          },
          text: textToInsert,
        },
      },
      {
        updateTextStyle: {
          range: {
            startIndex: 1,
            endIndex: 22, // Adjust this range to match the length of
            "Click to edit taplist" (21 characters + 1 to cover the full length)
          },
          textStyle: {
            link: {
              url: url,
            },
            fontSize: {
              magnitude: 32, // Twice the original font size (16 * 2)
              unit: 'PT',
            },
            fields: 'link,fontSize',
          },
        },
      },
    ],
  },
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
  console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running Your Server

1. Start your server:

```

bash
node server.js

```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

This updated setup should ensure that the `admin.html` page correctly handles editing, saving, and marking lines as empty, and that `index.html` retains its even and odd line color formatting and displays the opacity for empty items. Let me know if you encounter any further issues or need further assistance!



can we make the text color on index.html black please? Also it looks like color-admin.html is broken and returns a failed to load colors message when you attempt to edit any colors



Let's address both issues:

1. **Update the text color on `index.html` to black.**
2. **Fix the issue with `color-admin.html` to ensure it loads and saves colors correctly.**

Updated `index.html`

Update the text color to black.

index.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    const tapListDiv = document.getElementById('tap-list');
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.classList.add('tap-item');
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        beerDiv.innerHTML = `${index + 1}. <span>${beer.name}</span> by
<span class="brewery">${beer.brewery}</span> - ${beer.type} - ${beer.abv}`;
                        beerDiv.style.backgroundColor = index % 2 === 0 ? '#f58d72' :
                        '#fcda81';
                        tapListDiv.appendChild(beerDiv);
                    });
                });
        });
    </script>
```

```
</body>
</html>
```

Updated `styles.css`

Ensure the text color is black and that the even and odd rows are styled correctly.

styles.css:

```
css

body {
    background-color: black;
    font-family: Arial, sans-serif;
}

.banner {
    text-align: center;
    margin-bottom: 20px;
}

#business-name {
    max-width: 100%;
}

.tap-list .tap-item {
    padding: 10px;
    font-size: 24px;
    border-bottom: 1px solid white;
    color: black;
}

.tap-list .tap-item.faded {
    opacity: 0.1;
}

.tap-list .tap-item:nth-child(even) {
    background-color: #f58d72;
}

.tap-list .tap-item:nth-child(odd) {
    background-color: #fcda81;
}

.tap-item span {
    font-size: 29pt;
    color: black;
}

.tap-item .brewery {
    font-size: 21pt;
    color: black;
}
```

Updated `color-admin.html`

Fix the issue with loading and saving colors.

color-admin.html:

html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Colors</title>
    <style>
        #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
        body {
            background-color: gray;
            color: white;
            font-family: Arial, sans-serif;
        }
        .color-field {
            margin-bottom: 20px;
        }
    </style>
</head>
<body>
    <h1>Edit Colors</h1>
    <div class="color-field">
        <label for="evenColor">Even Rows Background Color:</label>
        <input type="text" id="evenColor" placeholder="#f58d72">
    </div>
    <div class="color-field">
        <label for="oddColor">Odd Rows Background Color:</label>
        <input type="text" id="oddColor" placeholder="#fcda81">
    </div>
    <button id="saveButton">Save Colors</button>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {
                    document.getElementById('evenColor').value = colors.evenColor;
                    document.getElementById('oddColor').value = colors.oddColor;
                })
                .catch(error => {
                    console.error('Failed to load colors:', error);
                });
        });

        document.getElementById('saveButton').addEventListener('click', () => {
            const colors = {
                evenColor: document.getElementById('evenColor').value,
                oddColor: document.getElementById('oddColor').value,
            };

            fetch('/save-colors', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(colors)
            })
                .then(response => response.text())
                .then(result => {
                    if (result === 'Colors saved successfully') {
                        alert('Colors saved successfully');
                    } else {
                        alert(result);
                    }
                })
                .catch(error => {
                    alert('Failed to save colors');
                    console.error(error);
                });
        });
    </script>
</body>

```

```
</script>
</body>
</html>
```

Updated `server.js`

Ensure the backend can handle saving and loading colors correctly.

server.js:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
```

```

});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
  const updatedBeers = req.body;
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save beers');
    } else {
      res.send('Beers saved successfully');
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
  const updatedColors = req.body;
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save colors');
    } else {
      res.send('Colors saved successfully');
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
}

```

```

}

const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
file path
    scopes: ['https://www.googleapis.com/auth/documents']
});

const docs = google.docs({ version: 'v1', auth });
const now = new Date();
const url = `http://${localIp}:3000/admin.html`;
const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
    const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
        return element.endIndex > maxIndex ? element.endIndex : maxIndex;
    }, 1);

    // Adjust the endIndex to exclude the newline character
    const adjustedEndIndex = endIndex - 1;

    // Clear existing content
    await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: {
            requests: [
                {
                    deleteContentRange: {
                        range: {
                            startIndex: 1,
                            endIndex: adjustedEndIndex,
                        },
                    },
                },
            ],
        },
    });

    // Insert new content
    const response = await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: {
            requests: [
                {
                    insertText: {
                        location: {
                            index: 1,
                        },
                        text: textToInsert,
                    },
                },
                {
                    updateTextStyle: {
                        range: {
                            startIndex: 1,
                            endIndex: 22, // Adjust this range to match the length of
                            "Click to edit taplist" (21 characters + 1 to cover the full length)
                        },
                        textStyle: {
                            link: {
                                url: url,
                            },
                            fontSize: {
                                magnitude: 32, // Twice the original font size (16 * 2)
                                unit: 'PT',
                            },
                        },
                        fields: 'link,fontSize',
                    },
                },
            ],
        },
    });
}

```

```
        console.log('IP address sent to Google Docs successfully:', response.data);
    } catch (error) {
        console.error('Failed to send IP address to Google Docs:', error);
    }
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});
```

Running Your Server

1. Start your server:

```
bash
node server.js
```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This updated setup should ensure that the `index.html` page correctly displays the tap list with the specified colors and the text color set to black, and that `color-admin.html` works correctly for loading and saving colors. Let me know if you encounter any further issues or need further assistance!



I dpm



It seems like there might have been a typo in your last message. Could you please clarify what you need help with?



I don't think color-admin.html is actually changing the background colors for the lines in index.html. I'd like to be able to use hex color codes to be able to modify the background color for the odd lines and even lines. I'd also like a place on color-admin.html to modify the text color on index.html using a hex color code



Let's update the `color-admin.html` page and ensure the server properly handles saving and loading the colors. We will also update `index.html` to dynamically apply the colors from the saved configuration.

Updated `color-admin.html`

Add fields to modify the text color along with the background colors for odd and even lines.

color-admin.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Colors</title>
    <style>
        #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
        body {
            background-color: gray;
            color: white;
            font-family: Arial, sans-serif;
        }
        .color-field {
            margin-bottom: 20px;
        }
    </style>
</head>
<body>
    <h1>Edit Colors</h1>
    <div class="color-field">
        <label for="evenColor">Even Rows Background Color:</label>
        <input type="text" id="evenColor" placeholder="#f58d72">
    </div>
    <div class="color-field">
        <label for="oddColor">Odd Rows Background Color:</label>
        <input type="text" id="oddColor" placeholder="#fcda81">
    </div>
    <div class="color-field">
        <label for="textColor">Text Color:</label>
        <input type="text" id="textColor" placeholder="#000000">
    </div>
    <button id="saveButton">Save Colors</button>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {

```

```

        document.getElementById('evenColor').value = colors.evenColor;
        document.getElementById('oddColor').value = colors.oddColor;
        document.getElementById('textColor').value = colors.textColor;
    })
    .catch(error => {
        console.error('Failed to load colors:', error);
    });

document.getElementById('saveButton').addEventListener('click', () => {
    const colors = {
        evenColor: document.getElementById('evenColor').value,
        oddColor: document.getElementById('oddColor').value,
        textColor: document.getElementById('textColor').value,
    };

    fetch('/save-colors', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(colors)
    })
    .then(response => response.text())
    .then(result => {
        if (result === 'Colors saved successfully') {
            alert('Colors saved successfully');
        } else {
            alert(result);
        }
    })
    .catch(error => {
        alert('Failed to save colors');
        console.error(error);
    });
});

```

</script>

</body>

</html>

Updated `index.html`

Apply the saved colors dynamically.

index.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {

```

```

colors.evenColor);
colors.oddColor);
colors.textColor);

fetch('/beers')
  .then(response => response.json())
  .then(beers => {
    const tapListDiv = document.getElementById('tap-list');
    beers.forEach((beer, index) => {
      const beerDiv = document.createElement('div');
      beerDiv.classList.add('tap-item');
      if (beer.empty) {
        beerDiv.classList.add('faded');
      }
      beerDiv.innerHTML = `${index + 1}. <span>${beer.name}</span>
by <span class="brewery">${beer.brewery}</span> - ${beer.type} - ${beer.abv}`;
      beerDiv.style.backgroundColor = index % 2 === 0 ?
colors.evenColor : colors.oddColor;
      beerDiv.style.color = colors.textColor;
      tapListDiv.appendChild(beerDiv);
    });
  });
</script>
</body>
</html>

```

Updated `styles.css`

Ensure the CSS variables are used for dynamic styling.

styles.css:

css

```

:root {
  --even-color: #f58d72;
  --odd-color: #fcda81;
  --text-color: black;
}

body {
  background-color: black;
  font-family: Arial, sans-serif;
}

.banner {
  text-align: center;
  margin-bottom: 20px;
}

#business-name {
  max-width: 100%;
}

.tap-list .tap-item {
  padding: 10px;
  font-size: 24px;
  border-bottom: 1px solid white;
  color: var(--text-color);
}

.tap-list .tap-item.faded {
  opacity: 0.1;
}

```

```

}
.tap-list .tap-item:nth-child(even) {
  background-color: var(--even-color);
}

.tap-list .tap-item:nth-child(odd) {
  background-color: var(--odd-color);
}

.tap-item span {
  font-size: 29pt;
  color: var(--text-color);
}

.tap-item .brewery {
  font-size: 21pt;
  color: var(--text-color);
}

```

Updated `server.js`

Ensure the backend can handle saving and loading colors correctly.

server.js:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
  if (req.path === '/admin.html' || req.path === '/color-admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

```

```

});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
  const updatedBeers = req.body;
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save beers');
    } else {
      res.send('Beers saved successfully');
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
  const updatedColors = req.body;
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save colors');
    } else {
      res.send('Colors saved successfully');
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
});

```

```

    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    const now = new Date();
    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
        const adjustedEndIndex = endIndex - 1;

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,
                                endIndex: adjustedEndIndex,
                            },
                        },
                    },
                    ],
                },
            });
        }

        // Insert new content
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                            text: textToInsert,
                        },
                    },
                    {
                        updateTextStyle: {
                            range: {
                                startIndex: 1,
                                endIndex: 22, // Adjust this range to match the length of
                                "Click to edit taplist" (21 characters + 1 to cover the full length)
                            }
                        }
                    }
                ]
            }
        });
    }
}

```

```

        },
        textStyle: {
            link: {
                url: url,
            },
            fontSize: {
                magnitude: 32, // Twice the original font size (16 * 2)
                unit: 'PT',
            },
            fields: 'link,fontSize',
        },
    },
],
),
);
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running Your Server

1. Start your server:

bash

node server.js

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This updated setup should ensure that the `index.html` page correctly displays the tap list with the specified colors and that `color-admin.html` works correctly for loading and saving colors. Let me know if you encounter any further issues or need further assistance!



can we remove the white line in between the even and odd lines. Can we also make all of the text on

the even and odd lines bold except for the word "by" and the number at the start of each line?



Certainly! Let's update the `styles.css` file to remove the white line between the even and odd lines and apply the text styling changes.

Updated `styles.css`

Make the necessary changes to the CSS for the tap list.

styles.css:

```
css

:root {
  --even-color: #f58d72;
  --odd-color: #fcda81;
  --text-color: black;
}

body {
  background-color: black;
  font-family: Arial, sans-serif;
}

.banner {
  text-align: center;
  margin-bottom: 20px;
}

#business-name {
  max-width: 100%;
}

.tap-list .tap-item {
  padding: 10px;
  font-size: 24px;
  color: var(--text-color);
}

.tap-list .tap-item.faded {
  opacity: 0.1;
}

.tap-list .tap-item:nth-child(even) {
  background-color: var(--even-color);
}

.tap-list .tap-item:nth-child(odd) {
  background-color: var(--odd-color);
}

.tap-item span {
  font-size: 29pt;
  color: var(--text-color);
  font-weight: bold;
}

.tap-item .brewery {
  font-size: 21pt;
  color: var(--text-color);
  font-weight: bold;
}
```

```
.tap-item .by, .tap-item .number {
    font-weight: normal;
}
```

Updated `index.html`

Ensure the `index.html` applies the correct classes for the new styles.

index.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {
                    document.documentElement.style.setProperty('--even-color',
colors.evenColor);
                    document.documentElement.style.setProperty('--odd-color',
colors.oddColor);
                    document.documentElement.style.setProperty('--text-color',
colors.textColor);

                    fetch('/beers')
                        .then(response => response.json())
                        .then(beers => {
                            const tapListDiv = document.getElementById('tap-list');
                            beers.forEach((beer, index) => {
                                const beerDiv = document.createElement('div');
                                beerDiv.classList.add('tap-item');
                                if (beer.empty) {
                                    beerDiv.classList.add('faded');
                                }
                                beerDiv.innerHTML = `<span class="number">${index + 1}</span><span class="name">${beer.name}</span> <span class="by">by</span> <span class="brewery">${beer.brewery}</span> - ${beer.type} - ${beer.abv}`;
                                beerDiv.style.backgroundColor = index % 2 === 0 ? colors.evenColor : colors.oddColor;
                                beerDiv.style.color = colors.textColor;
                                tapListDiv.appendChild(beerDiv);
                            });
                        });
                });
        </script>
    </body>
</html>
```

Updated `admin.html`

Ensure the `admin.html` applies the correct classes for the new styles.

admin.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
        .editable {
            border: 1px solid #000;
            padding: 5px;
        }
        .faded {
            opacity: 0.5;
        }
        #editButton, #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
    </style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML = `<span class="number">${index + 1}. </span>
<span>${beer.name}</span> <span class="by">by</span> <span class="brewery">${beer.brewery}
</span> - ${beer.type} - ${beer.abv} <input type="checkbox" data-field="empty" ${beer.empty
? 'checked' : ''} disabled>`;
                        beerDiv.dataset.index = index;
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        tapListDiv.appendChild(beerDiv);
                    });
                });
            editButton.addEventListener('click', () => {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div => {
                    const index = div.dataset.index;
                    const spans = div.querySelectorAll('span');
                    const name = spans[1].textContent;
                    const brewery = spans[3].textContent;
                    const type = spans[4].textContent;
                    const abv = spans[5].textContent;
                    const empty = div.querySelector('input[data-field="empty"]').checked;

                    div.innerHTML =
                        `<span class="number">${index + 1}. </span>
<input type="text" value="${name}" data-field="name">
<span class="by">by</span> <input type="text" value="${brewery}">`;
                });
            });
        });
    </script>
</body>

```

```

data-field="brewery">
    - <input type="text" value="${type}" data-field="type">
    - <input type="text" value="${abv}" data-field="abv">
        <label>Empty <input type="checkbox" data-field="empty" ${empty ? 'checked' : ''}></label>
    `;
    div.classList.add('editable');
    if (empty) {
        div.classList.add('faded');
    }
});
editButton.style.display = 'none';
saveButton.style.display = 'block';
});

saveButton.addEventListener('click', () => {
const updatedBeers = [];
const beerDivs = tapListDiv.querySelectorAll('div');
beerDivs.forEach(div => {
    const inputs = div.querySelectorAll('input[type="text"]');
    const checkbox = div.querySelector('input[type="checkbox"]');
    const beer = {};
    inputs.forEach(input => {
        const field = input.dataset.field;
        beer[field] = input.value;
    });
    beer.empty = checkbox.checked;
    updatedBeers.push(beer);
});

fetch('/save-beers', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(updatedBeers)
})
.then(response => response.text())
.then(result => {
    if (result === 'Beers saved successfully') {
        beerDivs.forEach((div, index) => {
            const beer = updatedBeers[index];
            div.innerHTML = `<span class="number">${index + 1}. </span>
<span>${beer.name}</span> <span class="by">by</span> <span class="brewery">${beer.brewery}
</span> - ${beer.type} - ${beer.abv} <input type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''}>`;
            div.classList.remove('editable');
            if (beer.empty) {
                div.classList.add('faded');
            } else {
                div.classList.remove('faded');
            }
        });
        editButton.style.display = 'block';
        saveButton.style.display = 'none';
    } else {
        alert(result);
    }
})
.catch(error => {
    alert('Failed to save beers');
    console.error(error);
});
});
});
</script>
</body>
</html>

```

Running Your Server

1. Start your server:

```
bash
node server.js
```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This setup should ensure that the `index.html` page correctly displays the tap list with the specified styles and that `color-admin.html` works correctly for loading and saving colors. Let me know if you encounter any further issues or need further assistance!



1. Ekto Kooler by Eagle Park Brewing Co - Hard Seltzer - 5.0%

this is how I'd like the text in the even and odd lines to be formatted



To achieve the text formatting as shown in the image, we need to adjust the styles and the structure of the HTML for the tap list items.

Updated `styles.css`

Adjust the styles to match the desired formatting.

styles.css:

```
css
:root {
  --even-color: #f58d72;
  --odd-color: #fcda81;
  --text-color: black;
}
body {
```

```

background-color: black;
font-family: Arial, sans-serif;
}

.banner {
  text-align: center;
  margin-bottom: 20px;
}

#business-name {
  max-width: 100%;
}

.tap-list .tap-item {
  padding: 10px;
  font-size: 24px;
  color: var(--text-color);
  display: flex;
  align-items: center;
}

.tap-list .tap-item.faded {
  opacity: 0.1;
}

.tap-list .tap-item:nth-child(even) {
  background-color: var(--even-color);
}

.tap-list .tap-item:nth-child(odd) {
  background-color: var(--odd-color);
}

.tap-item .number {
  font-size: 29pt;
  font-weight: normal;
  margin-right: 10px;
  color: var(--text-color);
}

.tap-item .name,
.tap-item .brewery,
.tap-item .type,
.tap-item .abv {
  font-size: 29pt;
  font-weight: bold;
  color: var(--text-color);
}

.tap-item .by {
  font-size: 29pt;
  font-weight: normal;
  margin: 0 5px;
  color: var(--text-color);
}

```

Updated `index.html`

Ensure the structure of the tap list items matches the desired formatting.

index.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Tap List</title>
<link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="banner">
    
  </div>
  <div class="tap-list" id="tap-list">
    <!-- Beers will be dynamically inserted here -->
  </div>
  <script>
    document.addEventListener('DOMContentLoaded', () => {
      fetch('/colors')
        .then(response => response.json())
        .then(colors => {
          document.documentElement.style.setProperty('--even-color',
colors.evenColor);
          document.documentElement.style.setProperty('--odd-color',
colors.oddColor);
          document.documentElement.style.setProperty('--text-color',
colors.textColor);

          fetch('/beers')
            .then(response => response.json())
            .then(beers => {
              const tapListDiv = document.getElementById('tap-list');
              beers.forEach((beer, index) => {
                const beerDiv = document.createElement('div');
                beerDiv.classList.add('tap-item');
                if (beer.empty) {
                  beerDiv.classList.add('faded');
                }
                beerDiv.innerHTML = `<span class="number">${index + 1}.
</span> <span class="name">${beer.name}</span> <span class="by">by</span> <span
class="brewery">${beer.brewery}</span> - <span class="type">${beer.type}</span> - <span
class="abv">${beer.abv}</span>`;
                beerDiv.style.backgroundColor = index % 2 === 0 ?
colors.evenColor : colors.oddColor;
                beerDiv.style.color = colors.textColor;
                tapListDiv.appendChild(beerDiv);
              });
            });
        });
    </script>
</body>
</html>

```

Running Your Server

1. Start your server:

```
bash
```

```
node server.js
```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This setup should ensure that the `index.html` page correctly displays the tap list with the desired formatting, and that `color-admin.html` works correctly for loading and saving colors. Let me know if you encounter any further issues or need further assistance!



can we change the style css back to this type of font and color styling

```
:root {  
  --odd-line-color: #fcda81; /* Default color */  
  --even-line-color: #f58d72; /* Default color */  
  --text-color: #000000; /* Default text color */  
}  
  
body {  
  font-family: Arial, sans-serif;  
  background-color: #000; /* Black background */  
  color: var(--text-color); /* Text color */  
  margin: 0;  
  padding: 0;  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  justify-content: start;  
}  
  
.banner {  
  width: 100%;  
  text-align: center;  
  margin: 20px 0;  
}  
  
.business-name {  
  max-width: 100%;  
  height: auto;  
}  
  
.tap-list {  
  width: 90%;  
  margin: 0 auto;  
}  
  
.beer {  
  padding: 10px 0;  
}
```

```

background-color: var(--odd-line-color); /* Light yellow background for odd items */
color: var(--text-color); /* Text color */
display: flex;
align-items: center;
justify-content: space-between;
}

.beer:nth-child(even) {
  background-color: var(--even-line-color); /* Light red background for even items */
}

.beer p {
  margin: 0;
  padding: 10px;
  display: flex;
  align-items: center;
  color: var(--text-color); /* Text color */
  flex-wrap: wrap; /* Allow text to wrap on smaller screens */
}

/* Responsive styles */
@media (max-width: 600px) {
  .beer {
    flex-direction: column;
    align-items: flex-start;
  }

  .beer p {
    font-size: 20px; /* Adjust font size for smaller screens */
  }
}
"
```



Sure! Let's update the `styles.css` file back to the specified font and color styling and adjust the `index.html` and `admin.html` accordingly.

