

EECS 4481 - Project Phase 3

By: Derui Liu, Harsh Patel, Jihal Patel, and Lukas Rose

Table of Contents

1. Penetration Testing	1
1.1 Weak Passwords	1
2. Static Code Analysis Test	2
2.1 Cleartext submission of password	2
Vulnerability Description	2
Solution	2
Testing Solution	3
2.2 Password field with autocomplete enabled	3
Vulnerability Description	3
Solution	4
Testing Solution	4
2.3 Unencrypted communications	4
Vulnerability Description	4
Solution	5
Testing Solution	5
3. Additional Requirements	6
3.1 File Upload Page	6
Vulnerability Testing	7
3.2 Session Control	8
Cookie Employed	8
Session Control in Action:	9
For Anonymous Users:	9
For Admin Users:	12
4. Source Code	16

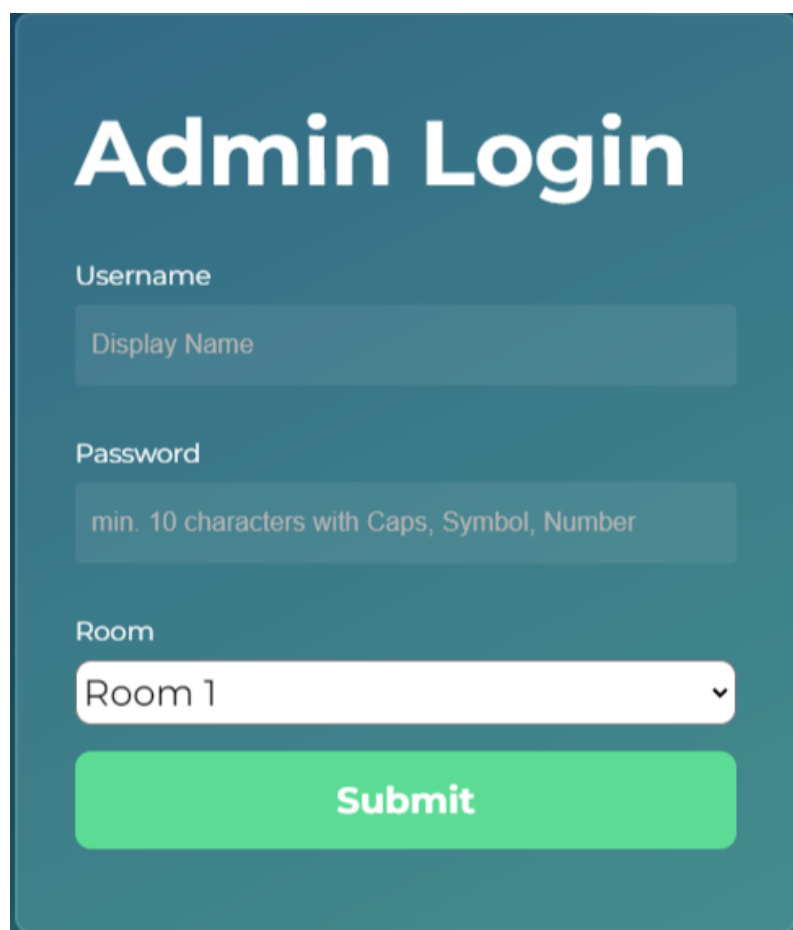
1. Penetration Testing

In phase 2 we performed Penetration Testing using Burp Suite and discovered two vulnerabilities, Weak Passwords and Tentative CSRF vulnerability. In this part, we present our solution to these vulnerabilities. After patching, these vulnerabilities are longer present.

1.1 Weak Passwords

In Phase 2 we used Hydra to perform dictionary attacks on our web login portal and discovered that weak passwords (i.e. passwords with less than 8 letters and no numbers, symbols etc) made our portal susceptible to dictionary attacks.

In order to rectify this we decided to rectify the passwords policy, where the user needs to have at least 10 letter passwords with a good mix of capitals, symbols and numbers.



The image shows a web form titled "Admin Login" on a dark teal background. The form contains three input fields: "Username" with a placeholder "Display Name", "Password" with a placeholder "min. 10 characters with Caps, Symbol, Number", and "Room" with a dropdown menu showing "Room 1". A green "Submit" button is at the bottom.

Admin Login

Username
Display Name

Password
min. 10 characters with Caps, Symbol, Number

Room
Room 1

Submit

2. Static Code Analysis Test

In phase 2 we performed Static Code Analysis using Burp Suite and discovered three vulnerabilities. In this part, we present our solution to these vulnerabilities. After patching, Burp Suite no longer showed these vulnerabilities.

2.1 Cleartext submission of password

Vulnerability Description

	Severity:	High
	Confidence:	Certain
	Host:	http://localhost:3000
	Path:	/adminform.html

Issue detail

The page contains a form with the following action URL, which is submitted over clear-text HTTP:

- `http://localhost:3000/adminLogin`

The form contains the following password field:

- `password`

The Password is being sent to the database for authentication in clear/plain text. This is a big vulnerability because an attacker may eavesdrop on the client's network traffic and obtain the client's credentials.