Updated `styles.css`

Revert the styles to match the specified font and color styling.

styles.css:

```
css

:root {
  --odd-line-color: #fcda81; /* Default color */
  --even-line-color: #f58d72; /* Default color */
  --text-color: #000000; /* Default text color */
}
```

```

body {
    font-family: Arial, sans-serif;
    background-color: #000; /* Black background */
    color: var(--text-color); /* Text color */
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: start;
}

.banner {
    width: 100%;
    text-align: center;
    margin: 20px 0;
}

#business-name {
    max-width: 100%;
    height: auto;
}

.tap-list {
    width: 90%;
    margin: 0 auto;
}

.beer {
    padding: 10px 0;
    background-color: var(--odd-line-color); /* Light yellow background for odd items */
    color: var(--text-color); /* Text color */
    display: flex;
    align-items: center;
    justify-content: space-between;
}

.beer:nth-child(even) {
    background-color: var(--even-line-color); /* Light red background for even items */
}

.beer p {
    margin: 0;
    padding: 10px;
    display: flex;
    align-items: center;
    color: var(--text-color); /* Text color */
    flex-wrap: wrap; /* Allow text to wrap on smaller screens */
}

/* Responsive styles */
@media (max-width: 600px) {
    .beer {
        flex-direction: column;
        align-items: flex-start;
    }

    .beer p {
        font-size: 20px; /* Adjust font size for smaller screens */
    }
}

```

Updated `index.html`

Ensure the structure of the tap list items matches the desired formatting and the CSS changes.

index.html:

html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {
                    document.documentElement.style.setProperty('--even-line-color',
colors.evenColor);
                    document.documentElement.style.setProperty('--odd-line-color',
colors.oddColor);
                    document.documentElement.style.setProperty('--text-color',
colors.textColor);

                    fetch('/beers')
                        .then(response => response.json())
                        .then(beers => {
                            const tapListDiv = document.getElementById('tap-list');
                            beers.forEach((beer, index) => {
                                const beerDiv = document.createElement('div');
                                beerDiv.classList.add('beer');
                                if (beer.empty) {
                                    beerDiv.classList.add('faded');
                                }
                                beerDiv.innerHTML = `
                                    <p>
                                        <span class="number">${index + 1}</span>
                                        <span class="name">${beer.name}</span>
                                        <span class="by">by</span>
                                        <span class="brewery">${beer.brewery}</span> -
                                        <span class="type">${beer.type}</span> -
                                        <span class="abv">${beer.abv}</span>
                                    </p>`;
                                tapListDiv.appendChild(beerDiv);
                            });
                        });
                });
            });
        </script>
    </body>
</html>

```

Running Your Server

1. Start your server:

bash

node server.js

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This setup should ensure that the `index.html` page correctly displays the tap list with the specified styles and that `color-admin.html` works correctly for loading and saving colors. Let me know if you encounter any further issues or need further assistance!



Here is an example of what I'm mostly looking for as far as font size and formatting for each odd and even line:

```
"<td><span STYLE="font-size: 29pt;">1.&nbsp;&nbsp;</span><span STYLE="font-size: 29pt; color: black; font-weight: bold">Ekto Kooler</span><span STYLE="font-size: 21pt; color: black"> by<b> Eagle Park Brewing Co - Hard Seltzer - 5.0%</b></span></td>"
```



Thank you for the clarification. Let's update the `index.html` and `styles.css` to match the desired font size and formatting for each odd and even line.

Updated `styles.css`

Make the necessary adjustments to the CSS for font sizes and bold formatting.

styles.css:

```
css

:root {
    --odd-line-color: #fcda81; /* Default color */
    --even-line-color: #f58d72; /* Default color */
    --text-color: #000000; /* Default text color */
}

body {
    font-family: Arial, sans-serif;
    background-color: #000; /* Black background */
    color: var(--text-color); /* Text color */
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
```

```
justify-content: start;
}

.banner {
  width: 100%;
  text-align: center;
  margin: 20px 0;
}

#business-name {
  max-width: 100%;
  height: auto;
}

.tap-list {
  width: 90%;
  margin: 0 auto;
}

.beer {
  padding: 10px 0;
  background-color: var(--odd-line-color); /* Light yellow background for odd items */
  color: var(--text-color); /* Text color */
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.beer:nth-child(even) {
  background-color: var(--even-line-color); /* Light red background for even items */
}

.beer p {
  margin: 0;
  padding: 10px;
  display: flex;
  align-items: center;
  color: var(--text-color); /* Text color */
  flex-wrap: wrap; /* Allow text to wrap on smaller screens */
}

.beer .number {
  font-size: 29pt;
  font-weight: normal;
  margin-right: 10px;
  color: var(--text-color);
}

.beer .name {
  font-size: 29pt;
  font-weight: bold;
  color: var(--text-color);
}

.beer .by {
  font-size: 21pt;
  font-weight: normal;
  margin: 0 5px;
  color: var(--text-color);
}

.beer .brewery,
.beer .type,
.beer .abv {
  font-size: 21pt;
  font-weight: bold;
  color: var(--text-color);
}

/* Responsive styles */
@media (max-width: 600px) {
  .beer {
    flex-direction: column;
    align-items: flex-start;
  }
}
```

```
.beer p {
    font-size: 20px; /* Adjust font size for smaller screens */
}
```

Updated `index.html`

Ensure the structure of the tap list items matches the desired formatting and the CSS changes.

index.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {
                    document.documentElement.style.setProperty('--even-line-color',
colors.evenColor);
                    document.documentElement.style.setProperty('--odd-line-color',
colors.oddColor);
                    document.documentElement.style.setProperty('--text-color',
colors.textColor);

                    fetch('/beers')
                        .then(response => response.json())
                        .then(beers => {
                            const tapListDiv = document.getElementById('tap-list');
                            beers.forEach((beer, index) => {
                                const beerDiv = document.createElement('div');
                                beerDiv.classList.add('beer');
                                if (beer.empty) {
                                    beerDiv.classList.add('faded');
                                }
                                beerDiv.innerHTML = `
                                    <p>
                                        <span class="number">${index + 1}</span>
                                        <span class="name">${beer.name}</span>
                                        <span class="by"> by </span>
                                        <span class="brewery">${beer.brewery} - ${beer.type}
- ${beer.abv}</span>
                                    </p>`;
                                tapListDiv.appendChild(beerDiv);
                            });
                        });
                    });
                });
            </script>
```

```
</body>
</html>
```

Running Your Server

1. Start your server:

```
bash
node server.js
```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This setup should ensure that the `index.html` page correctly displays the tap list with the specified styles and that `color-admin.html` works correctly for loading and saving colors. Let me know if you encounter any further issues or need further assistance!



can the font size on the word "by" be just a little bit smaller?



Sure! Let's slightly reduce the font size for the word "by" in the CSS.

Updated `styles.css`

Adjust the font size for the `.by` class.

styles.css:

```
css
:root {
    --odd-line-color: #fcda81; /* Default color */
    --even-line-color: #f58d72; /* Default color */
    --text-color: #000000; /* Default text color */
}
```

```
body {  
    font-family: Arial, sans-serif;  
    background-color: #000; /* Black background */  
    color: var(--text-color); /* Text color */  
    margin: 0;  
    padding: 0;  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
    justify-content: start;  
}  
  
.banner {  
    width: 100%;  
    text-align: center;  
    margin: 20px 0;  
}  
  
#business-name {  
    max-width: 100%;  
    height: auto;  
}  
  

```

```

    font-weight: bold;
    color: var(--text-color);
}

/* Responsive styles */
@media (max-width: 600px) {
    .beer {
        flex-direction: column;
        align-items: flex-start;
    }

    .beer p {
        font-size: 20px; /* Adjust font size for smaller screens */
    }
}

```

Updated `index.html`

Ensure the structure of the tap list items matches the desired formatting and the CSS changes.

index.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {
                    document.documentElement.style.setProperty('--even-line-color',
colors.evenColor);
                    document.documentElement.style.setProperty('--odd-line-color',
colors.oddColor);
                    document.documentElement.style.setProperty('--text-color',
colors.textColor);

                    fetch('/beers')
                        .then(response => response.json())
                        .then(beers => {
                            const tapListDiv = document.getElementById('tap-list');
                            beers.forEach((beer, index) => {
                                const beerDiv = document.createElement('div');
                                beerDiv.classList.add('beer');
                                if (beer.empty) {
                                    beerDiv.classList.add('faded');
                                }
                                beerDiv.innerHTML = `
                                    <p>
                                        <span class="number">${index + 1}.&nbsp;&nbsp;
                                        <span class="name">${beer.name}</span>
                                        <span class="by"> by </span>
                                        <span class="brewery">${beer.brewery} - ${beer.type}</span>
                                    </p>
                                `;
                                tapListDiv.appendChild(beerDiv);
                            });
                        });
                });
        });
    </script>
</body>

```

```

- ${beer.abv}</span>
          </p>';
      tapListDiv.appendChild(beerDiv);
    });
  });
});
</script>
</body>
</html>

```

Running Your Server

1. Start your server:

bash

node server.js

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This setup should ensure that the `index.html` page correctly displays the tap list with the specified styles, including the smaller font size for the word "by", and that `color-admin.html` works correctly for loading and saving colors. Let me know if you encounter any further issues or need further assistance!



on index.html, the bottom's of the text are not all in a line. Some of the text is raised. Can we have the bottom of all the text nicely in a line?



Sure, we can achieve this by using CSS flexbox properties to ensure the text elements align properly. Let's update the `styles.css` to ensure all text elements align to the baseline.

Updated `styles.css`

Adjust the CSS to align text elements properly.

styles.css:

```
css

:root {
  --odd-line-color: #fcda81; /* Default color */
  --even-line-color: #f58d72; /* Default color */
  --text-color: #000000; /* Default text color */
}

body {
  font-family: Arial, sans-serif;
  background-color: #000; /* Black background */
  color: var(--text-color); /* Text color */
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: start;
}

.banner {
  width: 100%;
  text-align: center;
  margin: 20px 0;
}

#business-name {
  max-width: 100%;
  height: auto;
}

.tap-list {
  width: 90%;
  margin: 0 auto;
}

.beer {
  padding: 10px 0;
  background-color: var(--odd-line-color); /* Light yellow background for odd items */
  color: var(--text-color); /* Text color */
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.beer:nth-child(even) {
  background-color: var(--even-line-color); /* Light red background for even items */
}

.beer p {
  margin: 0;
  padding: 10px;
  display: flex;
  align-items: baseline; /* Align items to the baseline */
  color: var(--text-color); /* Text color */
  flex-wrap: wrap; /* Allow text to wrap on smaller screens */
}

.beer .number {
  font-size: 29pt;
  font-weight: normal;
  margin-right: 10px;
  color: var(--text-color);
}

.beer .name {
  font-size: 29pt;
  font-weight: bold;
  color: var(--text-color);
}
```

```
.beer .by {
    font-size: 18pt; /* Slightly smaller font size */
    font-weight: normal;
    margin: 0 5px;
    color: var(--text-color);
}

.beer .brewery,
.beer .type,
.beer .abv {
    font-size: 21pt;
    font-weight: bold;
    color: var(--text-color);
}

/* Responsive styles */
@media (max-width: 600px) {
    .beer {
        flex-direction: column;
        align-items: flex-start;
    }

    .beer p {
        font-size: 20px; /* Adjust font size for smaller screens */
    }
}
```

Updated `index.html`

Ensure the structure of the tap list items matches the desired formatting and the CSS changes.

index.html:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {
                    document.documentElement.style.setProperty('--even-line-color',
colors.evenColor);
                    document.documentElement.style.setProperty('--odd-line-color',
colors.oddColor);
                    document.documentElement.style.setProperty('--text-color',
colors.textColor);

                    fetch('/beers')
                        .then(response => response.json())
                        .then(beers => {
                            const tapListDiv = document.getElementById('tap-list');
                            beers.forEach((beer, index) => {

```

```

const beerDiv = document.createElement('div');
beerDiv.classList.add('beer');
if (beer.empty) {
    beerDiv.classList.add('faded');
}
beerDiv.innerHTML =
    <p>
        <span class="number">${index + 1}  ;
        <span class="name">${beer.name}</span>
        <span class="by"> by </span>
        <span class="brewery">${beer.brewery} - ${beer.type}</span>
    - ${beer.abv}</span>
    </p>;
tapListDiv.appendChild(beerDiv);
});
});
</script>
</body>
</html>

```

Running Your Server

1. Start your server:

```

bash
node server.js

```

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This setup should ensure that the `index.html` page correctly displays the tap list with aligned text, and that `color-admin.html` works correctly for loading and saving colors. Let me know if you encounter any further issues or need further assistance!



When we mark something as empty using admin.html, can you make it so just the text turns opaque, not the background



Certainly! Let's adjust the `styles.css` and `index.html` to ensure that when an item is marked as empty, only the text turns opaque, not the background.

Updated `styles.css`

Update the CSS to ensure that only the text becomes opaque when an item is marked as empty.

styles.css:

```
css

:root {
  --odd-line-color: #fcda81; /* Default color */
  --even-line-color: #f58d72; /* Default color */
  --text-color: #000000; /* Default text color */
}

body {
  font-family: Arial, sans-serif;
  background-color: #000; /* Black background */
  color: var(--text-color); /* Text color */
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: start;
}

.banner {
  width: 100%;
  text-align: center;
  margin: 20px 0;
}

#business-name {
  max-width: 100%;
  height: auto;
}

.tap-list {
  width: 90%;
  margin: 0 auto;
}

.beer {
  padding: 10px 0;
  background-color: var(--odd-line-color); /* Light yellow background for odd items */
  color: var(--text-color); /* Text color */
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.beer:nth-child(even) {
  background-color: var(--even-line-color); /* Light red background for even items */
}

.beer p {
  margin: 0;
  padding: 10px;
  display: flex;
  align-items: baseline; /* Align items to the baseline */
  color: var(--text-color); /* Text color */
}
```

```

flex-wrap: wrap; /* Allow text to wrap on smaller screens */
}

.beer .number {
  font-size: 29pt;
  font-weight: normal;
  margin-right: 10px;
  color: var(--text-color);
}

.beer .name {
  font-size: 29pt;
  font-weight: bold;
  color: var(--text-color);
}

.beer .by {
  font-size: 18pt; /* Slightly smaller font size */
  font-weight: normal;
  margin: 0 5px;
  color: var(--text-color);
}

.beer .brewery,
.beer .type,
.beer .abv {
  font-size: 21pt;
  font-weight: bold;
  color: var(--text-color);
}

.beer.faded p {
  opacity: 0.1;
}

/* Responsive styles */
@media (max-width: 600px) {
  .beer {
    flex-direction: column;
    align-items: flex-start;
  }

  .beer p {
    font-size: 20px; /* Adjust font size for smaller screens */
  }
}

```

Updated `index.html`

Ensure the structure of the tap list items matches the desired formatting and the CSS changes.

index.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Tap List</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="banner">
    
  </div>
  <div class="tap-list" id="tap-list">

```

```

<!-- Beers will be dynamically inserted here -->
</div>
<script>
    document.addEventListener('DOMContentLoaded', () => {
        fetch('/colors')
            .then(response => response.json())
            .then(colors => {
                document.documentElement.style.setProperty('--even-line-color',
colors.evenColor);
                document.documentElement.style.setProperty('--odd-line-color',
colors.oddColor);
                document.documentElement.style.setProperty('--text-color',
colors.textColor);

                fetch('/beers')
                    .then(response => response.json())
                    .then(beers => {
                        const tapListDiv = document.getElementById('tap-list');
                        beers.forEach((beer, index) => {
                            const beerDiv = document.createElement('div');
                            beerDiv.classList.add('beer');
                            if (beer.empty) {
                                beerDiv.classList.add('faded');
                            }
                            beerDiv.innerHTML =
                                <p>
                                    <span class="number">${index + 1}&nbsp;&nbsp;
                                    <span class="name">${beer.name}</span>
                                    <span class="by"> by </span>
                                    <span class="brewery">${beer.brewery} - ${beer.type}
- ${beer.abv}</span>
                                </p>;
                            tapListDiv.appendChild(beerDiv);
                        });
                    });
                });
            });
        </script>
</body>
</html>

```

Updated `admin.html`

Ensure the structure of the admin list items matches the desired formatting and the CSS changes.

admin.html:

```

html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
        .editable {
            border: 1px solid #000;
            padding: 5px;
        }
        .faded {
            opacity: 0.5;
        }
        #editButton, #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
    </style>

```

```

</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML = `
                            <p>
                                <span class="number">${index + 1}</span>
                                <span class="name">${beer.name}</span>
                                <span class="by"> by </span>
                                <span class="brewery">${beer.brewery}</span> -
                                <span class="type">${beer.type}</span> -
                                <span class="abv">${beer.abv}</span>
                                <label>Empty <input type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled></label>
                            </p>;
                        beerDiv.dataset.index = index;
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        tapListDiv.appendChild(beerDiv);
                    });
                });

            editButton.addEventListener('click', () => {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div => {
                    const index = div.dataset.index;
                    const spans = div.querySelectorAll('span');
                    const name = spans[1].textContent;
                    const brewery = spans[3].textContent;
                    const type = spans[4].textContent;
                    const abv = spans[5].textContent;
                    const empty = div.querySelector('input[data-field="empty"]').checked;

                    div.innerHTML =
                        <span class="number">${index + 1}</span>
                        <input type="text" value="${name}" data-field="name">
                        <span class="by"> by </span>
                        <input type="text" value="${brewery}" data-field="brewery">
                        - <input type="text" value="${type}" data-field="type">
                        - <input type="text" value="${abv}" data-field="abv">
                        <label>Empty <input type="checkbox" data-field="empty" ${empty ? 'checked' : ''}></label>
                );
                div.classList.add('editable');
                if (empty) {
                    div.classList.add('faded');
                }
            });
            editButton.style.display = 'none';
            saveButton.style.display = 'block';
        });
    
```

saveButton.addEventListener('click', () => {
 const updatedBeers = [];
 const beerDivs = tapListDiv.querySelectorAll('div');
 beerDivs.forEach(div => {
 const inputs = div.querySelectorAll('input[type="text"]');
 const checkbox = div.querySelector('input[type="checkbox"]');
 const beer = {};
 });
});

```

inputs.forEach(input => {
    const field = input.dataset.field;
    beer[field] = input.value;
});
beer.empty = checkbox.checked;
updatedBeers.push(beer);
});

fetch('/save-beers', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(updatedBeers)
})
.then(response => response.text())
.then(result => {
    if (result === 'Beers saved successfully') {
        beerDivs.forEach((div, index) => {
            const beer = updatedBeers[index];
            div.innerHTML =
                <p>
                    <span class="number">${index + 1}</span>
                    <span class="name">${beer.name}</span>
                    <span class="by"> by </span>
                    <span class="brewery">${beer.brewery}</span> -
                    <span class="type">${beer.type}</span> -
                    <span class="abv">${beer.abv}</span>
                    <label>Empty <input type="checkbox" data-field="empty"
${beer.empty ? 'checked' : ''} disabled></label>
                </p>;
            div.classList.remove('editable');
            if (beer.empty) {
                div.classList.add('faded');
            } else {
                div.classList.remove('faded');
            }
        });
        editButton.style.display = 'block';
        saveButton.style.display = 'none';
    } else {
        alert(result);
    }
})
.catch(error => {
    alert('Failed to save beers');
    console.error(error);
});
});
});
</script>
</body>
</html>

```

Running Your Server

1. Start your server:

bash

node server.js

2. Access the Admin Page:

Open your browser and navigate to `http://localhost:3000/admin.html` to edit the tap list.

3. Access the Tap List Page:

Open your browser and navigate to `http://localhost:3000` to view the tap list.

4. Access the Color Admin Page:

Open your browser and navigate to `http://localhost:3000/color-admin.html` to edit the colors.

This setup should ensure that when an item is marked as empty, only the text becomes opaque while the background remains unaffected, and that the text aligns properly along the baseline. Let me know if you encounter any further issues or need further assistance!



Would you please write me some step by step instruction documentation on how to setup the Google Docs API we're using to print sendIpToGoogleDocs. I'm mainly looking for how to set it up on the google side and how I would go about it making a new google account to do so



Certainly! Here's a step-by-step guide on how to set up the Google Docs API to work with your project, focusing on setting it up with a new Google account and creating the necessary credentials.

Step-by-Step Instructions for Setting Up Google Docs API

Step 1: Create a Google Account

If you don't already have a Google account, you'll need to create one.

1. Go to [Google Sign Up](#).
2. Follow the instructions to create a new account.

Step 2: Enable Google Docs API

1. Go to the [Google Cloud Console](#).
2. Click on the project dropdown at the top left and select "New Project."
3. Name your project and click "Create."
4. Once the project is created, make sure it's selected. Then, go to the [API & Services Dashboard](#).
5. Click "Enable APIs and Services."
6. Search for "Google Docs API" and select it.
7. Click "Enable."

Step 3: Create Service Account Credentials

1. In the Google Cloud Console, navigate to "APIs & Services" > "Credentials."
2. Click "Create Credentials" and select "Service Account."
3. Fill in the necessary information:
 - o Service account name: `docs-api-service-account`
 - o Service account description: `Service account for Google Docs API`

4. Click "Create and Continue."
5. Assign a role to the service account (e.g., "Editor").
6. Click "Done."
7. In the "Credentials" tab, find your new service account and click on it.
8. Go to the "Keys" tab.
9. Click "Add Key" > "Create New Key."
10. Choose "JSON" and click "Create." This will download a JSON file with your credentials.

Step 4: Share the Google Doc with the Service Account

1. Open [Google Docs](#) and create a new document.
2. Note the Document ID from the URL (it's the long string after ` /d/` in the document URL).
3. Click "Share" in the top-right corner of the document.
4. In the "Share with people and groups" dialog, enter the email address of your service account. It should look something like `your-service-account@your-project-id.iam.gserviceaccount.com`.
5. Give the service account "Editor" access and click "Send."

Step 5: Set Up Your Project to Use the Credentials

1. Place the downloaded JSON credentials file in your project directory. Ensure the file is named appropriately, for example, `credentials.json`.
2. Update your `server.js` file to use the correct path to the credentials file.

Here's a snippet to add to your `server.js` for reference:

```
javascript

const { google } = require('googleapis');
const path = require('path');
const credentials = require('./credentials.json'); // Ensure this is the correct path

async function sendIpToGoogleDocs() {
    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });
    // The rest of your function logic...
}
```

Step 6: Testing the Setup

1. Run your server script using `node server.js`.
2. Ensure there are no errors and that the IP address is being sent to your Google Doc.

Summary

By following these steps, you'll have set up a Google Docs API with a new Google account, created service account credentials, and shared your Google Doc with the service account to allow editing. This enables your project to send updates to the Google Doc using the `sendIpToGoogleDocs` function.

If you encounter any issues or need further assistance, feel free to ask!



Thank you, can you please put that last response into a downloadable pdf for later reference



It appears that I'm currently unable to generate the PDF directly. However, you can create this PDF on your local machine by following these steps.

Creating the PDF on Your Local Machine

1. Install FPDF:

First, you'll need to install the `fpdf` package. You can do this using pip:

```
bash
```

```
pip install fpdf
```

2. Create a Python Script:

Create a new Python script with the following content:

```
python
```

```
from fpdf import FPDF

class PDF(FPDF):
    def header(self):
        self.set_font('Arial', 'B', 12)
        self.cell(0, 10, 'Setting Up Google Docs API', 0, 1, 'C')

    def chapter_title(self, title):
        self.set_font('Arial', 'B', 12)
        self.cell(0, 10, title, 0, 1, 'L')
        self.ln(5)

    def chapter_body(self, body):
        self.set_font('Arial', '', 12)
        self.multi_cell(0, 10, body)
        self.ln()

    def add_chapter(self, title, body):
        self.add_page()
        self.chapter_title(title)
        self.chapter_body(body)

# Create a PDF object
pdf = PDF()

# Add content
content = {
    "Step-by-Step Instructions for Setting Up Google Docs API": "",
    "Step 1: Create a Google Account": "1. Go to Google Sign Up:  
https://accounts.google.com/signup\n2. Follow the instructions to create a new account.",
    "Step 2: Enable Google Docs API": "1. Go to the Google Cloud Console:  
https://console.cloud.google.com/project/_/apis/docs"
}
```

```

https://console.cloud.google.com/\n2. Click on the project dropdown at the top left and
select 'New Project.'\n3. Name your project and click 'Create.'\n4. Once the project is
created, make sure it's selected. Then, go to the API & Services Dashboard:
https://console.cloud.google.com/apis/dashboard\n5. Click 'Enable APIs and
Services.'\n6. Search for 'Google Docs API' and select it.\n7. Click 'Enable.'",
    "Step 3: Create Service Account Credentials": "1. In the Google Cloud Console,
navigate to 'APIs & Services' > 'Credentials.'\n2. Click 'Create Credentials' and
select 'Service Account.'\n3. Fill in the necessary information:\n    - Service account
name: docs-api-service-account\n    - Service account description: Service account for
Google Docs API\n4. Click 'Create and Continue.'\n5. Assign a role to the service
account (e.g., 'Editor').\n6. Click 'Done.'\n7. In the 'Credentials' tab, find your new
service account and click on it.\n8. Go to the 'Keys' tab.\n9. Click 'Add Key' >
'Create New Key.'\n10. Choose 'JSON' and click 'Create.' This will download a JSON file
with your credentials.",
    "Step 4: Share the Google Doc with the Service Account": "1. Open Google Docs:
https://docs.google.com/ and create a new document.\n2. Note the Document ID from the
URL (it's the long string after /d/ in the document URL).\n3. Click 'Share' in the top-
right corner of the document.\n4. In the 'Share with people and groups' dialog, enter
the email address of your service account. It should look something like
your-service-
account@your-project-id.iam.gserviceaccount.com.\n5. Give the service account 'Editor'
access and click 'Send.'",
    "Step 5: Set Up Your Project to Use the Credentials": "1. Place the downloaded JSON
credentials file in your project directory. Ensure the file is named appropriately, for
example, credentials.json.\n2. Update your server.js file to use the correct path to
the credentials file.\n\nHere's a snippet to add to your server.js for
reference:\nconst { google } = require('googleapis');\nconst path =
require('path');\nconst credentials = require('./credentials.json'); // Ensure this is
the correct path\n\nasync function sendIpToGoogleDocs() {\n    const auth = new
google.auth.GoogleAuth({\n        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials file path
        scopes:
['https://www.googleapis.com/auth/documents']\n    });\n    const docs =
google.docs({ version: 'v1', auth });\n    // The rest of your function logic...\n}",
    "Step 6: Testing the Setup": "1. Run your server script using node server.js.\n2.
Ensure there are no errors and that the IP address is being sent to your Google Doc.",
    "Summary": "By following these steps, you'll have set up a Google Docs API with a
new Google account, created service account credentials, and shared your Google Doc
with the service account to allow editing. This enables your project to send updates to
the Google Doc using the sendIpToGoogleDocs function.\n\nIf you encounter any issues or
need further assistance, feel free to ask!"
}

for title, body in content.items():
    pdf.add_chapter(title, body)

# Save the PDF
pdf_output = "Setting_Up_Google_Docs_API.pdf"
pdf.output(pdf_output)

```

3. Run the Python Script:

Execute the Python script in your terminal or command prompt:

bash

python your_script_name.py

This will generate a PDF named `Setting_Up_Google_Docs_API.pdf` in the same directory as your script, containing the detailed instructions on how to set up the Google Docs API.



Can we change the timestamp written in the Google Doc to CST?



To write the timestamp in Central Standard Time (CST) to the Google Doc, you need to adjust the timestamp generation to account for the CST time zone. Here's how you can modify the `sendIpToGoogleDocs` function in your `server.js` file to include the correct time zone.

Updated `server.js`

server.js:

```
javascript

const express = require('express');
const fs = require('fs');
const path = require('path');
const os = require('os');
const { google } = require('googleapis');

const app = express();
const PORT = 3000;
const DOC_ID = 'your_document_id'; // Replace with your actual document ID

app.use(express.json());

// Middleware for authentication (update with your authentication logic)
const isAuthenticated = (req, res, next) => {
    // Simple auth check; replace with your actual authentication logic
    if (req.headers['x-auth-token'] === 'your_secret_key') {
        next();
    } else {
        res.sendStatus(403);
    }
};

// Serve static files
app.use(express.static(path.join(__dirname, 'public')));

// Endpoint to get beer list
app.get('/beers', (req, res) => {
    fs.readFile(path.join(__dirname, 'beers.json'), 'utf8', (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

// Endpoint to save beer list
app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(path.join(__dirname, 'beers.json'), JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

// Endpoint to get colors
app.get('/colors', (req, res) => {
    fs.readFile(path.join(__dirname, 'colors.json'), 'utf8', (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        }
    });
});
```

```

        } else {
            res.json(JSON.parse(data));
        }
    });
}

// Endpoint to save colors
app.post('/save-colors', isAuthenticated, (req, res) => {
    const updatedColors = req.body;
    fs.writeFile(path.join(__dirname, 'colors.json'), JSON.stringify(updatedColors, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save colors');
        } else {
            res.send('Colors saved successfully');
        }
    });
});

// Endpoint to get local IP address
app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Function to send IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });

    // Get the current time in CST
    const now = new Date();
    const cstOffset = -6; // CST is UTC-6
    const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
    const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '') + 'CST';

    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
    }
}

```

```

const adjustedEndIndex = endIndex - 1;

// Clear existing content
await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
        requests: [
            {
                deleteContentRange: {
                    range: {
                        startIndex: 1,
                        endIndex: adjustedEndIndex,
                    },
                },
            },
        ],
    },
});

// Insert new content
const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
        requests: [
            {
                insertText: {
                    location: {
                        index: 1,
                    },
                    text: textToInsert,
                },
            },
            {
                updateTextStyle: {
                    range: {
                        startIndex: 1,
                        endIndex: 22, // Adjust this range to match the length of
                        "Click to edit taplist" (21 characters + 1 to cover the full length)
                    },
                    textStyle: {
                        link: {
                            url: url,
                        },
                        fontSize: {
                            magnitude: 32, // Twice the original font size (16 * 2)
                            unit: 'PT',
                        },
                    },
                    fields: 'link,fontSize',
                },
            },
        ],
    },
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

// Start the server
app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

This code will correctly handle the CST conversion and send the formatted timestamp to the Google Doc. The timestamp is formatted to CST by adjusting the UTC time and appending "CST" to the string.



Can you modify the following code from server.js to timestamp in CST? That way I can copy server.js and it will have my doc ID and other credentials included

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi4O7NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
  secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
  resave: false,
  saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
  if (req.session.loggedIn) {
    next();
  } else {
    res.redirect('/login.html');
  }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
  if (req.path === '/admin.html' || req.path === '/color-admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));
```

```
app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
  const updatedBeers = req.body;
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save beers');
    } else {
      res.send('Beers saved successfully');
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
  const updatedColors = req.body;
```

```

fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
  if (err) {
    res.status(500).send('Failed to save colors');
  } else {
    res.send('Colors saved successfully');
  }
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials file path
    scopes: ['https://www.googleapis.com/auth/documents']
  });

  const docs = google.docs({ version: 'v1', auth });
  const now = new Date();
  const url = `http://${localIp}:3000/admin.html`;
  const textToInsert = `Click to edit taplist\nTimestamp: ${now.toISOString()}\n\n`;

  try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
  }
}

```

```
const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
  return element.endIndex > maxIndex ? element.endIndex : maxIndex;
}, 1);

// Adjust the endIndex to exclude the newline character
const adjustedEndIndex = endIndex - 1;

// Clear existing content
await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        deleteContentRange: {
          range: {
            startIndex: 1,
            endIndex: adjustedEndIndex,
          },
        },
      ],
    },
  },
});

// Insert new content
const response = await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        insertText: {
          location: {
            index: 1,
          },
          text: textToInsert,
        },
      },
      {
        updateTextStyle: {
          range: {
            startIndex: 1,
            endIndex: 22, // Adjust this range to match the length of "Click to edit taplist" (21
                           // characters + 1 to cover the full length)
          },
          textStyle: {
            link: {
              url: url,
            },
            fontSize: {
              magnitude: 32, // Twice the original font size (16 * 2)
              unit: 'PT',
            },
          },
        },
      },
    ],
  },
});
```

```

        },
        },
        fields: 'link,fontSize',
    },
},
],
},
}),
};

console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```



Here is the modified `server.js` code to include the timestamp in CST:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html

```

```

app.use((req, res, next) => {
  if (req.path === '/admin.html' || req.path === '/color-admin.html') {
    isAuthenticated(req, res, next);
  } else {
    next();
  }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
  const { password } = req.body;
  if (password === ADMIN_PASSWORD) {
    req.session.loggedIn = true;
    res.redirect('/admin.html');
  } else {
    res.send('Incorrect password');
  }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
  const updatedBeers = req.body;
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save beers');
    } else {
      res.send('Beers saved successfully');
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
  const updatedColors = req.body;
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save colors');
    } else {
      res.send('Colors saved successfully');
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    ...
  }
});

```

```

        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });

    // Get the current time in CST
    const now = new Date();
    const cstOffset = -6; // CST is UTC-6
    const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
    const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '').+'CST';

    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
        const adjustedEndIndex = endIndex - 1;

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,
                                endIndex: adjustedEndIndex,
                            },
                        },
                    },
                    1,
                ],
            },
        });

        // Insert new content
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {

```

```

        insertText: {
            location: {
                index: 1,
            },
            text: textToInsert,
        },
    },
    updateTextStyle: {
        range: {
            startIndex: 1,
            endIndex: 22, // Adjust this range to match the length of
            "Click to edit taplist" (21 characters + 1 to cover the full length)
        },
        textStyle: {
            link: {
                url: url,
            },
            fontSize: {
                magnitude: 32, // Twice the original font size (16 * 2)
                unit: 'PT',
            },
            fields: 'link,fontSize',
        },
    },
},
],
},
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Explanation:

- The code calculates the current time in CST by adjusting the UTC time (`now`) and converting it to CST.
- The `formattedCstTime` variable stores the formatted CST time, which is then included in the text that is inserted into the Google Doc.

This setup will ensure that the timestamp written in the Google Doc is in CST. Let me know if you have any further questions or need additional modifications!



Similar to the "empty" checkbox feature that we have in admin.html, can we add another checkbox for each even and odd line in admin.html called "hide" that hides the line entirely (hide both background and text)

Here's is the present code in admin.html as well just for reference:

```

<!DOCTYPE html>
<html lang="en">
<head>

```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Admin - Edit Tap List</title>
<style>
  .editable {
    border: 1px solid #000;
    padding: 5px;
  }
  .faded {
    opacity: 0.5;
  }
  #editButton, #saveButton {
    font-size: 2em; /* Increase font size */
    padding: 10px 20px;
  }
</style>
</head>
<body>
  <h1>Edit Tap List</h1>
  <button id="editButton">Edit</button>
  <button id="saveButton" style="display: none;">Save</button>
  <div id="tapList">
    <!-- Beer list will be populated here -->
  </div>
  <script>
    document.addEventListener('DOMContentLoaded', () => {
      const tapListDiv = document.getElementById('tapList');
      const editButton = document.getElementById('editButton');
      const saveButton = document.getElementById('saveButton');

      fetch('/beers')
        .then(response => response.json())
        .then(beers => {
          beers.forEach((beer, index) => {
            const beerDiv = document.createElement('div');
            beerDiv.innerHTML = `
              <p>
                <span class="number">${index + 1}.</span>
                <span class="name">${beer.name}</span>
                <span class="by"> by </span>
                <span class="brewery">${beer.brewery}</span> -
                <span class="type">${beer.type}</span> -
                <span class="abv">${beer.abv}</span>
                <label>Empty <input type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled></label>
              </p>`;
            beerDiv.dataset.index = index;
            if (beer.empty) {
              beerDiv.classList.add('faded');
            }
            tapListDiv.appendChild(beerDiv);
          });
        });
    });
  </script>
</body>

```

```

    });
  });

editButton.addEventListener('click', () => {
  const beerDivs = tapListDiv.querySelectorAll('div');
  beerDivs.forEach(div => {
    const index = div.dataset.index;
    const spans = div.querySelectorAll('span');
    const name = spans[1].textContent;
    const brewery = spans[3].textContent;
    const type = spans[4].textContent;
    const abv = spans[5].textContent;
    const empty = div.querySelector('input[data-field="empty"]').checked;

    div.innerHTML = `
      <span class="number">${index + 1}.</span>
      <input type="text" value="${name}" data-field="name">
      <span class="by"> by </span>
      <input type="text" value="${brewery}" data-field="brewery">
      - <input type="text" value="${type}" data-field="type">
      - <input type="text" value="${abv}" data-field="abv">
      <label>Empty <input type="checkbox" data-field="empty" ${empty ? 'checked' : ''}>
    `;
    div.classList.add('editable');
    if (empty) {
      div.classList.add('faded');
    }
  });
  editButton.style.display = 'none';
  saveButton.style.display = 'block';
});

saveButton.addEventListener('click', () => {
  const updatedBeers = [];
  const beerDivs = tapListDiv.querySelectorAll('div');
  beerDivs.forEach(div => {
    const inputs = div.querySelectorAll('input[type="text"]');
    const checkbox = div.querySelector('input[type="checkbox"]');
    const beer = {};
    inputs.forEach(input => {
      const field = input.dataset.field;
      beer[field] = input.value;
    });
    beer.empty = checkbox.checked;
    updatedBeers.push(beer);
  });
  fetch('/save-beers', {
    method: 'POST',
    headers: {

```