Solution

We decided to **hash** our passwords using the SHA-256 Algorithm. This way, whenever the user submits credentials for authentication, the password will be hashed and a potential attacker can not decrypt it. Similarly, we also only stored hashes of the password in our database.

We also decided to implement **HTTPS** so that communication between the client and the server is encrypted and can not be eavesdropped.


Testing Solution

Screenshot of hash

```
admin login page for credentials!!  
These are the credentials requested by the admin -->  
username: test password: 1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014
```

2.2 Password field with autocomplete enabled

Vulnerability Description

	Severity:	Low
	Confidence:	Certain
	Host:	http://localhost:3000
	Path:	/adminform.html

Issue detail

The page contains a form with the following action URL:

- <http://localhost:3000/adminLogin>

The form contains the following password field with autocomplete enabled:

- password

Most browsers have a functionality to remember user credentials that are entered into HTML forms. If the auto fill function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.

The stored credentials can be captured by an attacker who gains control over the user's computer. Furthermore, this vulnerability can also be exploited by an attacker who finds a cross-site scripting vulnerability to retrieve a user's browser-stored credentials.

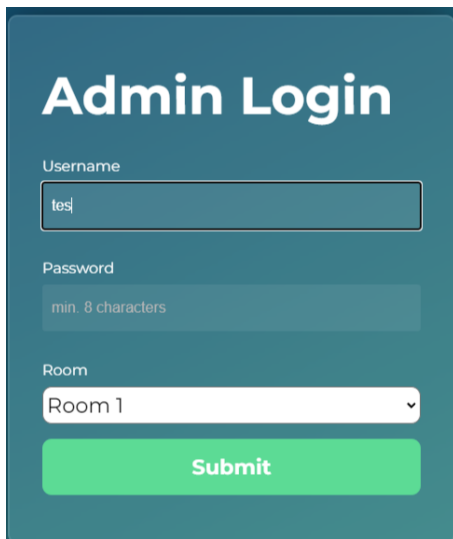
Solution

To prevent browsers from storing credentials entered into HTML forms, the attribute **autocomplete="off"** should be included within the form tag (to protect all form fields) or in input tags (to protect specific individual fields).

```
<h1>Admin Login</h1>
<form action="/adminLogin" method="POST" autocomplete="off">
  <label>Username</label>
  <input type="text" placeholder="Display Name" name="username" required>
  <label>Password</label>
  <input type="password" placeholder="min. 8 characters" name="password" required>
```

Testing Solution

Admin form page : Suggestions for username no longer show up. Auto fill doesn't work.

A screenshot of a web form titled "Admin Login" on a teal background. The form contains three input fields: "Username" with the text "tes" entered, "Password" with the placeholder "min. 8 characters", and "Room" with a dropdown menu showing "Room 1". A green "Submit" button is at the bottom.

2.3 Unencrypted communications

Vulnerability Description

The application HelpDeskApp allows users to connect to it over unencrypted connections (HTTP). An attacker eavesdropping on clients' network traffic could record and monitor their interactions with the application and obtain all the information the client provides.

Solution

The application should use transport-level encryption (SSL or TLS) to protect all sensitive communications passing between the client and the server.

So we decided to remove HTTP and instead employ HTTPS with a private key and certificate. HTTPS employs TLS which protects traffic between the client and server. Our current certificate is self signed. This is because CA authorities usually need a website to be on the internet before giving a certificate. Once we have our website hosted we can easily change the certificate.

We also went a step forward and created a redirect from http to https. So that an attacker cannot abuse it.

```
// HTTPS CONFIG
var fs = require('fs');
var https = require('https');
var privateKey = fs.readFileSync(path.resolve('key.pem'));
var certificate = fs.readFileSync(path.resolve('cert.pem'));

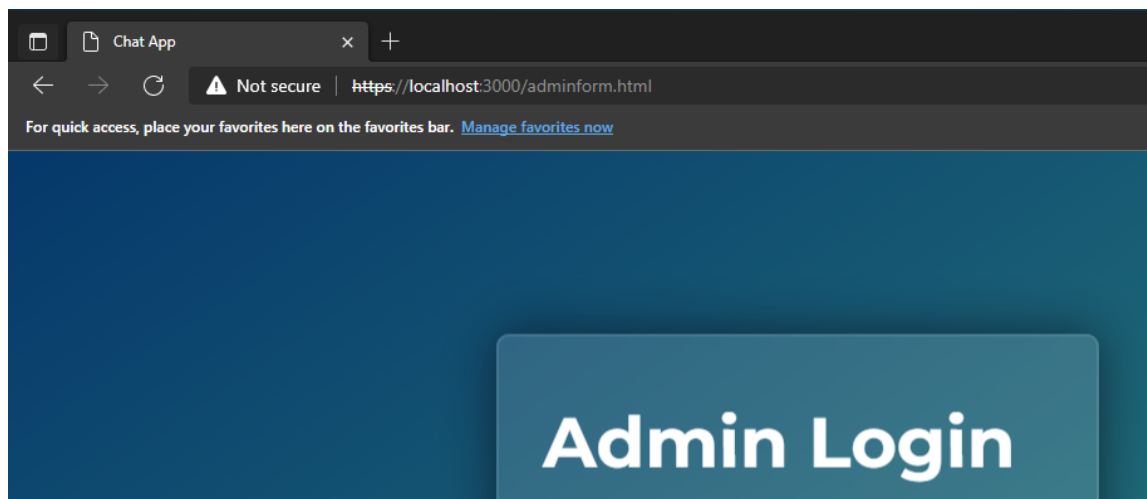
var credentials = {key: privateKey, cert: certificate};
var express = require('express');
var app = express();
app.enable('trust proxy')

// your express configuration here

var httpsServer = https.createServer(credentials, app);
```

Testing Solution

HTTPS website address in browser (due to self signed certificate it shows not secure)



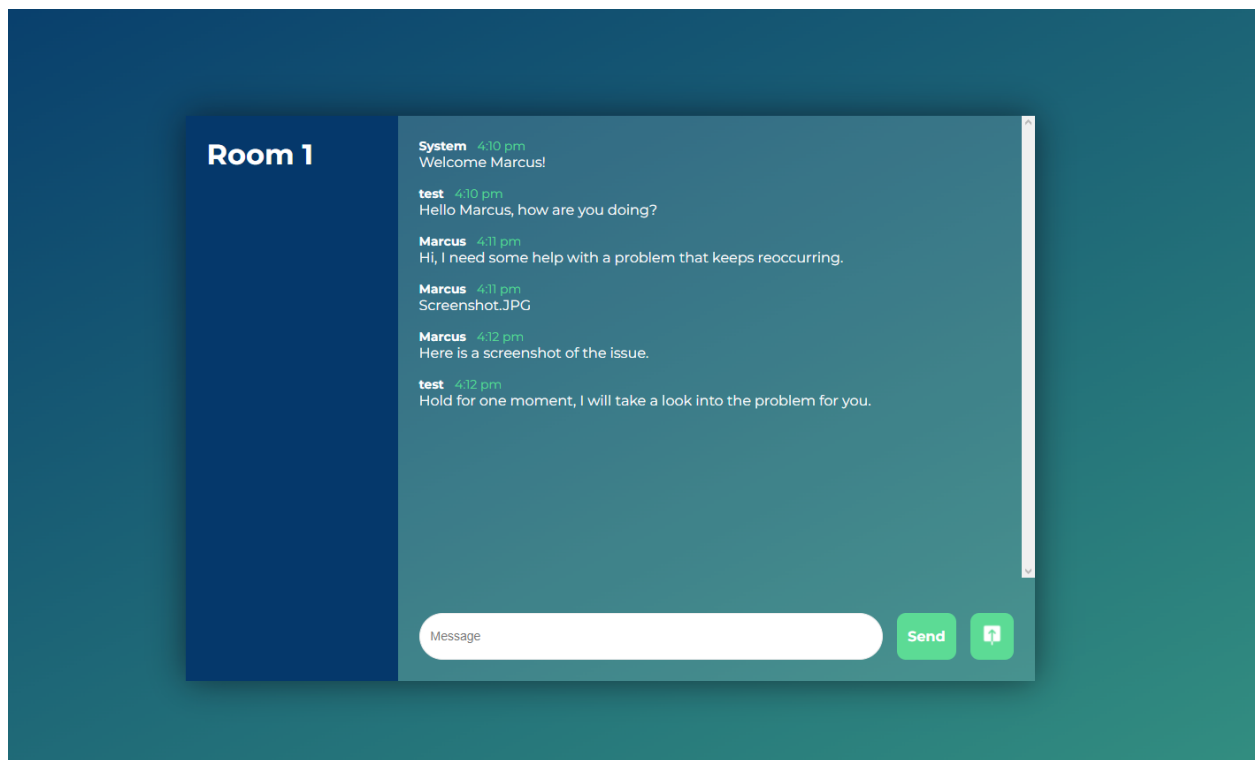
3. Additional Requirements

3.1 File Upload Page

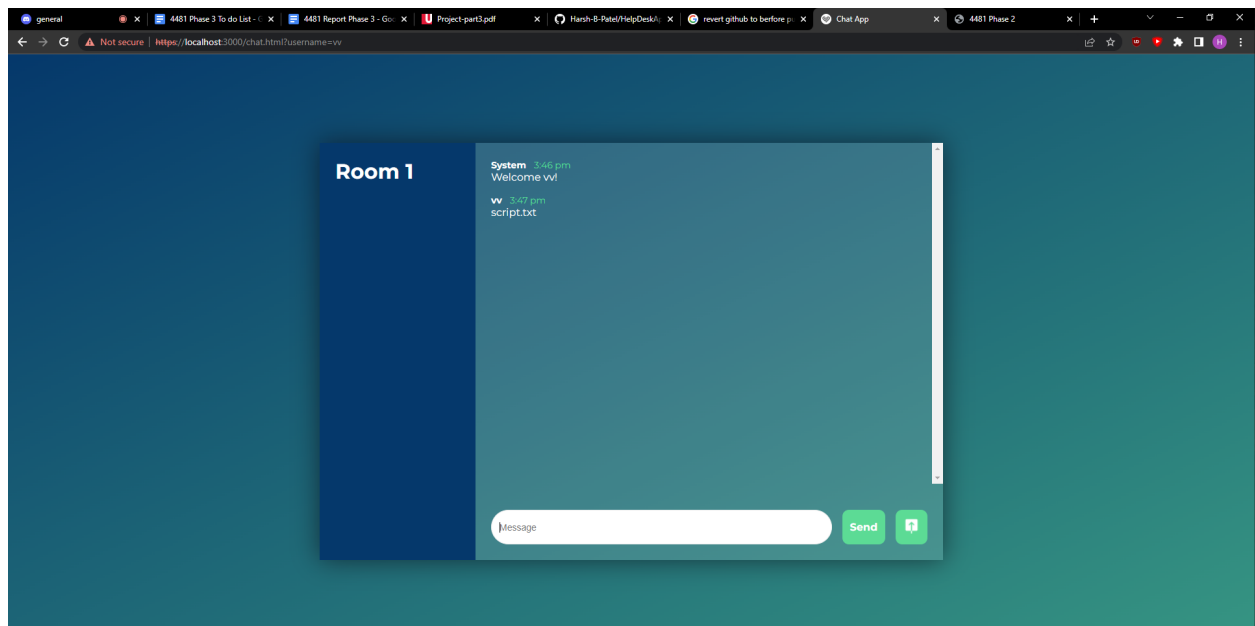
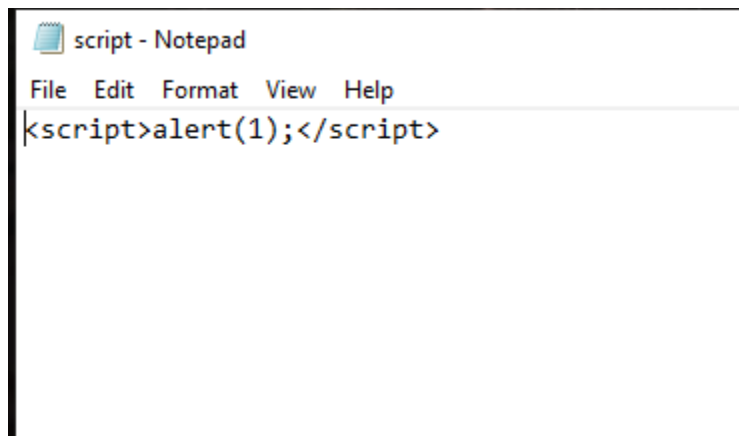
During Phase 3 we added various features to our application which both improved security and usability aspects. A fundamental addition in this phase is the feature for a user to upload a file into the chat room.

The feature is an added button which, upon clicking, opens the user's file explorer to select a file which they wish to upload to the HelpDeskApp. Upon selection the file automatically gets added to the chat.

This feature can be seen in action in the below screenshot.