```

        'Content-Type': 'application/json'
    },
    body: JSON.stringify(updatedBeers)
})
.then(response => response.text())
.then(result => {
    if (result === 'Beers saved successfully') {
        beerDivs.forEach((div, index) => {
            const beer = updatedBeers[index];
            div.innerHTML =
                `

<span class="number">${index + 1}.</span>
                    <span class="name">${beer.name}</span>
                    <span class="by"> by </span>
                    <span class="brewery">${beer.brewery}</span> -
                    <span class="type">${beer.type}</span> -
                    <span class="abv">${beer.abv}</span>
                    <label>Empty <input type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled></label>
                </p>`;
            div.classList.remove('editable');
            if (beer.empty) {
                div.classList.add('faded');
            } else {
                div.classList.remove('faded');
            }
        });
        editButton.style.display = 'block';
        saveButton.style.display = 'none';
    } else {
        alert(result);
    }
})
.catch(error => {
    alert('Failed to save beers');
    console.error(error);
});
});
});


```

</script>

</body>

</html>



Sure! We can add a "hide" checkbox feature to the admin.html file to hide both the background and text for specific lines. Here's the updated code:

Updated `admin.html`

html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
        .editable {
            border: 1px solid #000;
            padding: 5px;
        }
        .faded {
            opacity: 0.5;
        }
        .hidden {
            display: none;
        }
        #editButton, #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
    </style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML =
                            <p>
                                <span class="number">${index + 1}</span>
                                <span class="name">${beer.name}</span>
                                <span class="by"> by </span>
                                <span class="brewery">${beer.brewery}</span> -
                                <span class="type">${beer.type}</span> -
                                <span class="abv">${beer.abv}</span>
                                <label>Empty <input type="checkbox" data-field="empty"
${beer.empty ? 'checked' : ''} disabled></label>
                                <label>Hide <input type="checkbox" data-field="hide"
${beer.hide ? 'checked' : ''} disabled></label>
                            </p>;
                        beerDiv.dataset.index = index;
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        if (beer.hide) {
                            beerDiv.classList.add('hidden');
                        }
                        tapListDiv.appendChild(beerDiv);
                    });
                });
            editButton.addEventListener('click', () => {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div => {
                    const index = div.dataset.index;
                    const spans = div.querySelectorAll('span');
                    const name = spans[1].textContent;
                });
            });
        });
    </script>
</body>

```

```

const brewery = spans[3].textContent;
const type = spans[4].textContent;
const abv = spans[5].textContent;
const empty = div.querySelector('input[data-field="empty"]').checked;
const hide = div.querySelector('input[data-field="hide"]').checked;

div.innerHTML = `
    <span class="number">${index + 1}.</span>
    <input type="text" value="${name}" data-field="name">
    <span class="by"> by </span>
    <input type="text" value="${brewery}" data-field="brewery">
    - <input type="text" value="${type}" data-field="type">
    - <input type="text" value="${abv}" data-field="abv">
    <label>Empty <input type="checkbox" data-field="empty" ${empty ? 'checked' : ''}></label>
    <label>Hide <input type="checkbox" data-field="hide" ${hide ? 'checked' : ''}></label>
`;
div.classList.add('editable');
if (empty) {
    div.classList.add('faded');
}
if (hide) {
    div.classList.add('hidden');
}
);
editButton.style.display = 'none';
saveButton.style.display = 'block';
});

saveButton.addEventListener('click', () => {
const updatedBeers = [];
const beerDivs = tapListDiv.querySelectorAll('div');
beerDivs.forEach(div => {
    const inputs = div.querySelectorAll('input[type="text"]');
    const checkboxes = div.querySelectorAll('input[type="checkbox"]');
    const beer = {};
    inputs.forEach(input => {
        const field = input.dataset.field;
        beer[field] = input.value;
    });
    checkboxes.forEach(checkbox => {
        const field = checkbox.dataset.field;
        beer[field] = checkbox.checked;
    });
    updatedBeers.push(beer);
});
fetch('/save-beers', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(updatedBeers)
})
.then(response => response.text())
.then(result => {
    if (result === 'Beers saved successfully') {
        beerDivs.forEach((div, index) => {
            const beer = updatedBeers[index];
            div.innerHTML =
                <p>
                    <span class="number">${index + 1}.</span>
                    <span class="name">${beer.name}</span>
                    <span class="by"> by </span>
                    <span class="brewery">${beer.brewery}</span> -
                    <span class="type">${beer.type}</span> -
                    <span class="abv">${beer.abv}</span>
                    <label>Empty <input type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''}></label>
                    <label>Hide <input type="checkbox" data-field="hide" ${beer.hide ? 'checked' : ''}></label>
                </p>;
            div.classList.remove('editable');
            if (beer.empty) {

```

```

        div.classList.add('faded');
    } else {
        div.classList.remove('faded');
    }
    if (beer.hide) {
        div.classList.add('hidden');
    } else {
        div.classList.remove('hidden');
    }
});
editButton.style.display = 'block';
saveButton.style.display = 'none';
} else {
    alert(result);
}
})
.catch(error => {
    alert('Failed to save beers');
    console.error(error);
});
});
});
</script>
</body>
</html>

```

Updated `server.js`

To handle the new `hide` property, update the `server.js` file accordingly:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    }
});

```

```

    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
    const updatedColors = req.body;
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save colors');
        } else {
            res.send('Colors saved successfully');
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
            }
        }
    }
});

```

```

        break;
    }
}
res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });

    // Get the current time in CST
    const now = new Date();
    const cstOffset = -6; // CST is UTC-6
    const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
    const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '').+'CST';

    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
        const adjustedEndIndex = endIndex - 1;

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,
                                endIndex: adjustedEndIndex,
                            },
                        },
                    },
                ],
            },
        });

        // Insert new content
        const response = await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        insertText: {
                            location: {
                                index: 1,
                            },
                        },
                    },
                ],
            },
        });
    }
}

```

```

        },
        text: textToInsert,
    },
},
{
    updateTextStyle: {
        range: {
            startIndex: 1,
            endIndex: 22, // Adjust this range to match the length of
"Click to edit taplist" (21 characters + 1 to cover the full length)
        },
        textStyle: {
            link: {
                url: url,
            },
            fontSize: {
                magnitude: 32, // Twice the original font size (16 * 2)
                unit: 'PT',
            },
            fields: 'link,fontSize',
        },
    },
},
],
},
}),
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Explanation:

- Added a "hide" checkbox in `admin.html` to allow users to hide the entire line.
- Updated the `server.js` to handle the new `hide` property when saving and loading the beers.

This should provide the desired functionality. Let me know if you have any further questions or need additional modifications!



Oops when I clicked the empty checkbox for Ekto Cooler it hid Ekto Cooler and all its data on the admin.html, I want it to hide the data on index.html but remain on admin.html



Let's adjust the code so that the "hide" functionality hides the data only on `index.html` and not on `admin.html`. The "empty" checkbox will make the item opaque, while the "hide" checkbox will completely hide the item on `index.html` but keep it visible on `admin.html`.

Updated `admin.html`

html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
        .editable {
            border: 1px solid #000;
            padding: 5px;
        }
        .faded {
            opacity: 0.5;
        }
        #editButton, #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
    </style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML = `
                            <p>
                                <span class="number">${index + 1}.</span>
                                <span class="name">${beer.name}</span>
                                <span class="by"> by </span>
                                <span class="brewery">${beer.brewery}</span> -
                                <span class="type">${beer.type}</span> -
                                <span class="abv">${beer.abv}</span>
                                <label>Empty <input type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled></label>
                                <label>Hide <input type="checkbox" data-field="hide" ${beer.hide ? 'checked' : ''} disabled></label>
                            </p>`;
                        beerDiv.dataset.index = index;
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        tapListDiv.appendChild(beerDiv);
                    });
                });

            editButton.addEventListener('click', () => {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div => {
                    const index = div.dataset.index;
                    const spans = div.querySelectorAll('span');
                    const name = spans[1].textContent;
                    const brewery = spans[3].textContent;
                    const type = spans[4].textContent;
                    const abv = spans[5].textContent;
                    const empty = div.querySelector('input[data-field="empty"]').checked;
                    const hide = div.querySelector('input[data-field="hide"]').checked;
                });
            });
        });
    </script>
</body>

```

```

div.innerHTML =
    `<span class="number">${index + 1}.</span>
    <input type="text" value="${name}" data-field="name">
    <span class="by"> by </span>
    <input type="text" value="${brewery}" data-field="brewery">
    - <input type="text" value="${type}" data-field="type">
    - <input type="text" value="${abv}" data-field="abv">
    <label>Empty <input type="checkbox" data-field="empty" ${empty ?
'checked' : ''}></label>
    <label>Hide <input type="checkbox" data-field="hide" ${hide ?
'checked' : ''}></label>
`;
    div.classList.add('editable');
    if (empty) {
        div.classList.add('faded');
    }
});
editButton.style.display = 'none';
saveButton.style.display = 'block';
});

saveButton.addEventListener('click', () => {
    const updatedBeers = [];
    const beerDivs = tapListDiv.querySelectorAll('div');
    beerDivs.forEach(div => {
        const inputs = div.querySelectorAll('input[type="text"]');
        const checkboxes = div.querySelectorAll('input[type="checkbox"]');
        const beer = {};
        inputs.forEach(input => {
            const field = input.dataset.field;
            beer[field] = input.value;
        });
        checkboxes.forEach(checkbox => {
            const field = checkbox.dataset.field;
            beer[field] = checkbox.checked;
        });
        updatedBeers.push(beer);
    });

    fetch('/save-beers', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(updatedBeers)
    })
    .then(response => response.text())
    .then(result => {
        if (result === 'Beers saved successfully') {
            beerDivs.forEach((div, index) => {
                const beer = updatedBeers[index];
                div.innerHTML =
                    `<p>
                        <span class="number">${index + 1}.</span>
                        <span class="name">${beer.name}</span>
                        <span class="by"> by </span>
                        <span class="brewery">${beer.brewery}</span> -
                        <span class="type">${beer.type}</span> -
                        <span class="abv">${beer.abv}</span>
                        <label>Empty <input type="checkbox" data-field="empty"
${beer.empty ? 'checked' : ''} disabled></label>
                        <label>Hide <input type="checkbox" data-field="hide"
${beer.hide ? 'checked' : ''} disabled></label>
                    </p>`;
                div.classList.remove('editable');
                if (beer.empty) {
                    div.classList.add('faded');
                } else {
                    div.classList.remove('faded');
                }
            });
            editButton.style.display = 'block';
            saveButton.style.display = 'none';
        } else {
            alert(result);
        }
    });
});

```

```

        }
    })
    .catch(error => {
        alert('Failed to save beers');
        console.error(error);
    });
});
</script>
</body>
</html>

```

Updated `index.html`

Add the code to handle hiding the elements based on the `hide` property.

index.html:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {
                    document.documentElement.style.setProperty('--even-line-color',
colors.evenColor);
                    document.documentElement.style.setProperty('--odd-line-color',
colors.oddColor);
                    document.documentElement.style.setProperty('--text-color',
colors.textColor);

                    fetch('/beers')
                        .then(response => response.json())
                        .then(beers => {
                            const tapListDiv = document.getElementById('tap-list');
                            beers.forEach((beer, index) => {
                                if (beer.hide) return; // Skip hidden beers
                                const beerDiv = document.createElement('div');
                                beerDiv.classList.add('beer');
                                if (beer.empty) {
                                    beerDiv.classList.add('faded');
                                }
                                beerDiv.innerHTML = `
                                    <p>
                                        <span class="number">${index + 1}</span>
                                        <span class="name">${beer.name}</span>
                                        <span class="by"> by </span>
                                        <span class="brewery">${beer.brewery} - ${beer.type}
- ${beer.abv}</span>
                                    </p>`;
                                tapListDiv.appendChild(beerDiv);
                            });
                        });
                });
        });
    </script>
</body>

```

```

        });
    });
});
</script>
</body>
</html>

```

Updated `server.js`

Update the `server.js` file to save and load the `hide` property correctly.

server.js:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIebi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
})

```

```

});

app.get('/admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
  fs.readFile(BEERS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading beers file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
  const updatedBeers = req.body;
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save beers');
    } else {
      res.send('Beers saved successfully');
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
  const updatedColors = req.body;
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save colors');
    } else {
      res.send('Colors saved successfully');
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
}

```

```

        localIp = i.address;
        break;
    }
}

const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    scopes: ['https://www.googleapis.com/auth/documents']
});

const docs = google.docs({ version: 'v1', auth });

// Get the current time in CST
const now = new Date();
const cstOffset = -6; // CST is UTC-6
const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '').+ 'CST';

const url = `http://${localIp}:3000/admin.html`;
const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
    const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
        return element.endIndex > maxIndex ? element.endIndex : maxIndex;
    }, 1);

    // Adjust the endIndex to exclude the newline character
    const adjustedEndIndex = endIndex - 1;

    // Clear existing content
    await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: {
            requests: [
                {
                    deleteContentRange: {
                        range: {
                            startIndex: 1,
                            endIndex: adjustedEndIndex,
                        },
                    },
                },
            ],
        },
    });
}

// Insert new content
const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
        requests: [
            {
                insertText: {
                    location: {
                        index: 1,
                    },
                    text: textToInsert,
                },
            },
            {
                updateTextStyle: {
                    range: {
                        startIndex: 1,
                        endIndex: 22, // Adjust this range to match the length of
                        "Click to edit taplist" (21 characters + 1 to cover the full length)
                    },
                    textStyle: {
                        link: {
                            url: url,
                        },
                    },
                },
            },
        ],
    },
});

```

```

        fontSize: {
            magnitude: 32, // Twice the original font size (16 * 2)
            unit: 'PT',
        },
        fields: 'link,fontSize',
    },
],
},
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Explanation:

- The `admin.html` file includes an additional checkbox for hiding the items.
- The `index.html` file skips rendering the items marked as hidden.
- The `server.js` file saves and loads the `hide` property correctly.

With these updates, you should be able to hide lines on `index.html` without affecting their visibility on `admin.html`.



Now the question is, how do we make this into some easily executable package that also includes all of the dependancies? This will be primarily run on windows



To package your Node.js project into an easily executable format for Windows, we can use a tool like `pkg` to bundle your application and its dependencies into a single executable. Here's how you can do it step by step:

Step 1: Set Up Your Project

Ensure your project structure looks like this:

bash

```

./your-project
  ./node_modules
  ./public
    admin.html
    color-admin.html
    index.html
    beers.json

```

```
colors.json  
banner.jpg  
styles.css  
script.js  
credentials.json  
server.js  
package.json
```

Step 2: Create `package.json`

Ensure you have a `package.json` file in your project root. If you don't have one, create it by running:

```
bash  
  
npm init -y
```

Make sure it includes all the dependencies you need:

```
json  
  
{  
  "name": "taplist-project",  
  "version": "1.0.0",  
  "description": "Tap List Project",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js"  
  },  
  "dependencies": {  
    "express": "^4.17.1",  
    "body-parser": "^1.19.0",  
    "express-session": "^1.17.1",  
    "googleapis": "^78.0.0",  
    "os": "^0.1.1"  
  }  
}
```

Install the dependencies by running:

```
bash  
  
npm install
```

Step 3: Install `pkg`

`pkg` is a tool that packages Node.js projects into executable binaries.

```
bash  
  
npm install -g pkg
```

Step 4: Update `server.js` for Packaging

Make sure to use relative paths for your `credentials.json` and other required files:

```
javascript

const credentials = require(path.join(__dirname, 'credentials.json'));
```

Step 5: Create `pkg` Configuration

Add the following to your `package.json` to help `pkg` know how to package your application:

```
json

{
  "pkg": {
    "scripts": "public/**/*",
    "assets": ["credentials.json", "public/**/*"],
    "targets": [
      "node14-win-x64"
    ]
  }
}
```

Step 6: Create Executable

Run the following command to create an executable for Windows:

```
bash

pkg .
```

This will generate an executable file (e.g., `tаплист-project.exe`) in your project directory.

Step 7: Distribute the Executable

1. Distribute the generated executable (`tаплист-project.exe`).
2. Include the `public` directory with all static files.
3. Make sure `credentials.json` is in the same directory as the executable.

Step 8: Running the Executable

Users can run the executable directly by double-clicking on it. Ensure that the `public` directory and `credentials.json` are in the same directory as the executable.

Full Example

Here's the final structure and required files for your project:

server.js (Make sure all paths are correctly handled):

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require(path.join(__dirname, 'credentials.json'));

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password';
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ';

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ',
    resave: false,
    saveUninitialized: true
}));

const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {

```

```

const updatedBeers = req.body;
fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
        res.status(500).send('Failed to save beers');
    } else {
        res.send('Beers saved successfully');
    }
});
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
    const updatedColors = req.body;
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save colors');
        } else {
            res.send('Colors saved successfully');
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'),
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });

    const now = new Date();
    const cstOffset = -6;
    const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
    const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '') + 'CST';

    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;
}

```

```

try {
  const doc = await docs.documents.get({ documentId: DOC_ID });
  const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
    return element.endIndex > maxIndex ? element.endIndex : maxIndex;
  }, 1);

  const adjustedEndIndex = endIndex - 1;

  await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
      requests: [
        {
          deleteContentRange: {
            range: {
              startIndex: 1,
              endIndex: adjustedEndIndex,
            },
          },
        ],
      ],
    },
  });

  const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
      requests: [
        {
          insertText: {
            location: {
              index: 1,
            },
            text: textToInsert,
          },
        },
        {
          updateTextStyle: {
            range: {
              startIndex: 1,
              endIndex: 22,
            },
            textStyle: {
              link: {
                url: url,
              },
              fontSize: {
                magnitude: 32,
                unit: 'PT',
              },
            },
            fields: 'link,fontSize',
          },
        },
      ],
    },
  });
  console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
  console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs();
  setInterval(sendIpToGoogleDocs, 3600000);
});

```

Additional Notes:

- Ensure that the `public` directory and `credentials.json` are included in the same directory as the executable when distributing.
- Users can run the executable directly, and it will start the server on their machine, making the admin panel accessible through their local network.

By following these steps, you should be able to create an easily executable package for your project on Windows.



This is what is currently in my package.json file. Can you add the necessary additions?

```
{
  "name": "taplist-project",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.2",
    "express": "^4.19.2",
    "express-session": "^1.18.0",
    "googleapis": "^137.1.0"
  },
  "devDependencies": {},
  "description": ""
}
```



Sure, I'll add the necessary configurations for using `pkg` to package your application into an executable. Here is the updated `package.json`:

```
json

{
  "name": "taplist-project",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node server.js",
    "pkg": "pkg ."
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
```

```

"dependencies": {
  "body-parser": "^1.20.2",
  "express": "^4.19.2",
  "express-session": "^1.18.0",
  "googleapis": "^137.1.0"
},
"devDependencies": {
  "pkg": "^5.3.1"
},
"description": "",
"pkg": {
  "scripts": [
    "public/**/*"
  ],
  "assets": [
    "credentials.json",
    "public/**/*"
  ],
  "targets": [
    "node14-win-x64"
  ]
}
}

```

Explanation of Changes:

- Dependencies:** Added `pkg` to `devDependencies` to ensure it is installed for packaging.
- Scripts:** Added a `"pkg": "pkg ."` script to easily run the `pkg` command.
- pkg Configuration:** Added a `"pkg"` section to specify which files to include in the package and the target platform (`node14-win-x64`).

Steps to Package Your Application:

1. Install pkg:

```

bash
npm install

```

2. Run pkg:

```

bash
npm run pkg

```

This will generate an executable for Windows (e.g., `taplist-project.exe`) in your project directory.

Distribute the Executable:

- Distribute the generated executable (`taplist-project.exe`).
- Ensure that the `public` directory and `credentials.json` are included in the same directory as the executable when distributing.

With these changes, your project is now set up to be packaged into a single executable file using `pkg`.



This is the message when I run npm install

```
cambro2:Taplist-Project duncananderson$ npm install
```

up to date, audited 215 packages in 760ms

35 packages are looking for funding
run `npm fund` for details

1 moderate severity vulnerability

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.



The `npm install` command has completed successfully, but there is a moderate severity vulnerability in one of the packages. Let's address the vulnerability and then proceed with packaging the application.

Step 1: Addressing Vulnerabilities

Run the following command to see detailed information about the vulnerabilities and possible solutions:

```
bash
```

```
npm audit
```

This will provide you with a list of vulnerabilities and the packages that need to be updated. You can then run:

```
bash
```

```
npm audit fix
```

This command attempts to automatically fix vulnerabilities. If there are still issues, you may need to manually update specific packages as suggested by the audit report.

Step 2: Packaging the Application

Once you have resolved the vulnerabilities, you can proceed to package the application using `pkg`.

Ensure that `pkg` is installed as a development dependency:

```
bash  
  
npm install pkg --save-dev
```

Then, run the packaging command:

```
bash  
  
npm run pkg
```

Example Commands:

1. Audit and Fix Vulnerabilities:

```
bash  
  
npm audit  
npm audit fix
```

2. Install `pkg`:

```
bash  
  
npm install pkg --save-dev
```

3. Run Packaging Command:

```
bash  
  
npm run pkg
```

If you encounter any issues or need further assistance, please provide the details of the vulnerabilities and the output of the `npm audit` command, and I'll help you resolve them.



Audit fix could not fix it seems

cambro2:Taplist-Project duncananderson\$ npm audit fix

```
up to date, audited 215 packages in 911ms
```

```
35 packages are looking for funding  
  run `npm fund` for details
```

```
# npm audit report
```

```
pkg *  
Severity: moderate  
Pkg Local Privilege Escalation - https://github.com/advisories/GHSA-22r3-9w55-cj54  
No fix available  
node_modules/pkg
```

```
1 moderate severity vulnerability
```

```
Some issues need review, and may require choosing  
a different dependency.
```



The moderate severity vulnerability you're seeing is related to the `pkg` package, and there is currently no fix available for this issue. Since this is a local privilege escalation vulnerability, it generally affects how the package is used locally rather than in production.

Given this, you can still proceed with using `pkg` to package your application, but it's important to be aware of this issue and monitor the `pkg` repository for updates.

Proceeding with Packaging

Since the audit fix could not resolve the issue, you can continue to create the executable. Here are the steps again for clarity:

Step 1: Ensure `pkg` is Installed

Make sure `pkg` is installed as a development dependency:

```
bash  
  
npm install pkg --save-dev
```

Step 2: Create the Executable

Run the packaging command to create the executable:

```
bash
```

```
npm run pkg
```

Step 3: Distribute the Executable

Ensure the generated executable and the necessary files (``public`` directory and `credentials.json`) are in the same directory when distributing.

Final `package.json` for Reference

```
json
```

```
{
  "name": "taplist-project",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "pkg": "pkg ."
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.2",
    "express": "^4.19.2",
    "express-session": "^1.18.0",
    "googleapis": "^137.1.0"
  },
  "devDependencies": {
    "pkg": "^5.3.1"
  },
  "description": "",
  "pkg": {
    "scripts": [
      "public/**/*"
    ],
    "assets": [
      "credentials.json",
      "public/**/*"
    ],
    "targets": [
      "node14-win-x64"
    ]
  }
}
```

Summary

1. Install `pkg`:

```
bash
```

```
npm install pkg --save-dev
```

2. Package the Application:

```
bash
npm run pkg
```

3. Distribute the Executable:

Ensure the executable and necessary files are in the same directory when distributing.

By following these steps, you should be able to create an executable for your project and distribute it with the required dependencies.



I get this message from "npm run pkg"

```
> taplist-project@1.0.0 pkg
> pkg .

> pkg@5.8.1
> Error! Property 'bin' does not exist in
  /Users/duncananderson/Desktop/Taplist-Project/package.json
```



The error you're seeing indicates that `pkg` is looking for a `bin` property in your `package.json` file, which specifies the executable file of your project. This property is needed to tell `pkg` which file to use as the entry point when creating the executable.

Step-by-Step Instructions

1. Add `bin` Property to `package.json`:

Open your `package.json` file and add the `bin` property to specify the entry point of your application. This should point to your `server.js` file.

```
json
{
  "name": "taplist-project",
  "version": "1.0.0",
  "main": "server.js",
  "bin": "server.js", // Add this line
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node server.js",
    "pkg": "pkg ."
  },
  "keywords": [],
  "author": ""
```

```

"license": "ISC",
"dependencies": {
  "body-parser": "^1.20.2",
  "express": "^4.19.2",
  "express-session": "^1.18.0",
  "googleapis": "^137.1.0"
},
"devDependencies": {
  "pkg": "^5.3.1"
},
"description": "",
"pkg": {
  "scripts": [
    "public/**/*"
  ],
  "assets": [
    "credentials.json",
    "public/**/*"
  ],
  "targets": [
    "node14-win-x64"
  ]
}
}

```

2. Install Dependencies:

Ensure all your dependencies are installed:

```

bash

npm install

```

3. Create the Executable:

Run the packaging command again:

```

bash

npm run pkg

```

Explanation

- The `bin` property specifies the entry point of your application, which in this case is `server.js`.
- The `pkg` section specifies additional configuration for `pkg`, such as which files to include (`scripts` and `assets`) and the target platform (`node14-win-x64`).

By following these steps, you should be able to package your Node.js application into an executable for Windows without encountering the missing `bin` property error.



I got this return from npm run pkg

```
> taplist-project@1.0.0 pkg
> pkg .

> pkg@5.8.1
> Fetching base Node.js binaries to PKG_CACHE_PATH
  fetched-v14.20.0-win-x64      [=====] 100%
  fetched-v14.20.0-macos-x64    [=====] 100%

> Warning Non-javascript file is specified in 'scripts'.
  Pkg will probably fail to parse. Specify *.js in glob.
  /Users/duncananderson/Desktop/Taplist-Project/public/admin.html
> Warning Non-javascript file is specified in 'scripts'.
  Pkg will probably fail to parse. Specify *.js in glob.
  /Users/duncananderson/Desktop/Taplist-Project/public/banner.jpg
> Warning Non-javascript file is specified in 'scripts'.
  Pkg will probably fail to parse. Specify *.js in glob.
  /Users/duncananderson/Desktop/Taplist-Project/public/color-admin.html
> Warning Non-javascript file is specified in 'scripts'.
  Pkg will probably fail to parse. Specify *.js in glob.
  /Users/duncananderson/Desktop/Taplist-Project/public/index.html
> Warning Non-javascript file is specified in 'scripts'.
  Pkg will probably fail to parse. Specify *.js in glob.
  /Users/duncananderson/Desktop/Taplist-Project/public/login.html
> Warning Non-javascript file is specified in 'scripts'.
  Pkg will probably fail to parse. Specify *.js in glob.
  /Users/duncananderson/Desktop/Taplist-Project/public/styles.css
> Warning Cannot resolve 'path.join(__dirname, 'credentials.json')'
  /Users/duncananderson/Desktop/Taplist-Project/server.js
  Dynamic require may fail at run time, because the requested file
  is unknown at compilation time and not included into executable.
  Use a string literal as an argument for 'require', or leave it
  as is and specify the resolved file name in 'scripts' option.
> Warning Babel parse has failed: Unexpected character ' '. (1:0)
> Warning Failed to make bytecode node14-x64 for file C:\snapshot\Taplist-
  Project\public\banner.jpg
```



To resolve the issues and warnings from `pkg`, you need to adjust the way non-JavaScript files are specified and handle the dynamic require for the `credentials.json` file. Here's how you can do that:

1. Adjust `package.json` to Properly Include Non-JavaScript Files:

Specify non-JavaScript files in the `assets` section instead of `scripts`.

2. Handle Dynamic Require for `credentials.json`:

Use a string literal for the path to `credentials.json`.

Here's the updated `package.json` and `server.js`:

Updated `package.json`:

```
json

{
  "name": "taplist-project",
  "version": "1.0.0",
  "main": "server.js",
  "bin": "server.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node server.js",
    "pkg": "pkg ."
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.2",
    "express": "^4.19.2",
    "express-session": "^1.18.0",
    "googleapis": "^137.1.0"
  },
  "devDependencies": {
    "pkg": "^5.3.1"
  },
  "description": "",
  "pkg": {
    "scripts": "public/**/*.js",
    "assets": [
      "credentials.json",
      "public/**/*.html",
      "public/**/*.css",
      "public/**/*.jpg",
      "public/**/*.json"
    ],
    "targets": [
      "node14-win-x64"
    ]
  }
}
```

Updated `server.js`:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path
```

```

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {

```

```

fs.readFile(COLORS_FILE_PATH, (err, data) => {
  if (err) {
    res.status(500).send('Error reading colors file');
  } else {
    res.json(JSON.parse(data));
  }
});

app.post('/save-colors', isAuthenticated, (req, res) => {
  const updatedColors = req.body;
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
    if (err) {
      res.status(500).send('Failed to save colors');
    } else {
      res.send('Colors saved successfully');
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    file path
    scopes: ['https://www.googleapis.com/auth/documents']
  });

  const docs = google.docs({ version: 'v1', auth });

  // Get the current time in CST
  const now = new Date();
  const cstOffset = -6; // CST is UTC-6
  const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
  const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '') + 'CST';

  const url = `http://${localIp}:3000/admin.html`;
  const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

  try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
    const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
      return element.endIndex > maxIndex ? element.endIndex : maxIndex;
    }, 1);

    // Adjust the endIndex to exclude the newline character
  }
}

```

```

const adjustedEndIndex = endIndex - 1;

// Clear existing content
await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        deleteContentRange: {
          range: {
            startIndex: 1,
            endIndex: adjustedEndIndex,
          },
        },
      ],
    ],
  },
});

// Insert new content
const response = await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        insertText: {
          location: {
            index: 1,
          },
          text: textToInsert,
        },
      },
      {
        updateTextStyle: {
          range: {
            startIndex: 1,
            endIndex: 22, // Adjust this range to match the length of
              "Click to edit taplist" (21 characters + 1 to cover the full length)
          },
          textStyle: {
            link: {
              url: url,
            },
            fontSize: {
              magnitude: 32, // Twice the original font size (16 * 2)
              unit: 'PT',
            },
            fields: 'link,fontSize',
          },
        },
      },
    ],
  },
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
  console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Summary

1. Add the `bin` Property:

Make sure the `bin` property is added to your `package.json`.

2. Specify Non-JavaScript Files in the `assets` Section:

Update the `pkg` configuration to include non-JavaScript files in the `assets` section.

3. Update the `server.js` File:

Ensure the `credentials.json` file is required using a static path.

4. Run Packaging Command:

Run the packaging command to create the executable.

```
bash
```

```
npm install
npm run pkg
```

These steps should resolve the warnings and allow you to create an executable for your project using `pkg`.



So in theory what should happen when you open the generated .exe file on a windows machine?



When you open the generated `exe` file on a Windows machine, the following should happen:

1. **Node.js Server Starts:** The executable should start a Node.js server running on port 3000. This is similar to running `node server.js` on a development machine.
2. **Serve Static Files:** The server should serve static files from the `public` directory, including `index.html`, `admin.html`, `color-admin.html`, and other static assets.
3. **Session Management:** The server should handle session management to protect the `admin.html` and `color-admin.html` pages, ensuring only authenticated users can access these pages.
4. **Dynamic Content:** The server should handle dynamic content, such as reading and writing to the `beers.json` and `colors.json` files, and updating the Google Docs document with the local IP address.
5. **Access Via Browser:** You should be able to access the web application by opening a web browser and navigating to `http://localhost:3000`. The following routes should be available:
 - o `http://localhost:3000/` to view the tap list.
 - o `http://localhost:3000/admin.html` to access the admin panel (after logging in).
 - o `http://localhost:3000/color-admin.html` to access the color admin panel (after logging in).

Steps to Run the Executable

1. **Double-Click the Executable:** Locate the generated ` .exe` file and double-click it to run. This will start the Node.js server in the background.
2. **Open a Web Browser:** Open a web browser on the same machine.
3. **Navigate to `localhost:3000`:**
 - o Go to `http://localhost:3000/` to see the tap list.
 - o Go to `http://localhost:3000/admin.html` to access the admin panel. You will need to log in with the specified password.
 - o Go to `http://localhost:3000/color-admin.html` to access the color admin panel. You will also need to log in with the specified password.

Additional Considerations

- **Dependencies:** Ensure all necessary files (`public` directory and `credentials.json`) are in the same directory as the executable.
- **Ports and Firewalls:** Ensure port 3000 is open and not blocked by a firewall.
- **User Permissions:** Ensure the user has the necessary permissions to run the executable and access the required files.

Troubleshooting

- **Server Not Starting:** If the server does not start, open a command prompt and run the executable from there to see any error messages.
- **Files Not Found:** Ensure all required files are in the same directory as the executable.
- **Access Issues:** Check firewall settings and ensure port 3000 is not blocked.

By following these steps, you should be able to run the generated ` .exe` file on a Windows machine and access your web application as intended.



What all should be in the folder with the .exe file? What should that file structure look like?



To ensure that the generated ` .exe` file runs correctly on a Windows machine, you should have the following file structure. This includes the ` .exe` file, the `public` directory, and any other necessary files such as `credentials.json`.

File Structure

```

bash

/taplist-project
/public
    admin.html
    color-admin.html
    index.html

```

```

login.html
beers.json
colors.json
banner.jpg
styles.css
script.js
credentials.json
taplist-project.exe

```

Explanation

1. **public Directory:** This directory contains all static files that your application serves.
 - o `admin.html`: The admin panel HTML file.
 - o `color-admin.html`: The color admin panel HTML file.
 - o `index.html`: The main tap list HTML file.
 - o `login.html`: The login page HTML file.
 - o `beers.json`: The JSON file containing beer information.
 - o `colors.json`: The JSON file containing color information.
 - o `banner.jpg`: The image file for the banner.
 - o `styles.css`: The CSS file for styling.
 - o `script.js`: The JavaScript file for client-side scripts.
2. **credentials.json:** This file contains your Google API credentials. It must be in the same directory as the `exe` file so that it can be accessed by your application.
3. **taplist-project.exe:** The executable file generated by `pkg`. This file runs your Node.js server.

Steps to Run the Executable

1. **Double-Click the Executable:** Locate the `taplist-project.exe` file and double-click it to run. This will start the Node.js server in the background.
2. **Open a Web Browser:** Open a web browser on the same machine.
3. **Navigate to `localhost:3000`:**
 - o Go to `http://localhost:3000/` to see the tap list.
 - o Go to `http://localhost:3000/admin.html` to access the admin panel. You will need to log in with the specified password.
 - o Go to `http://localhost:3000/color-admin.html` to access the color admin panel. You will also need to log in with the specified password.

Example Directory Structure

Here is a more detailed view of the directory structure and the contents:

```

graphql

/taplist-project
/public
    admin.html          # Admin panel HTML file
    color-admin.html    # Color admin panel HTML file

```

```

index.html          # Main tap list HTML file
login.html         # Login page HTML file
beers.json         # JSON file containing beer information
colors.json        # JSON file containing color information
banner.jpg         # Image file for the banner
styles.css         # CSS file for styling
script.js          # JavaScript file for client-side scripts
credentials.json   # Google API credentials
taplist-project.exe # Executable file generated by `pkg`
```

Final Steps

Ensure that all files are in the correct locations as specified in the directory structure above. This will allow your application to run correctly when the ` .exe` file is executed on a Windows machine.



When you run the .exe on a windows machine and then go to admin.html to edit something, once you try to save changes it says Failed to save beers



The "Failed to save beers" message indicates that there is likely an issue with the file writing process. Let's troubleshoot and ensure that the Node.js server has the necessary permissions to read and write the `beers.json` file.

Steps to Troubleshoot

- Check File Paths:** Ensure that the paths to the `beers.json` and `colors.json` files are correct in your `server.js` file. Since we are packaging with `pkg`, the paths should be handled correctly.
- Permissions:** Ensure that the application has the necessary permissions to read and write the `beers.json` and `colors.json` files. On Windows, file permissions might sometimes restrict access.
- Error Logging:** Add additional logging to the `server.js` to capture more details about why the file writing might be failing.

Updated `server.js` with Additional Logging

Add some logging to capture any errors while reading or writing files.

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path
```

```

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

```

```

});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      console.error('Error reading colors file:', err);
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
  const updatedColors = req.body;
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
    if (err) {
      console.error('Failed to save colors:', err);
      res.status(500).send('Failed to save colors');
    } else {
      res.send('Colors saved successfully');
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    file path
    scopes: ['https://www.googleapis.com/auth/documents']
  });

  const docs = google.docs({ version: 'v1', auth });

  // Get the current time in CST
  const now = new Date();
  const cstOffset = -6; // CST is UTC-6
  const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
  const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '').+'CST';

  const url = `http://${localIp}:3000/admin.html`;
  const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

  try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
  }
}

```

```

const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
  return element.endIndex > maxIndex ? element.endIndex : maxIndex;
}, 1);

// Adjust the endIndex to exclude the newline character
const adjustedEndIndex = endIndex - 1;

// Clear existing content
await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        deleteContentRange: {
          range: {
            startIndex: 1,
            endIndex: adjustedEndIndex,
          },
        },
      ],
    ],
  },
});

// Insert new content
const response = await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        insertText: {
          location: {
            index: 1,
          },
          text: textToInsert,
        },
      },
      {
        updateTextStyle: {
          range: {
            startIndex: 1,
            endIndex: 22, // Adjust this range to match the length of
              "Click to edit taplist" (21 characters + 1 to cover the full length)
          },
          textStyle: {
            link: {
              url: url,
            },
            fontSize: {
              magnitude: 32, // Twice the original font size (16 * 2)
              unit: 'PT',
            },
          },
          fields: 'link,fontSize',
        },
      },
    ],
  },
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
  console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running and Testing

- Run the Executable:** Double-click the `*.exe` file to start the server.
- Open a Web Browser:** Navigate to `http://localhost:3000/admin.html`.
- Login and Edit:** Attempt to edit the beer list and save changes.
- Check Logs:** If saving fails, check the console logs for error messages. This should help you pinpoint the issue.

By adding the logging, you can capture and understand what might be causing the "Failed to save beers" error. This approach helps diagnose issues related to file paths, permissions, or other unexpected errors during the file write process.



Where are the console logs with error messages? I don't see anything in the terminal window the .exe opens up



To view console logs and error messages when running the .exe file, you can redirect the output to a file or make sure the console window stays open. Here are some methods to help you see the logs:

Method 1: Keep Console Window Open

Make sure the console window stays open when running the .exe file by running it from the Command Prompt:

- Open Command Prompt:** Press `Win + R`, type `cmd`, and press `Enter`.
- Navigate to the Directory:** Use the `cd` command to navigate to the directory containing your `*.exe` file.

```
sh
cd path\to\your\taplist-project
```

- Run the Executable:** Run the executable by typing its name and pressing `Enter`.

```
sh
taplist-project.exe
```

The console will stay open, and you should be able to see any logs or error messages printed there.

Method 2: Redirect Output to a Log File

Modify your `server.js` to log messages to a file. This way, you can check the log file for any errors.

- Update `server.js`:** Add code to log to a file.