Vulnerability Testing

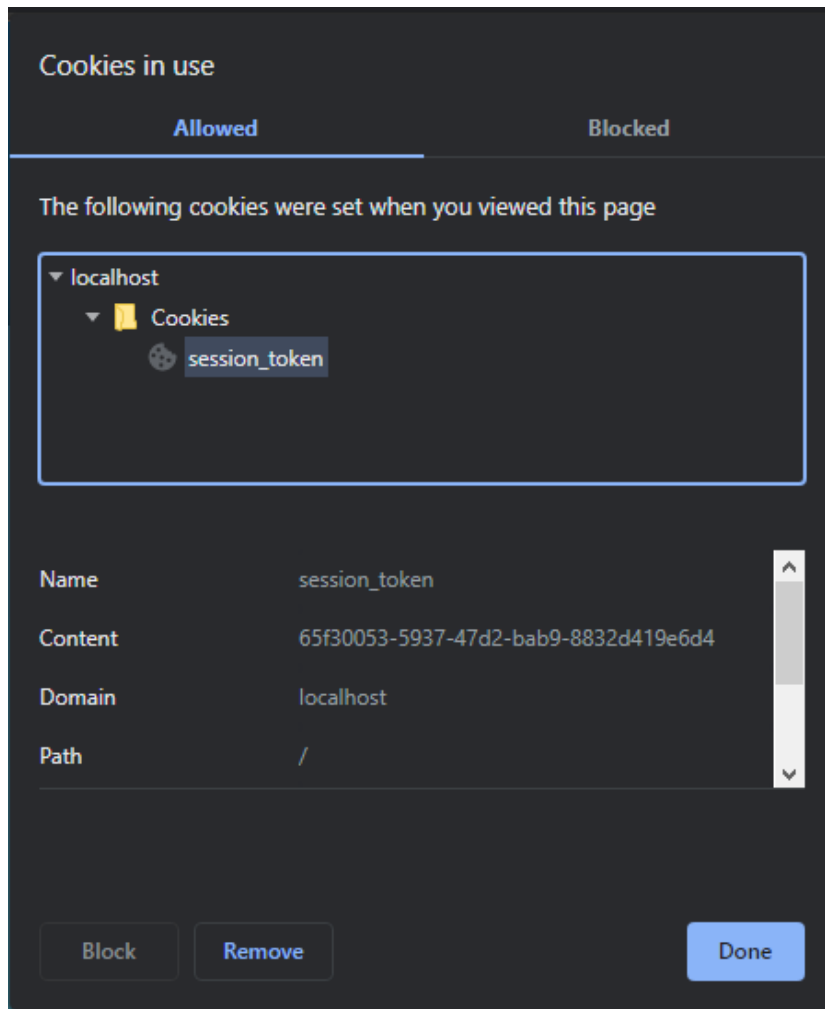


Our team tested the new feature of uploading files by trying to upload a file that would be deemed malicious. The file above is a script file which upon execution would open an alert on the screen.

3.2 Session Control

For Session control we employed the use of session cookies. We created a cookie called session-token to store a unique token to track active sessions and allow returning users to skip the login/registration pages:

Cookie Employed



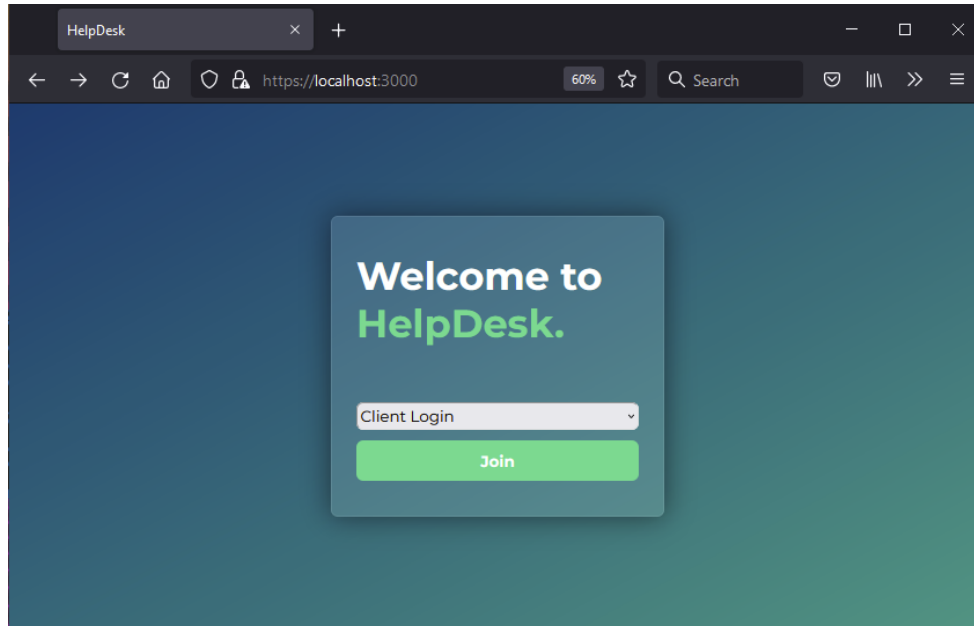
For security, we chose to have the session cookie contain only a unique randomly-generated identifier (UUID) as a token. All