```
javascript
```

```
const fs = require('fs');
const path = require('path');
const logFile = fs.createWriteStream(path.join(__dirname, 'server.log'), { flags: 'a' });

function logToFile(message) {
    const timestamp = new Date().toISOString();
    logFile.write(`[${timestamp}] ${message}\n`);
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

// Your existing server.js code...
```

2. Run the Executable: Double-click the `*.exe` file to start the server.

3. Check the Log File: Open the `server.log` file in the same directory as your `*.exe` file to see the logs.

Example Updated `server.js`:

Here's an example of how to update your `server.js` to log messages to a file:

```
javascript
```

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

const logFile = fs.createWriteStream(path.join(__dirname, 'server.log'), { flags: 'a' });

function logToFile(message) {
    const timestamp = new Date().toISOString();
    logFile.write(`[${timestamp}] ${message}\n`);
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));
```

```

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading colors file:', err);
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
    const updatedColors = req.body;
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {

```

```

    if (err) {
      console.error('Failed to save colors:', err);
      res.status(500).send('Failed to save colors');
    } else {
      res.send('Colors saved successfully');
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
    file path
    scopes: ['https://www.googleapis.com/auth/documents']
  });

  const docs = google.docs({ version: 'v1', auth });

  // Get the current time in CST
  const now = new Date();
  const cstOffset = -6; // CST is UTC-6
  const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
  const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, ''). + 'CST';

  const url = `http://${localIp}:3000/admin.html`;
  const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

  try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
    const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
      return element.endIndex > maxIndex ? element.endIndex : maxIndex;
    }, 1);

    // Adjust the endIndex to exclude the newline character
    const adjustedEndIndex = endIndex - 1;

    // Clear existing content
    await docs.documents.batchUpdate({
      documentId: DOC_ID,
      requestBody: {
        requests: [
          {
            deleteContentRange: {
              range: {
                startIndex: 1,

```

```

        endIndex: adjustedEndIndex,
    },
},
],
},
}),
};

// Insert new content
const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
        requests: [
            {
                insertText: {
                    location: {
                        index: 1,
                    },
                    text: textToInsert,
                },
            },
            {
                updateTextStyle: {
                    range: {
                        startIndex: 1,
                        endIndex: 22, // Adjust this range to match the length of
                        "Click to edit taplist" (21 characters + 1 to cover the full length)
                    },
                    textStyle: {
                        link: {
                            url: url,
                        },
                        fontSize: {
                            magnitude: 32, // Twice the original font size (16 * 2)
                            unit: 'PT',
                        },
                        fields: 'link,fontSize',
                    },
                },
            },
        ],
    },
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Next Steps

- Run the Executable:** Double-click the `*.exe` file to start the server.
- Check `server.log`:** Open the `server.log` file in the same directory as your `*.exe` file to see the logs.
- Test Admin Panel:** Navigate to `http://localhost:3000/admin.html`, log in, and try to edit and save changes.

This setup will ensure you have a way to capture and view error messages even if the console window doesn't display them directly.



can we make it also so index.html refreshes every 60 seconds



To make `index.html` refresh every 60 seconds, you can add a small JavaScript snippet that uses `setInterval` to reload the page. Here's how you can update `index.html`:

Updated `index.html`:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tap List</title>
    <link rel="stylesheet" href="styles.css">
    <script>
        // Refresh the page every 60 seconds (60000 milliseconds)
        setInterval(function() {
            window.location.reload();
        }, 60000);
    </script>
</head>
<body>
    <div class="banner">
        
    </div>
    <div class="tap-list" id="tap-list">
        <!-- Beers will be dynamically inserted here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            fetch('/colors')
                .then(response => response.json())
                .then(colors => {
                    document.documentElement.style.setProperty('--even-line-color',
colors.evenColor);
                    document.documentElement.style.setProperty('--odd-line-color',
colors.oddColor);
                    document.documentElement.style.setProperty('--text-color',
colors.textColor);

                    fetch('/beers')
                        .then(response => response.json())
                        .then(beers => {
                            const tapListDiv = document.getElementById('tap-list');
                            beers.forEach((beer, index) => {
                                if (beer.hide) return; // Skip hidden beers
                                const beerDiv = document.createElement('div');
                                beerDiv.classList.add('beer');
                                if (beer.empty) {
                                    beerDiv.classList.add('faded');
                                }
                                beerDiv.innerHTML = `
                                    <p>
                                        <span class="number">${index + 1}</span>
                                        <span class="name">${beer.name}</span>
                                        <span class="by"> by </span>
                                        <span class="brewery">${beer.brewery} - ${beer.type}</span>
                                    </p>
                                `;
                                tapListDiv.appendChild(beerDiv);
                            });
                        });
                });
        });
    </script>
</body>
</html>
```

```

- ${beer.abv}</span>
          </p>';
      tapListDiv.appendChild(beerDiv);
    });
  });
};

</script>
</body>
</html>

```

Explanation

- **JavaScript Snippet:** The `<script>` tag inside the `<head>` section includes a `setInterval` function that reloads the page every 60 seconds (60000 milliseconds).
- **Document Content:** The rest of the `index.html` file remains the same, with dynamic content being loaded and styled as before.

By adding this snippet, `index.html` will automatically refresh every 60 seconds, ensuring that any changes to the tap list are updated without the need for manual refresh.



It still says Failed to save beers and there's no console errors being output and there is also no server.log file being generated



Let's troubleshoot the issue step-by-step:

1. **Ensure Logging is Correctly Implemented:** Make sure the logging is set up correctly in `server.js` to capture errors. Additionally, ensure that the path to the log file is correct and writable.
2. **Check File Paths and Permissions:** Verify that the `beers.json` and `colors.json` files are in the correct location and that the application has the necessary permissions to read and write to these files.
3. **Add More Detailed Logging:** Enhance the logging to capture more details about the request and response.

Updated `server.js` with Enhanced Logging

Here's an updated version of `server.js` that includes more detailed logging:

```

javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');

```

```

const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

const logFile = fs.createWriteStream(path.join(__dirname, 'server.log'), { flags: 'a' });

function logToFile(message) {
    const timestamp = new Date().toISOString();
    logFile.write(`[${timestamp}] ${message}\n`);
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            res.status(500).send('Error reading beers file');
        } else {

```

```

        res.json(JSON.parse(data)));
    }
});

app.post('/save-beers', isAuthenticated, (req, res) => {
  console.log(`Received request to save beers: ${JSON.stringify(req.body)}`);
  const updatedBeers = req.body;
  fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
      console.error('Failed to save beers:', err);
      res.status(500).send('Failed to save beers');
    } else {
      console.log('Beers saved successfully');
      res.send('Beers saved successfully');
    }
  });
});

app.get('/colors', (req, res) => {
  fs.readFile(COLORS_FILE_PATH, (err, data) => {
    if (err) {
      console.error('Error reading colors file:', err);
      res.status(500).send('Error reading colors file');
    } else {
      res.json(JSON.parse(data));
    }
  });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
  console.log(`Received request to save colors: ${JSON.stringify(req.body)}`);
  const updatedColors = req.body;
  fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
    if (err) {
      console.error('Failed to save colors:', err);
      res.status(500).send('Failed to save colors');
    } else {
      console.log('Colors saved successfully');
      res.send('Colors saved successfully');
    }
  });
});

app.get('/ip', (req, res) => {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
  res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = '';
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }

  const auth = new google.auth.GoogleAuth({
    keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
  });
}

```

```

file path
    scopes: ['https://www.googleapis.com/auth/documents']
});

const docs = google.docs({ version: 'v1', auth });

// Get the current time in CST
const now = new Date();
const cstOffset = -6; // CST is UTC-6
const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '').+ 'CST';

const url = `http://${localIp}:3000/admin.html`;
const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

try {
    // Get the document to determine its end index
    const doc = await docs.documents.get({ documentId: DOC_ID });
    const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
        return element.endIndex > maxIndex ? element.endIndex : maxIndex;
    }, 1);

    // Adjust the endIndex to exclude the newline character
    const adjustedEndIndex = endIndex - 1;

    // Clear existing content
    await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: {
            requests: [
                {
                    deleteContentRange: {
                        range: {
                            startIndex: 1,
                            endIndex: adjustedEndIndex,
                        },
                    },
                },
            ],
        },
    });
}

// Insert new content
const response = await docs.documents.batchUpdate({
    documentId: DOC_ID,
    requestBody: {
        requests: [
            {
                insertText: {
                    location: {
                        index: 1,
                    },
                    text: textToInsert,
                },
            },
            {
                updateTextStyle: {
                    range: {
                        startIndex: 1,
                        endIndex: 22, // Adjust this range to match the length of
                        "Click to edit taplist" (21 characters + 1 to cover the full length)
                    },
                    textStyle: {
                        link: {
                            url: url,
                        },
                        fontSize: {
                            magnitude: 32, // Twice the original font size (16 * 2)
                            unit: 'PT',
                        },
                    },
                    fields: 'link,fontSize',
                },
            },
        ],
    },
});

```

```

        ],
      });
      console.log('IP address sent to Google Docs successfully:', response.data);
    } catch (error) {
      console.error('Failed to send IP address to Google Docs:', error);
    }
  }

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Steps to Verify and Troubleshoot

- Run the Executable:** Run the ` `.exe` file again by double-clicking it.
- Check the `server.log` File:** Look for the `server.log` file in the same directory as your ` `.exe` file. Open it to see if any errors are logged.
- Attempt to Save Beers:** Navigate to `http://localhost:3000/admin.html`, log in, and try to save changes.
- Review Logs:** If saving fails, check the `server.log` file for any error messages or logs related to the save operation.

This enhanced logging should provide more information about why the save operation might be failing and help you diagnose and fix the issue.



Still says Failed to save beers and there's no server.log file being generated



Let's ensure we cover all potential issues step by step:

- Verify File Paths and Permissions:** Ensure that `beers.json` exists in the `public` directory and is writable.
- Confirm Correct Paths in `server.js`:** Double-check that the paths to `beers.json` and `colors.json` are correct.
- Console Output:** Ensure console logging is working properly.
- Simplify and Test:** Simplify the `server.js` to narrow down the issue.

Step-by-Step Guide

1. Verify File Paths and Permissions

Make sure `beers.json` and `colors.json` exist in the `public` directory and are writable. Check the file permissions.

2. Confirm Correct Paths in `server.js`

Ensure the paths in `server.js` are correct and use the `path` module to handle paths properly.

3. Check Console Output

First, run the `exe` from the command prompt to ensure you see the console output.

1. **Open Command Prompt:** Press `Win + R`, type `cmd`, and press `Enter`.

2. **Navigate to Directory:** Use `cd` to navigate to your project's directory.

```
sh
cd path\to\your\taplist-project
```

3. **Run Executable:** Run the executable from the command prompt.

```
sh
taplist-project.exe
```

4. Simplify and Test

Simplify the `server.js` to focus on saving `beers.json` and ensure logging works.

Simplified `server.js` with Basic Logging

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const logFilePath = path.join(__dirname, 'server.log');

const logToFile = (message) => {
    const timestamp = new Date().toISOString();
    fs.appendFileSync(logFilePath, `[${timestamp}] ${message}\n`, 'utf8');
};

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) =>
```

```

if (req.session.loggedIn) {
    next();
} else {
    res.redirect('/login.html');
}
};

// Serve static files from public directory, but protect admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    console.log(`Received request to save beers: ${JSON.stringify(req.body)}`);
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            res.status(500).send('Failed to save beers');
        } else {
            console.log('Beers saved successfully');
            res.send('Beers saved successfully');
        }
    });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

```

Test Again

- Run the Executable:** Run the `exe` from the command prompt to see any console output.
- Check the `server.log` File:** Check if the `server.log` file is generated and contains logs.
- Attempt to Save Beers:** Navigate to `http://localhost:3000/admin.html`, log in, and try to save changes.
- Review Logs:** Review the `server.log` file for any errors.

By following these steps, you should be able to identify why saving beers is failing and ensure that logs are correctly captured.



Still says Failed to save beers and there's no server.log file being generated



Let's further troubleshoot this by isolating and confirming the basic functionality step-by-step. We'll start by simplifying the logging and file handling to ensure that everything is working correctly.

Simplified Steps and Debugging

- Ensure Files Exist and Are Writable:** Double-check that `beers.json` exists in the `public` directory and that it is writable.
- Check Paths:** Verify that the paths in `server.js` are correct.
- Console Output:** Ensure that the console output works when running the `exe` from the command prompt.
- Minimal Example:** Create a minimal example of `server.js` to test basic functionality.

Minimal `server.js` for Debugging

Here's a minimal version of `server.js` that focuses on saving to `beers.json` and includes basic logging:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const LOG_FILE_PATH = path.join(__dirname, 'server.log');

// Function to log messages to a file
function logToFile(message) {
    const timestamp = new Date().toISOString();
    fs.appendFileSync(LOG_FILE_PATH, `[${timestamp}] ${message}\n`, 'utf8');
}

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    }
}
```

```

    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory
app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            logToFile(`Error reading beers file: ${err}`);
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    console.log(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    logToFile(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            logToFile(`Failed to save beers: ${err}`);
            res.status(500).send('Failed to save beers');
        } else {
            console.log('Beers saved successfully');
            logToFile('Beers saved successfully');
            res.send('Beers saved successfully');
        }
    });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    logToFile(`Server is running on http://localhost:${PORT}`);
});

```

Test Procedure

1. **Create `beers.json`:** Ensure that `beers.json` exists in the `public` directory with initial content. For example:

```
json
```

```
[]
```

2. **Run from Command Prompt:**

- Open Command Prompt.
- Navigate to your project directory.
- Run the executable:

```
sh
taplist-project.exe
```

3. **Verify Console Output:** Check the command prompt for console output.
4. **Check `server.log`:** Ensure that the `server.log` file is generated and contains logs.
5. **Access Admin Page:** Open a browser and go to `http://localhost:3000/admin.html`.
6. **Login and Test:** Log in and attempt to save beers. Check for errors in the console and log file.

Additional Debugging Steps

- **File System Permissions:** Ensure the application has permission to write to the `public` directory and `server.log`.
- **Dynamic Path Resolution:** Make sure the dynamic path resolution using `path.join(__dirname, ...)` is correct.

By following these steps, you should be able to isolate the issue and ensure that logging works, helping you identify why saving beers is failing.



Can you elaborate on this

"File System Permissions: Ensure the application has permission to write to the public directory and server.log.

Dynamic Path Resolution: Make sure the dynamic path resolution using path.join(__dirname, ...) is correct."



Certainly! Here's a more detailed explanation of both file system permissions and dynamic path resolution:

File System Permissions

File system permissions determine whether your application can read from and write to specific files and directories. If your application lacks the necessary permissions, it won't be able to perform file operations, leading to errors.

Check and Set Permissions on Windows

1. **Locate the Directory:** Find the directory containing your `public` folder and `server.js` file.
2. **Check Permissions:**

- Right-click on the directory and select "Properties".
- Go to the "Security" tab.
- Click "Edit" to change permissions.

3. Set Permissions:

- Ensure the user account running the application has "Full Control" permission.
- If the user doesn't have "Full Control", add it by selecting the user and checking the "Full Control" box under "Allow".

4. Apply Changes:

Click "Apply" and "OK" to save the changes.

Dynamic Path Resolution

Dynamic path resolution ensures that your file paths are correctly constructed, regardless of where the application is executed. This is crucial when packaging Node.js applications with `pkg`.

Using `path.join(__dirname, ...)` helps to construct file paths relative to the current file's directory (`__dirname`). This avoids issues related to the current working directory (`process.cwd()`), which might be different when running the application.

Example: Using `path.join(__dirname, ...)`

Here's how you use `path.join(__dirname, ...)` to create paths dynamically:

```
javascript

const path = require('path');

// Example paths
const beersFilePath = path.join(__dirname, 'public', 'beers.json');
const logFilePath = path.join(__dirname, 'server.log');
```

This ensures that `beers.json` and `server.log` are correctly located in the same directory as your script.

Combining Both Concepts in Your Application

1. **Check Permissions:** Ensure that the application can write to the `public` directory and create/write to `server.log`.
2. **Verify Dynamic Paths:** Confirm that the paths constructed using `path.join(__dirname, ...)` are correct.

Full Example with Detailed Logging and Permission Check

Here's an updated `server.js`:

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const LOG_FILE_PATH = path.join(__dirname, 'server.log');

// Function to log messages to a file
function logToFile(message) {
    const timestamp = new Date().toISOString();
    fs.appendFileSync(LOG_FILE_PATH, `[$timestamp] ${message}\n`, 'utf8');
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory
app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            logToFile(`Error reading beers file: ${err}`);
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    console.log(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
});

```

```

logToFile(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
    if (err) {
        console.error('Failed to save beers:', err);
        logToFile(`Failed to save beers: ${err}`);
        res.status(500).send('Failed to save beers');
    } else {
        console.log('Beers saved successfully');
        logToFile('Beers saved successfully');
        res.send('Beers saved successfully');
    }
});
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    logToFile(`Server is running on http://localhost:${PORT}`);
});

```

Final Steps

- Ensure Permissions:** Check that the `public` directory and `server.log` have the correct permissions as outlined.
- Run the Executable:** Run the `exe` file from the command prompt to monitor console output.
- Check `server.log`:** Verify that the `server.log` file is created and contains logs.
- Test Functionality:** Log in to the admin panel and try to save changes, then check the logs for any errors.

By ensuring correct file system permissions and dynamic path resolution, and by adding detailed logging, you should be able to diagnose and resolve the issue with saving the beer list.



So on my windows machine, if I run server.js in an IDE it will create server.log, but when I run the .exe file it does not create server.log



If running the `server.js` in an IDE creates the `server.log` file, but running the `exe` file does not, it suggests that there might be an issue with how the executable is handling paths or permissions when it runs.

Let's address the following potential issues:

- Current Working Directory:** When you run the `exe` file, the current working directory might not be what you expect.
- Permissions:** The `exe` file might not have the necessary permissions to create or write to the `server.log` file.

Step-by-Step Solution

- Ensure Correct Path Handling:** Ensure that the paths are correctly handled and the `server.log` file is created in the same directory as the executable.

2. **Ensure Permissions:** Verify that the ` .exe` has the necessary permissions to create and write to files in its directory.

Updated `server.js` to Handle Path and Permissions

Here's how you can modify `server.js` to ensure it handles paths correctly:

1. **Ensure the Log Directory Exists:** Before writing to the log file, ensure the directory exists.
2. **Use Absolute Paths:** Use absolute paths to ensure the paths are correctly resolved.

Updated `server.js`

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const LOG_DIR = __dirname;
const LOG_FILE_PATH = path.join(LOG_DIR, 'server.log');

// Ensure the log directory exists
if (!fs.existsSync(LOG_DIR)) {
    fs.mkdirSync(LOG_DIR, { recursive: true });
}

// Function to log messages to a file
function logToFile(message) {
    const timestamp = new Date().toISOString();
    fs.appendFileSync(LOG_FILE_PATH, `[${timestamp}] ${message}\n`, 'utf8');
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory
app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
    }
});
```

```

        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            logToFile(`Error reading beers file: ${err}`);
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    console.log(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    logToFile(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            logToFile(`Failed to save beers: ${err}`);
            res.status(500).send('Failed to save beers');
        } else {
            console.log('Beers saved successfully');
            logToFile('Beers saved successfully');
            res.send('Beers saved successfully');
        }
    });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    logToFile(`Server is running on http://localhost:${PORT}`);
});

```

Test the Executable

1. **Rebuild the Executable:** Ensure you rebuild the ` .exe` file with the updated `server.js`.

2. **Run from Command Prompt:**

- o Open Command Prompt.
- o Navigate to your project directory.
- o Run the executable:

```

sh
taplist-project.exe

```

3. **Verify:** Check the directory for the `server.log` file and ensure it contains the expected logs.

By ensuring correct path handling and permissions, this setup should enable the `server.log` file to be created and written to correctly when running the ` .exe` file.



this error was returned when trying to run the .exe in the cmd prompt

```
pkg/prelude/bootstrap.js:438
```

```
if (error) throw error;
```

```
^
```

Error: File or directory 'C:**\Taplist-Project\server.log' was not included into executable at compilation stage. Please recompile adding it as asset or script.

```
at errorENOENT (pkg/prelude/bootstrap.js:657:19)
```

```
at openFromSnapshot (pkg/prelude/bootstrap.js:755:29)
```

```
at Object.openSync (pkg/prelude/bootstrap.js:798:12)
```

```
at Object.writeFileSync (fs.js:1529:35)
```

```
at Object.appendFileSync (fs.js:1568:6)
```

```
at logToFile (C:\snapshot\Taplist-Project\server.js:22:8)
```

```
at console.log (C:\snapshot\Taplist-Project\server.js:26:28)
```

```
at Server.<anonymous> (C:\snapshot\Taplist-Project\server.js:93:13)
```

```
at Object.onceWrapper (events.js:519:28)
```

```
at Server.emit (events.js:400:28) {
```

```
errno: -4058,
```

```
code: 'ENOENT',
```

```
path: 'C:\snapshot\Taplist-Project\server.log',
```

```
pkg: true
```

```
}
```



The error you're seeing indicates that the `server.log` file wasn't included as an asset in the executable during the compilation stage. Since `pkg` creates a virtual filesystem within the executable, you need to handle paths to writable locations outside the virtual filesystem.

Solution

We should place the `server.log` file in a directory that is guaranteed to be writable, such as a temporary directory or a user directory.

Update `server.js` to Use a Writable Directory

We'll use the system's temporary directory for logging.

Updated `server.js`:

```
javascript
```

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
```

```

const fs = require('fs');
const session = require('express-session');
const os = require('os');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const LOG_FILE_PATH = path.join(os.tmpdir(), 'server.log');

// Function to log messages to a file
function logToFile(message) {
    const timestamp = new Date().toISOString();
    fs.appendFileSync(LOG_FILE_PATH, `[$timestamp] ${message}\n`, 'utf8');
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory
app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            logToFile(`Error reading beers file: ${err}`);
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    console.log(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    logToFile(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            logToFile(`Failed to save beers: ${err}`);
            res.status(500).send('Failed to save beers');
        }
    });
});

```

```

    } else {
      console.log('Beers saved successfully');
      logToFile('Beers saved successfully');
      res.send('Beers saved successfully');
    }
  });
};

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  logToFile(`Server is running on http://localhost:${PORT}`);
});

```

Rebuild the Executable

- 1. Rebuild the Executable:** Ensure you rebuild the `*.exe` file with the updated `server.js`.

```

bash

npm run pkg

```

- 2. Run from Command Prompt:**

- Open Command Prompt.
- Navigate to your project directory.
- Run the executable:

```

sh

taplist-project.exe

```

- 3. Check the Log File:**

- Open the system's temporary directory. You can find it by running `%TEMP%` in the Run dialog (Win + R).
- Look for the `server.log` file in the temporary directory and open it to see the logs.

Verify the Changes

- 1. Run the Executable:** Run the `*.exe` file and verify it starts without errors.
- 2. Check the Temporary Directory:** Verify that the `server.log` file is created and logs are being written to it.
- 3. Test Admin Functionality:** Open a browser, navigate to `http://localhost:3000/admin.html`, log in, and try to save changes. Verify that logs are correctly captured in `server.log`.

By writing the log file to the system's temporary directory, you ensure that the application can write logs without encountering issues related to the virtual filesystem created by `pkg`.



The logs are in temp now but when you try to use admin.html it is not functional and returns a dialogue box full of html when you press the save button



If the `admin.html` is returning a dialog box full of HTML when you press the save button, it suggests that there might be an issue with the response handling or an error being sent as HTML instead of JSON.

Let's break down the issue and add more debugging information to understand what's happening. We need to:

1. **Ensure Proper Error Handling:** Make sure all errors are correctly handled and returned as JSON where necessary.
2. **Check `admin.html` JavaScript:** Ensure the JavaScript in `admin.html` correctly handles the responses.

Updated `server.js`

First, let's ensure the server is correctly handling responses as JSON:

```
javascript

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const LOG_FILE_PATH = path.join(os.tmpdir(), 'server.log');

// Function to log messages to a file
function logToFile(message) {
    const timestamp = new Date().toISOString();
    fs.appendFileSync(LOG_FILE_PATH, `[${timestamp}] ${message}\n`, 'utf8');
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedin) {
        next();
    }
}
```

```

    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory
app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            logToFile(`Error reading beers file: ${err}`);
            res.status(500).json({ error: 'Error reading beers file' });
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    console.log(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    logToFile(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            logToFile(`Failed to save beers: ${err}`);
            res.status(500).json({ error: 'Failed to save beers' });
        } else {
            console.log('Beers saved successfully');
            logToFile('Beers saved successfully');
            res.json({ message: 'Beers saved successfully' });
        }
    });
});

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    logToFile(`Server is running on http://localhost:${PORT}`);
});

```

Updated `admin.html`

Ensure the JavaScript in `admin.html` properly handles responses:

```

html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
```

```

    .editable {
        border: 1px solid #000;
        padding: 5px;
    }
    .faded {
        opacity: 0.5;
    }
    #editButton, #saveButton {
        font-size: 2em; /* Increase font size */
        padding: 10px 20px;
    }
</style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here -->
    </div>
    <script>
        document.addEventListener('DOMContentLoaded', () => {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response => response.json())
                .then(beers => {
                    beers.forEach((beer, index) => {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML = `
                            <p>
                                <span class="number">${index + 1}</span>
                                <span class="name">${beer.name}</span>
                                <span class="by"> by </span>
                                <span class="brewery">${beer.brewery}</span> -
                                <span class="type">${beer.type}</span> -
                                <span class="abv">${beer.abv}</span>
                                <label>Empty <input type="checkbox" data-field="empty"
${
                            beer.empty ? 'checked' : ''
                        } disabled></label>
                            </p>`;
                        beerDiv.dataset.index = index;
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        tapListDiv.appendChild(beerDiv);
                    });
                });
            editButton.addEventListener('click', () => {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div => {
                    const index = div.dataset.index;
                    const spans = div.querySelectorAll('span');
                    const name = spans[1].textContent;
                    const brewery = spans[3].textContent;
                    const type = spans[4].textContent;
                    const abv = spans[5].textContent
                });
            });
        });
    </script>
</body>

```



I'm rolling back server.js to the following

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');

```

```
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi4O7NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
```

```
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save beers');
        } else {
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
    const updatedColors = req.body;
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
        if (err) {
            res.status(500).send('Failed to save colors');
        } else {
            res.send('Colors saved successfully');
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
```

```

let localIp = "";
for (const interfaceName in networkInterfaces) {
  const interfaces = networkInterfaces[interfaceName];
  for (const i of interfaces) {
    if (i.family === 'IPv4' && !i.internal) {
      localIp = i.address;
      break;
    }
  }
}
res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
  const networkInterfaces = os.networkInterfaces();
  let localIp = "";
  for (const interfaceName in networkInterfaces) {
    const interfaces = networkInterfaces[interfaceName];
    for (const i of interfaces) {
      if (i.family === 'IPv4' && !i.internal) {
        localIp = i.address;
        break;
      }
    }
  }
}

const auth = new google.auth.GoogleAuth({
  keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials file path
  scopes: ['https://www.googleapis.com/auth/documents']
});

const docs = google.docs({ version: 'v1', auth });

// Get the current time in CST
const now = new Date();
const cstOffset = -6; // CST is UTC-6
const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '') + ' CST';

const url = `http://${localIp}:3000/admin.html`;
const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

try {
  // Get the document to determine its end index
  const doc = await docs.documents.get({ documentId: DOC_ID });
  const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
    return element.endIndex > maxIndex ? element.endIndex : maxIndex;
  }, 1);

  // Adjust the endIndex to exclude the newline character
}

```

```
const adjustedEndIndex = endIndex - 1;

// Clear existing content
await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        deleteContentRange: {
          range: {
            startIndex: 1,
            endIndex: adjustedEndIndex,
          },
        },
      },
    ],
  },
});

// Insert new content
const response = await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        insertText: {
          location: {
            index: 1,
          },
          text: textToInsert,
        },
      },
      {
        updateTextStyle: {
          range: {
            startIndex: 1,
            endIndex: 22, // Adjust this range to match the length of "Click to edit taplist" (21
                           // characters + 1 to cover the full length)
          },
          textStyle: {
            link: {
              url: url,
            },
            fontSize: {
              magnitude: 32, // Twice the original font size (16 * 2)
              unit: 'PT',
            },
          },
          fields: 'link,fontSize',
        },
      },
    ],
  },
});
```

```

        ],
    },
});

console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```



Alright, let's re-implement the minimal logging to make sure we capture any issues while keeping your server functional.

Updated `server.js` with Minimal Logging

1. We'll log to the system's temporary directory.
2. We'll capture the logs and errors in a simple format.

javascript

```

const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID
const LOG_FILE_PATH = path.join(os.tmpdir(), 'server.log');

// Function to log messages to a file
function logToFile(message) {
    const timestamp = new Date().toISOString();
    fs.appendFileSync(LOG_FILE_PATH, `[$${timestamp}] ${message}\n`, 'utf8');
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
})

```

```

});;

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            logToFile(`Error reading beers file: ${err}`);
            res.status(500).send('Error reading beers file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    console.log(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    logToFile(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            logToFile(`Failed to save beers: ${err}`);
            res.status(500).send('Failed to save beers');
        } else {
            console.log('Beers saved successfully');
            logToFile('Beers saved successfully');
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading colors file:', err);
            logToFile(`Error reading colors file: ${err}`);
            res.status(500).send('Error reading colors file');
        }
    });
});

```

```

        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
    const updatedColors = req.body;
    console.log(`Received request to save colors: ${JSON.stringify(updatedColors)}`);
    logToFile(`Received request to save colors: ${JSON.stringify(updatedColors)}`);
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
        if (err) {
            console.error('Failed to save colors:', err);
            logToFile(`Failed to save colors: ${err}`);
            res.status(500).send('Failed to save colors');
        } else {
            console.log('Colors saved successfully');
            logToFile('Colors saved successfully');
            res.send('Colors saved successfully');
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });

    // Get the current time in CST
    const now = new Date();
    const cstOffset = -6; // CST is UTC-6
    const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
    const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, '').+'CST';

    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 0);
    }
}

```

```

    }, 1);

// Adjust the endIndex to exclude the newline character
const adjustedEndIndex = endIndex - 1;

// Clear existing content
await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        deleteContentRange: {
          range: {
            startIndex: 1,
            endIndex: adjustedEndIndex,
          },
        },
      ],
    ],
  },
});

// Insert new content
const response = await docs.documents.batchUpdate({
  documentId: DOC_ID,
  requestBody: {
    requests: [
      {
        insertText: {
          location: {
            index: 1,
          },
          text: textToInsert,
        },
      },
      {
        updateTextStyle: {
          range: {
            startIndex: 1,
            endIndex: 22, // Adjust this range to match the length of
              "Click to edit taplist" (21 characters + 1 to cover the full length)
          },
          textStyle: {
            link: {
              url: url,
            },
            fontSize: {
              magnitude: 32, // Twice the original font size (16 * 2)
              unit: 'PT',
            },
            fields: 'link,fontSize',
          },
        },
      },
    ],
  },
});
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
  console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
  console.log(`Server is running on http://localhost:${PORT}`);
  sendIpToGoogleDocs(); // Send IP address at startup
  setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Updated `admin.html`

Ensure the JavaScript in `admin.html` properly handles responses:

```
html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - Edit Tap List</title>
    <style>
        .editable {
            border: 1px solid #000;
            padding: 5px;
        }
        .faded {
            opacity: 0.5;
        }
        #editButton, #saveButton {
            font-size: 2em; /* Increase font size */
            padding: 10px 20px;
        }
    </style>
</head>
<body>
    <h1>Edit Tap List</h1>
    <button id="editButton">Edit</button>
    <button id="saveButton" style="display: none;">Save</button>
    <div id="tapList">
        <!-- Beer list will be populated here --&gt;
    &lt;/div&gt;
    &lt;script&gt;
        document.addEventListener('DOMContentLoaded', () =&gt; {
            const tapListDiv = document.getElementById('tapList');
            const editButton = document.getElementById('editButton');
            const saveButton = document.getElementById('saveButton');

            fetch('/beers')
                .then(response =&gt; response.json())
                .then(beers =&gt; {
                    beers.forEach((beer, index) =&gt; {
                        const beerDiv = document.createElement('div');
                        beerDiv.innerHTML =
                            &lt;p&gt;
                                &lt;span class="number"&gt;${index + 1}&lt;/span&gt;
                                &lt;span class="name"&gt;${beer.name}&lt;/span&gt;
                                &lt;span class="by"&gt; by &lt;/span&gt;
                                &lt;span class="brewery"&gt;${beer.brewery}&lt;/span&gt; -
                                &lt;span class="type"&gt;${beer.type}&lt;/span&gt; -
                                &lt;span class="abv"&gt;${beer.abv}&lt;/span&gt;
                                &lt;label&gt;Empty &lt;input type="checkbox" data-field="empty"
${beer.empty ? 'checked' : ''} disabled&gt;&lt;/label&gt;
                            &lt;/p&gt;;
                        beerDiv.dataset.index = index;
                        if (beer.empty) {
                            beerDiv.classList.add('faded');
                        }
                        tapListDiv.appendChild(beerDiv);
                    });
                });
            editButton.addEventListener('click', () =&gt; {
                const beerDivs = tapListDiv.querySelectorAll('div');
                beerDivs.forEach(div =&gt; {
                    const index = div.dataset.index;
                    const spans = div.querySelectorAll('span');
                    const name = spans[1].textContent;
                    const brewery = spans[3].textContent;
                    const type = spans[4].textContent;
                    const abv = spans[5].textContent;
                    const empty = div.querySelector('input[data-field="empty"]').checked;
                });
            });
        });
    &lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>

```

```

        div.innerHTML = ` 
            <span class="number">${index + 1}.</span>
            <input type="text" value="${name}" data-field="name">
            <span class="by"> by </span>
            <input type="text" value="${brewery}" data-field="brewery">
            - <input type="text" value="${type}" data-field="type">
            - <input type="text" value="${abv}" data-field="abv">
            <label>Empty <input type="checkbox" data-field="empty" ${empty ? 
'checked' : ''}></label>
        `;
        div.classList.add('editable');
        if (empty) {
            div.classList.add('faded');
        }
    });
    editButton.style.display = 'none';
    saveButton.style.display = 'block';
});

saveButton.addEventListener('click', () => {
    const updatedBeers = [];
    const beerDivs = tapListDiv.querySelectorAll('div');
    beerDivs.forEach(div => {
        const inputs = div.querySelectorAll('input[type="text"]');
        const checkbox = div.querySelector('input[type="checkbox"]');
        const beer = {};
        inputs.forEach(input => {
            const field = input.dataset.field;
            beer[field] = input.value;
        });
        beer.empty = checkbox.checked;
        updatedBeers.push(beer);
    });
    fetch('/save-beers', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(updatedBeers)
    })
    .then(response => response.json())
    .then(result => {
        if (result.message === 'Beers saved successfully') {
            beerDivs.forEach((div, index) => {
                const beer = updatedBeers[index];
                div.innerHTML = `
                    <p>
                        <span class="number">${index + 1}.</span>
                        <span class="name">${beer.name}</span>
                        <span class="by"> by </span>
                        <span class="brewery">${beer.brewery}</span> -
                        <span class="type">${beer.type}</span> -
                        <span class="abv">${beer.abv}</span>
                        <label>Empty <input type="checkbox" data-field="empty" ${beer.empty ? 'checked' : ''} disabled></label>
                    </p>`;
                div.classList.remove('editable');
                if (beer.empty) {
                    div.classList.add('faded');
                } else {
                    div.classList.remove('faded');
                }
            });
            editButton.style.display = 'block';
            saveButton.style.display = 'none';
        } else {
            alert(result.error);
        }
    })
    .catch(error => {
        alert('Failed to save beers');
        console.error(error);
    });
});
});

```

```
    });
  </script>
</body>
</html>
```

Steps to Test

- 1. Rebuild the Executable:** Ensure you rebuild the `*.exe` file with the updated `server.js`.

```
sh
npm run pkg
```

- 2. Run from Command Prompt:**

- Open Command Prompt.
- Navigate to your project directory.
- Run the executable:

```
sh
taplist-project.exe
```

- 3. Check Logs:**

- Open the system's temporary directory. You can find it by running `%TEMP%` in the Run dialog (Win + R).
- Look for the `server.log` file in the temporary directory and open it to see the logs.

- 4. Test Admin Functionality:**

- Open a browser, navigate to `http://localhost:3000/admin.html`, log in, and try to save changes.
- Verify that logs are correctly captured in `server.log`.
- Ensure the changes are applied correctly without HTML in the alert.

This should address the issues and provide a clear logging mechanism to help troubleshoot further if needed.



Instead of using pkg to build the exe can we use NW.js



Yes, you can use NW.js to build the executable for your project. NW.js (previously known as node-webkit) allows you to build desktop applications with web technologies by embedding a web browser engine (Chromium) and Node.js runtime into a single executable.

Steps to Build an Executable with NW.js

- 1. Install NW.js:** First, install NW.js. You can download the appropriate version for your platform from the [NW.js website](#).
- 2. Create Project Structure:** Set up your project structure to include an `index.html` file as the entry point, along with your Node.js server and other resources.
- 3. Create `package.json` for NW.js:** Create a `package.json` file specifically for NW.js that includes the main entry point and window configuration.
- 4. Bundle the Application:** Use NW.js to run and package your application.

Detailed Steps

1. Install NW.js

Download and extract the appropriate version of NW.js for your platform from the [NW.js downloads page](#).

2. Project Structure

Your project structure should look something like this:

```
java
my-nwjs-app/
├── package.json
├── index.html
└── server.js
└── public/
    ├── admin.html
    ├── color-admin.html
    ├── index.html
    ├── styles.css
    ├── script.js
    ├── beers.json
    ├── colors.json
    └── banner.jpg
credentials.json
```

3. Create `package.json` for NW.js

Create a `package.json` file in the root of your project directory with the following content:

json

```
{
  "name": "taplist-project",
  "main": "index.html",
  "window": {
    "title": "Tap List Project",
    "icon": "public/banner.jpg",
    "toolbar": false,
    "frame": true,
    "width": 800,
    "height": 600
  },
  "dependencies": {
    "body-parser": "^1.20.2",
    "express": "^4.19.2",
    "express-session": "^1.18.0",
    "googleapis": "^137.1.0"
  },
  "scripts": {
    "start": "node server.js"
  }
}
```

This `package.json` file specifies the main entry point for NW.js (`index.html`) and configures the window properties.

4. Create `index.html`

Create an `index.html` file in the root of your project directory. This file will load your Node.js server and open your main application window. For example:

html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Tap List Project</title>
  <script>
    const nw = require('nw.gui');
    const path = require('path');
    const { exec } = require('child_process');
    const server = exec('node server.js', { cwd: path.resolve(__dirname) });

    server.stdout.on('data', data => {
      console.log(`stdout: ${data}`);
    });

    server.stderr.on('data', data => {
      console.error(`stderr: ${data}`);
    });

    server.on('close', code => {
      console.log(`child process exited with code ${code}`);
    });

    nw.Window.get().show();
  </script>
</head>
<body>
  <h1>Tap List Project</h1>
  <p>The server is running. Open <a href="http://localhost:3000/admin.html">Admin Page</a>.

```

```
</p>
</body>
</html>
```

This `index.html` file starts your Node.js server when the NW.js application is launched.

5. Run the Application

Navigate to your project directory in the command prompt or terminal and run NW.js with your project directory:

```
sh
/path/to/nw my-nwjs-app/
```

Replace `/path/to/nw` with the path to the NW.js executable you downloaded.

6. Package the Application

To create a distributable package, you can use tools like `nw-builder`:

1. Install `nw-builder`:

```
sh
npm install -g nw-builder
```

2. Package the Application:

```
sh
nwbuild -v 0.50.0 -p win64,win32,osx64,linux64 my-nwjs-app/
```

Replace `0.50.0` with the version of NW.js you are using and adjust the platforms (`-p`) as needed.

This will create executables for the specified platforms.

Example `server.js`

Make sure your `server.js` is set up as described previously, including the minimal logging.

```
javascript
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
```

```

const fs = require('fs');
const session = require('express-session');
const os = require('os');
const { google } = require('googleapis');
const credentials = require('./credentials.json'); // Ensure this is the correct path

const app = express();
const PORT = 3000;
const ADMIN_PASSWORD = 'your_secure_password'; // Replace with your actual secure password
const BEERS_FILE_PATH = path.join(__dirname, 'public', 'beers.json');
const COLORS_FILE_PATH = path.join(__dirname, 'public', 'colors.json');
const DOC_ID = '1T1HqTNp-VH3ZTIEbi407NUJltII9LNGZc8yXNz6v3NQ'; // Your Google Doc ID
const LOG_FILE_PATH = path.join(os.tmpdir(), 'server.log');

// Function to log messages to a file
function logToFile(message) {
    const timestamp = new Date().toISOString();
    fs.appendFileSync(LOG_FILE_PATH, `[${timestamp}] ${message}\n`, 'utf8');
}

// Override console methods
console.log = (message) => logToFile(message);
console.error = (message) => logToFile(`ERROR: ${message}`);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(session({
    secret: 's2K7m1!xY9t#Vw$3b4E%Fg@8RkLpQ', // Secure random string
    resave: false,
    saveUninitialized: true
}));

// Middleware to check if user is logged in
const isAuthenticated = (req, res, next) => {
    if (req.session.loggedIn) {
        next();
    } else {
        res.redirect('/login.html');
    }
};

// Serve static files from public directory, but protect admin.html and color-admin.html
app.use((req, res, next) => {
    if (req.path === '/admin.html' || req.path === '/color-admin.html') {
        isAuthenticated(req, res, next);
    } else {
        next();
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.post('/login', (req, res) => {
    const { password } = req.body;
    if (password === ADMIN_PASSWORD) {
        req.session.loggedIn = true;
        res.redirect('/admin.html');
    } else {
        res.send('Incorrect password');
    }
});

app.get('/admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'admin.html'));
});

app.get('/color-admin.html', isAuthenticated, (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'color-admin.html'));
});

app.get('/beers', (req, res) => {
    fs.readFile(BEERS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading beers file:', err);
            logToFile(`Error reading beers file: ${err}`);
        }
    });
});

```

```

        res.status(500).send('Error reading beers file');
    } else {
        res.json(JSON.parse(data));
    }
});
});

app.post('/save-beers', isAuthenticated, (req, res) => {
    const updatedBeers = req.body;
    console.log(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    logToFile(`Received request to save beers: ${JSON.stringify(updatedBeers)}`);
    fs.writeFile(BEERS_FILE_PATH, JSON.stringify(updatedBeers, null, 2), (err) => {
        if (err) {
            console.error('Failed to save beers:', err);
            logToFile(`Failed to save beers: ${err}`);
            res.status(500).send('Failed to save beers');
        } else {
            console.log('Beers saved successfully');
            logToFile('Beers saved successfully');
            res.send('Beers saved successfully');
        }
    });
});

app.get('/colors', (req, res) => {
    fs.readFile(COLORS_FILE_PATH, (err, data) => {
        if (err) {
            console.error('Error reading colors file:', err);
            logToFile(`Error reading colors file: ${err}`);
            res.status(500).send('Error reading colors file');
        } else {
            res.json(JSON.parse(data));
        }
    });
});

app.post('/save-colors', isAuthenticated, (req, res) => {
    const updatedColors = req.body;
    console.log(`Received request to save colors: ${JSON.stringify(updatedColors)}`);
    logToFile(`Received request to save colors: ${JSON.stringify(updatedColors)}`);
    fs.writeFile(COLORS_FILE_PATH, JSON.stringify(updatedColors, null, 2), (err) => {
        if (err) {
            console.error('Failed to save colors:', err);
            logToFile(`Failed to save colors: ${err}`);
            res.status(500).send('Failed to save colors');
        } else {
            console.log('Colors saved successfully');
            logToFile('Colors saved successfully');
            res.send('Colors saved successfully');
        }
    });
});

app.get('/ip', (req, res) => {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {
            if (i.family === 'IPv4' && !i.internal) {
                localIp = i.address;
                break;
            }
        }
    }
    res.json({ ip: localIp });
});

// Send the local IP address to Google Docs
async function sendIpToGoogleDocs() {
    const networkInterfaces = os.networkInterfaces();
    let localIp = '';
    for (const interfaceName in networkInterfaces) {
        const interfaces = networkInterfaces[interfaceName];
        for (const i of interfaces) {

```

```

        if (i.family === 'IPv4' && !i.internal) {
            localIp = i.address;
            break;
        }
    }

    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
        file path
        scopes: ['https://www.googleapis.com/auth/documents']
    });

    const docs = google.docs({ version: 'v1', auth });

    // Get the current time in CST
    const now = new Date();
    const cstOffset = -6; // CST is UTC-6
    const cstTime = new Date(now.getTime() + cstOffset * 60 * 60 * 1000);
    const formattedCstTime = cstTime.toISOString().replace('T', ' ').replace(/\..+/, ''). + 'CST';

    const url = `http://${localIp}:3000/admin.html`;
    const textToInsert = `Click to edit taplist\nTimestamp: ${formattedCstTime}\n\n`;

    try {
        // Get the document to determine its end index
        const doc = await docs.documents.get({ documentId: DOC_ID });
        const endIndex = doc.data.body.content.reduce((maxIndex, element) => {
            return element.endIndex > maxIndex ? element.endIndex : maxIndex;
        }, 1);

        // Adjust the endIndex to exclude the newline character
        const adjustedEndIndex = endIndex - 1;

        // Clear existing content
        await docs.documents.batchUpdate({
            documentId: DOC_ID,
            requestBody: {
                requests: [
                    {
                        deleteContentRange: {
                            range: {
                                startIndex: 1,
                                endIndex: adjustedEndIndex,
                            },
                        },
                    },
                ],
            },
        });
    }

    // Insert new content
    const response = await docs.documents.batchUpdate({
        documentId: DOC_ID,
        requestBody: {
            requests: [
                {
                    insertText: {
                        location: {
                            index: 1,
                        },
                        text: textToInsert,
                    },
                },
                {
                    updateTextStyle: {
                        range: {
                            startIndex: 1,
                            endIndex: 22, // Adjust this range to match the length of
                            "Click to edit taplist" (21 characters + 1 to cover the full length)
                        },
                        textStyle: {
                            link: {
                                url: url,
                            }
                        }
                    }
                }
            ]
        }
    });
}

```

```

        },
        fontSize: {
            magnitude: 32, // Twice the original font size (16 * 2)
            unit: 'PT',
        },
        fields: 'link,fontSize',
    },
],
},
),
);
console.log('IP address sent to Google Docs successfully:', response.data);
} catch (error) {
    console.error('Failed to send IP address to Google Docs:', error);
}
}

app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server is running on http://localhost:${PORT}`);
    sendIpToGoogleDocs(); // Send IP address at startup
    setInterval(sendIpToGoogleDocs, 3600000); // Send IP address every hour
});

```

Running the NW.js Application

1. Run from Command Prompt:

- o Open Command Prompt.
- o Navigate to your project directory.
- o Run NW.js with your project directory:

```

sh
path\to\nw.exe my-nwjs-app

```

2. Verify the Application:

The application should open with a window showing your `index.html` content. The Node.js server should start, and you should be able to access the admin panel and other routes.

By following these steps, you can create an NW.js application that embeds your Node.js server and provides a desktop application experience. This approach should resolve the issues you encountered with `pkg` and provide a more robust solution.



Is there a GUI program that can turn my node project into an exe



Yes, there are several GUI tools that can help you turn your Node.js project into an executable (exe) file. One popular tool is **Electron**, which has a rich ecosystem and tools to help package your application. Electron allows you to build cross-platform desktop applications with JavaScript, HTML, and CSS.

To make the process simpler, you can use Electron's GUI tools like **Electron Forge** or **Electron Packager**. Here's a guide on how to use Electron Forge with a GUI to package your Node.js application:

Using Electron with Electron Forge

Step 1: Install Electron Forge

First, you need to install Electron Forge. Electron Forge provides a GUI tool to manage and package Electron applications.

```
bash
npm install -g @electron-forge/cli
```

Step 2: Create a New Electron Project

Create a new Electron project using Electron Forge. This step sets up a new project with the necessary configuration.

```
bash
electron-forge init my-app
```

Navigate to your project directory:

```
bash
cd my-app
```

Step 3: Add Your Node.js Code

Copy your existing Node.js files (``server.js``, `public` directory, etc.) into the `my-app` directory. Make sure your main entry point for Electron is correctly set up.

Step 4: Update `package.json`

Edit the `package.json` file in your Electron project to include the necessary configurations. Your `package.json` might look something like this:

```
json
{
  "name": "taplist-project",
  "version": "1.0.0",
  "description": "A Tap List Project",
  "main": "main.js",
  "scripts": {
```

```

"start": "electron-forge start",
"package": "electron-forge package",
"make": "electron-forge make"
},
"dependencies": {
  "electron-squirrel-startup": "^1.0.0",
  "express": "^4.19.2",
  "body-parser": "^1.20.2",
  "express-session": "^1.18.0",
  "googleapis": "^137.1.0"
},
"devDependencies": [
  "@electron-forge/cli": "^6.0.0-beta.54",
  "@electron-forge/maker-squirrel": "^6.0.0-beta.54",
  "@electron-forge/maker-zip": "^6.0.0-beta.54",
  "@electron-forge/maker-deb": "^6.0.0-beta.54",
  "@electron-forge/maker-rpm": "^6.0.0-beta.54"
},
"config": {
  "forge": {
    "packagerConfig": {},
    "makers": [
      {
        "name": "@electron-forge/maker-squirrel",
        "config": {}
      },
      {
        "name": "@electron-forge/maker-zip",
        "platforms": [
          "darwin"
        ]
      },
      {
        "name": "@electron-forge/maker-deb",
        "config": {}
      },
      {
        "name": "@electron-forge/maker-rpm",
        "config": {}
      }
    ]
  }
}
}

```

Step 5: Create `main.js`

Create a `main.js` file in the root of your project to start your Node.js server and open a browser window. Here's an example `main.js` file:

```

javascript

const { app, BrowserWindow } = require('electron');
const path = require('path');
const { exec } = require('child_process');

function createWindow () {
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
      contextIsolation: false,
    }
  });

  mainWindow.loadFile('index.html');
}

```

```
// Open the DevTools (optional)
// mainWindow.webContents.openDevTools();

// Start the server
const server = exec('node server.js', { cwd: path.resolve(__dirname) });

server.stdout.on('data', data => {
  console.log(`stdout: ${data}`);
});

server.stderr.on('data', data => {
  console.error(`stderr: ${data}`);
});

server.on('close', code => {
  console.log(`child process exited with code ${code}`);
});
}

app.whenReady().then(() => {
  createWindow();

  app.on('activate', function () {
    if (BrowserWindow.getAllWindows().length === 0) createWindow();
  });
});

app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') app.quit();
});
```

Step 6: Package the Application

Run the following command to package your application:

```
bash
npm run package
```

This command will package your application and create an executable in the `out` directory.

Using Electron Packager

Another option is to use Electron Packager directly:

1. Install Electron Packager:

```
bash
npm install electron-packager --save-dev
```

2. Add Package Script:

Update the `scripts` section in your `package.json` to include a packaging command:

json

```
"scripts": {
  "start": "electron .",
  "package": "electron-packager . --out=out --overwrite --platform=win32 --arch=x64 --
prune=true --electron-version=12.0.0"
}
```

3. Run the Packaging Command:

bash

```
npm run package
```

This command will package your application into an executable in the `out` directory.

Summary

By using Electron Forge or Electron Packager, you can convert your Node.js project into an executable file with a GUI. Both tools offer straightforward ways to package your application, and Electron Forge provides additional features and a more streamlined process with a graphical interface.



Can we automate the Google Cloud Console setup process as outlined below:

"""Step-by-Step Instructions for Setting Up Google Docs API": "",

"Step 1: Create a Google Account": "1. Go to Google Sign Up: <https://accounts.google.com/signup> | 2. Follow the instructions to create a new account.",

"Step 2: Enable Google Docs API": "1. Go to the Google Cloud Console:

<https://console.cloud.google.com/> | 2. Click on the project dropdown at the top left and select 'New Project.' | 3. Name your project and click 'Create.' | 4. Once the project is created, make sure it's selected. Then, go to the API & Services Dashboard:

<https://console.cloud.google.com/apis/dashboard> | 5. Click 'Enable APIs and Services.' | 6. Search for 'Google Docs API' and select it. | 7. Click 'Enable.'",

"Step 3: Create Service Account Credentials": "1. In the Google Cloud Console, navigate to 'APIs & Services' > 'Credentials.' | 2. Click 'Create Credentials' and select 'Service Account.' | 3. Fill in the necessary information: | - Service account name: docs-api-service-account | - Service account description: Service account for Google Docs API | 4. Click 'Create and Continue.' | 5. Assign a role to the service account (e.g., 'Editor'). | 6. Click 'Done.' | 7. In the 'Credentials' tab, find your new service account and click on it. | 8. Go to the 'Keys' tab. | 9. Click 'Add Key' > 'Create New Key.' | 10. Choose 'JSON' and click 'Create.' This will download a JSON file with your credentials.",

"Step 4: Share the Google Doc with the Service Account": "1. Open Google Docs: <https://docs.google.com/> and create a new document. | 2. Note the Document ID from the URL (it's the long string after /d/ in the document URL). | 3. Click 'Share' in the top-right corner of the document. | 4. In the 'Share with people and groups' dialog, enter the email address of your service account. It should look something like your-service-account@your-project-id.iam.gserviceaccount.com. | 5. Give the service account 'Editor' access and click 'Send.'",

"Step 5: Set Up Your Project to Use the Credentials": "1. Place the downloaded JSON credentials file in your project directory. Ensure the file is named appropriately, for example, credentials.json. | 2. Update your server.js file to use the correct path to the credentials file. | | Here's a snippet to add to your server.js for reference: | const { google } = require('googleapis'); | const path = require('path'); | const credentials = require('./credentials.json'); // Ensure this is the correct path | | async function sendIpToGoogleDocs() { | | | const auth = new google.auth.GoogleAuth({ | keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials file path | scopes: ['https://www.googleapis.com/auth/documents'] | | |}); | | | const docs = google.docs({ | version: 'v1', auth }); | | // The rest of your function logic... | |",

"Step 6: Testing the Setup": "1. Run your server script using node server.js. | 2. Ensure there are no errors and that the IP address is being sent to your Google Doc.",

"Summary": "By following these steps, you'll have set up a Google Docs API with a new Google account, created service account credentials, and shared your Google Doc with the service account to allow editing. This enables your project to send updates to the Google Doc using the sendIpToGoogleDocs function. | | If you encounter any issues or need further assistance, feel free to ask!""



To automate the Google Cloud Console setup process as outlined, you can use a combination of the Google Cloud SDK (gcloud), APIs, and scripts to perform these steps programmatically. Below is a step-by-step guide to automate each step using the `gcloud` CLI and APIs.

Prerequisites

1. Install the [Google Cloud SDK](#).
2. Ensure you have Python installed (required for some Google Cloud SDK functionalities).

Step-by-Step Automation Script

1. Create a Google Account (Manual Step)

Unfortunately, creating a Google Account programmatically isn't supported due to security and privacy reasons. You'll need to create a Google Account manually.

2. Enable Google Docs API

bash

```
#!/bin/bash

# Set variables
PROJECT_NAME="taplist-project"
PROJECT_ID="taplist-project-$(date +%s)"
SERVICE_ACCOUNT_NAME="docs-api-service-account"
KEY_FILE="credentials.json"

# Authenticate with Google Cloud
gcloud auth login

# Create a new project
gcloud projects create $PROJECT_ID --name=$PROJECT_NAME

# Set the project
gcloud config set project $PROJECT_ID

# Enable Google Docs API
gcloud services enable docs.googleapis.com
```

3. Create Service Account Credentials

bash

```
# Create a service account
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME \
    --description="Service account for Google Docs API" \
    --display-name="docs-api-service-account"

# Assign a role to the service account
gcloud projects add-iam-policy-binding $PROJECT_ID \
    --member="serviceAccount:$SERVICE_ACCOUNT_NAME@$PROJECT_ID.iam.gserviceaccount.com" \
    --role="roles/editor"

# Create and download a JSON key for the service account
gcloud iam service-accounts keys create $KEY_FILE \
    --iam-account=$SERVICE_ACCOUNT_NAME@$PROJECT_ID.iam.gserviceaccount.com
```

4. Share the Google Doc with the Service Account (Manual Step)

1. Manually create a new Google Doc.
2. Note the Document ID from the URL.
3. Share the document with the service account email: `your-service-account@\$PROJECT_ID.iam.gserviceaccount.com`.

5. Set Up Your Project to Use the Credentials

javascript

```
// Ensure the credentials.json file is in your project directory
const { google } = require('googleapis');
const path = require('path');
const credentials = require('./credentials.json'); // Ensure this is the correct path

async function sendIpToGoogleDocs() {
    const auth = new google.auth.GoogleAuth({
        keyFile: path.join(__dirname, 'credentials.json'), // Replace with your credentials
```

```

file path
    scopes: ['https://www.googleapis.com/auth/documents']
});

const docs = google.docs({ version: 'v1', auth });

// The rest of your function logic...
}

```

6. Testing the Setup

```

bash

# Run your server script
node server.js

# Ensure there are no errors and that the IP address is being sent to your Google Doc

```

Complete Bash Script

Here is the complete bash script combining all steps:

```

bash

#!/bin/bash

# Set variables
PROJECT_NAME="taplist-project"
PROJECT_ID="taplist-project-$(date +%)"
SERVICE_ACCOUNT_NAME="docs-api-service-account"
KEY_FILE="credentials.json"

# Authenticate with Google Cloud
gcloud auth login

# Create a new project
gcloud projects create $PROJECT_ID --name=$PROJECT_NAME

# Set the project
gcloud config set project $PROJECT_ID

# Enable Google Docs API
gcloud services enable docs.googleapis.com

# Create a service account
gcloud iam service-accounts create $SERVICE_ACCOUNT_NAME \
--description="Service account for Google Docs API" \
--display-name="docs-api-service-account"

# Assign a role to the service account
gcloud projects add-iam-policy-binding $PROJECT_ID \
--member="serviceAccount:$SERVICE_ACCOUNT_NAME@$PROJECT_ID.iam.gserviceaccount.com" \
--role="roles/editor"

# Create and download a JSON key for the service account
gcloud iam service-accounts keys create $KEY_FILE \
--iam-account=$SERVICE_ACCOUNT_NAME@$PROJECT_ID.iam.gserviceaccount.com

echo "Service account created and credentials saved to $KEY_FILE"
echo "Manually share the Google Doc with the service account email:
$SERVICE_ACCOUNT_NAME@$PROJECT_ID.iam.gserviceaccount.com"

```

Summary

- **Automated Steps:** Project creation, API enabling, service account creation, and key generation.
- **Manual Steps:** Google account creation and sharing the Google Doc with the service account.

By following these steps, you can automate most of the setup process for the Google Docs API. For the manual steps, make sure to follow the instructions provided. If you encounter any issues, refer to the Google Cloud documentation or ask for further assistance.

ChatGPT can make mistakes. Check important info.