other session information (username, admin status, session expiry etc.) is stored on the server and is inaccessible to the user. Session information is evaluated only when a cookie with a valid token is provided. This prevents malicious users from modifying their cookies to impersonate another user, or gain unauthorized access to admin-only parts of the application.

If the token provided is not valid, the user will be redirected to the login/register screen, and if the token is valid, but the session is expired, the session will be deleted, and then the user will be redirected to the login/register screen.

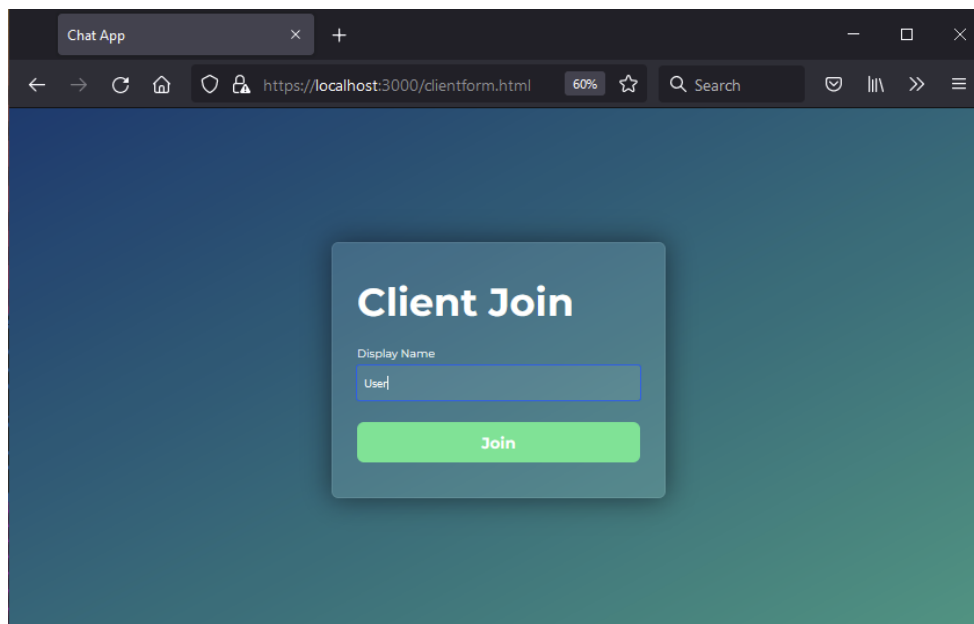
Session Control in Action:

For Anonymous Users:

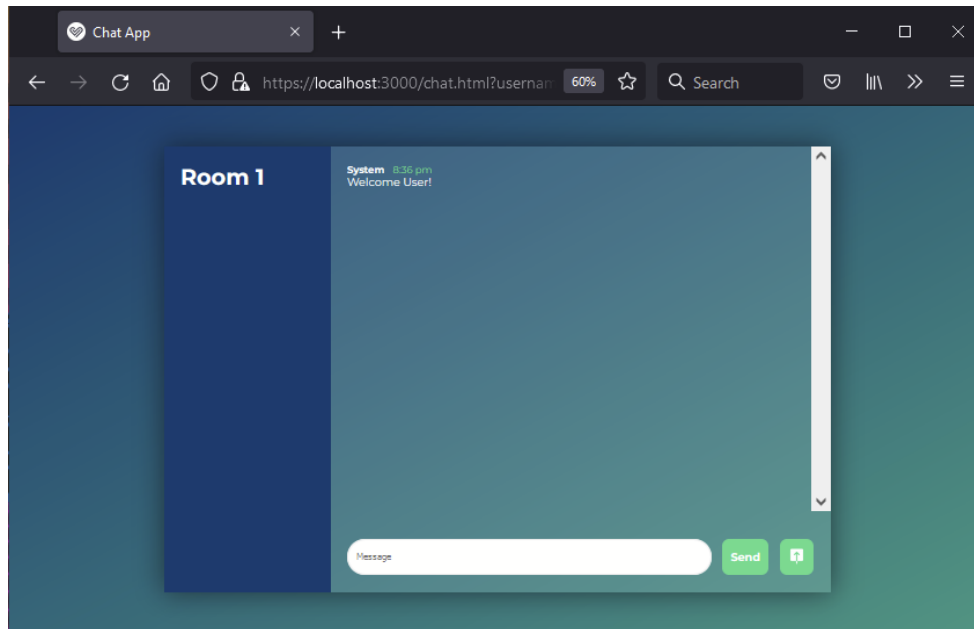
A new anonymous user visiting the site for the first time:



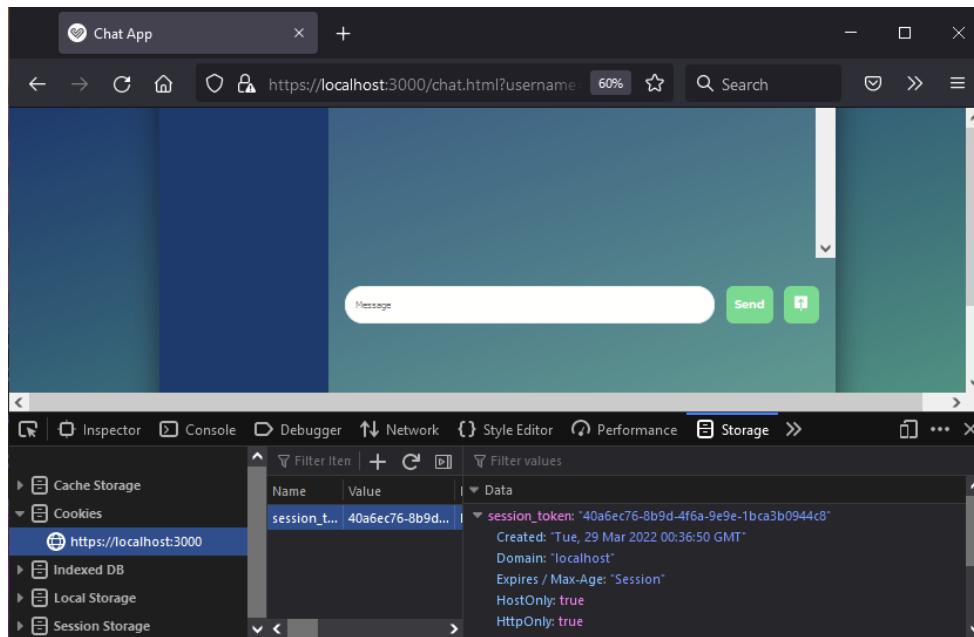
Registering as a new user:



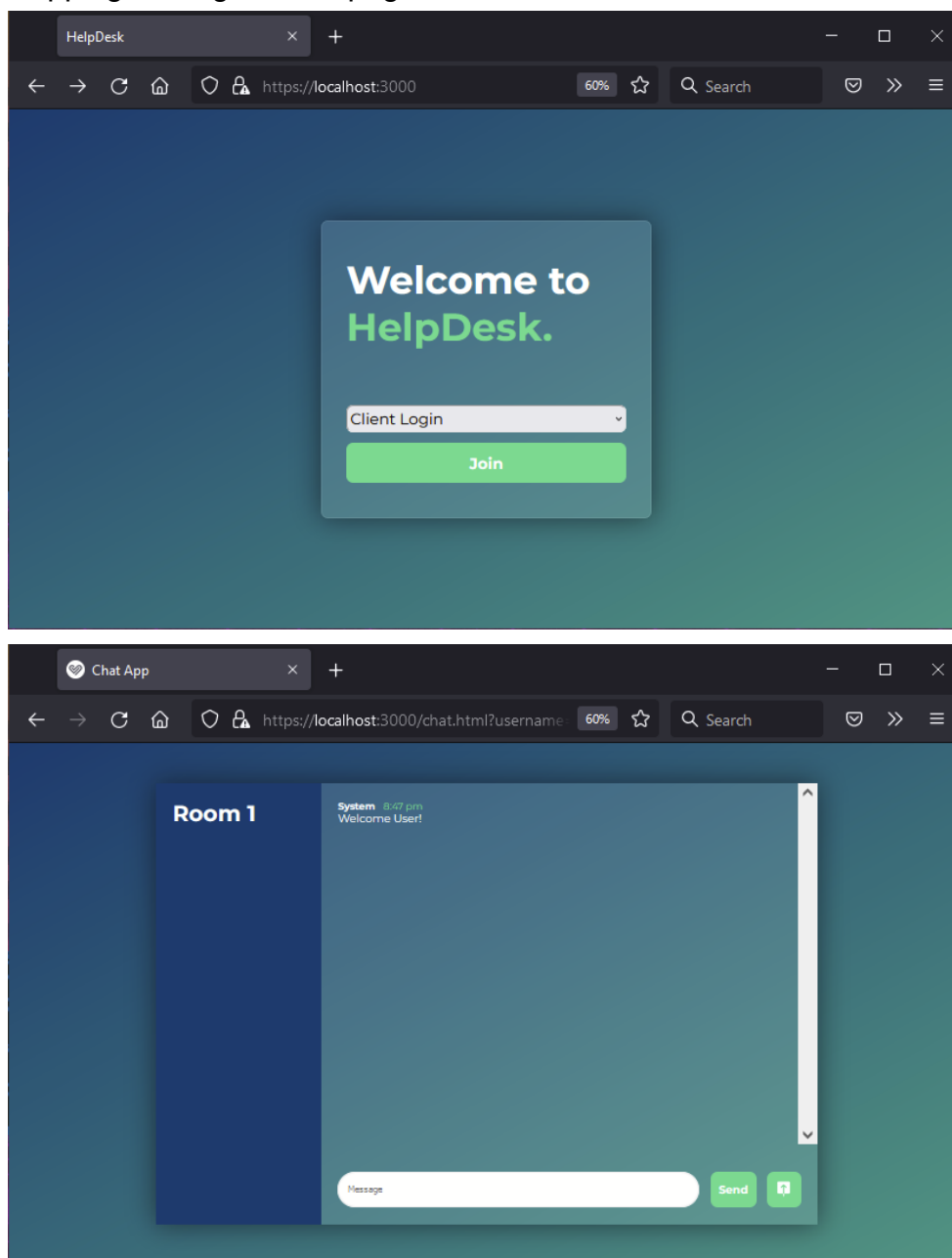
Entered into the chat room:



Session cookie:

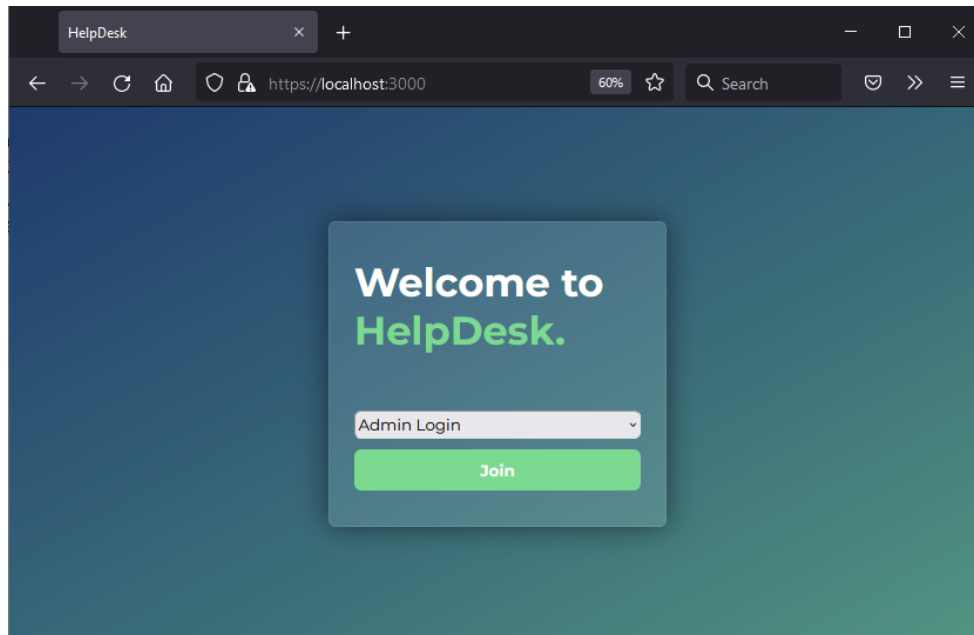


Skipping the registration page:

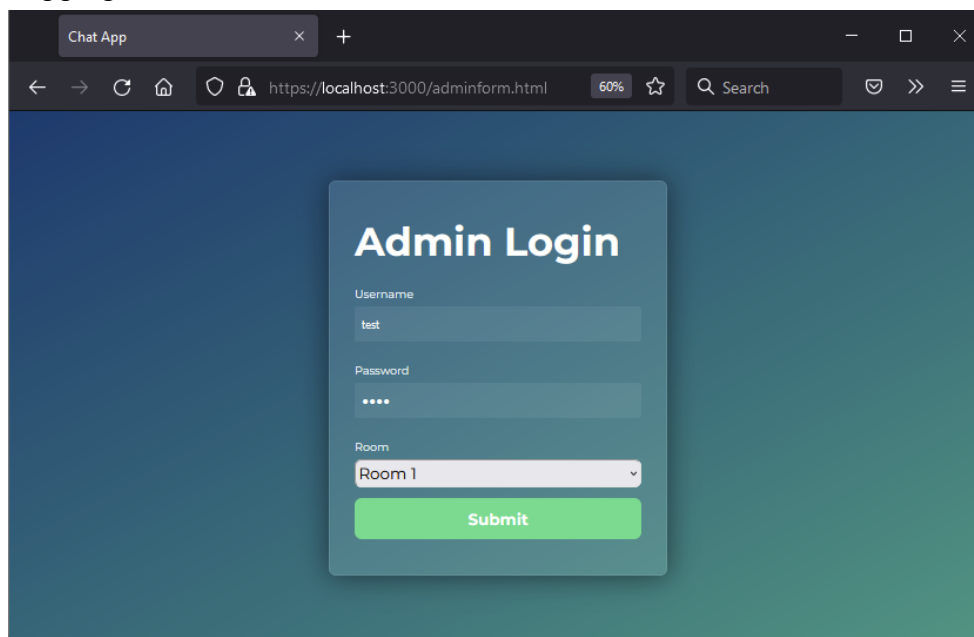


For Admin Users:

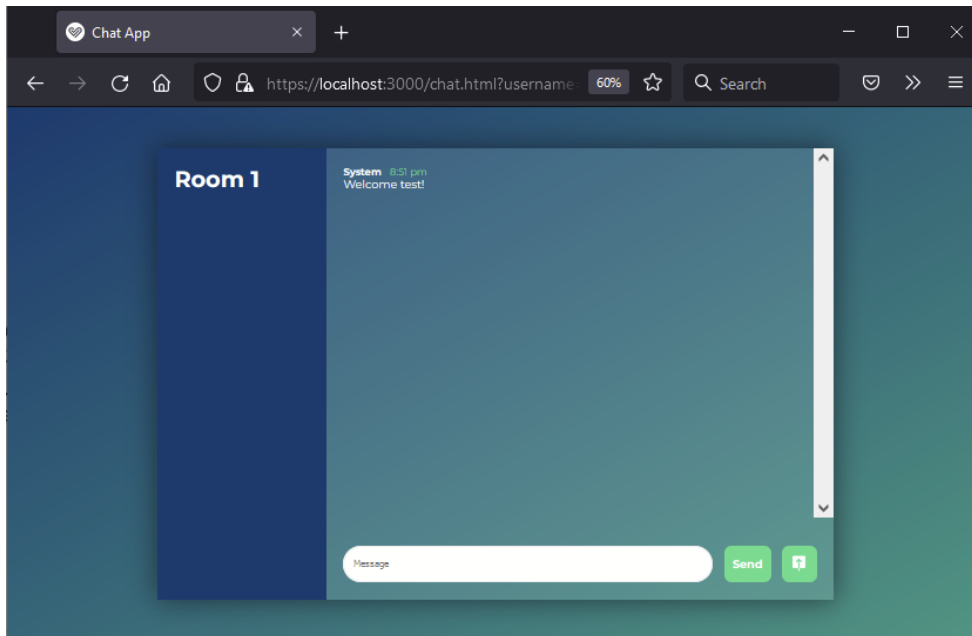
An admin user at the homepage:



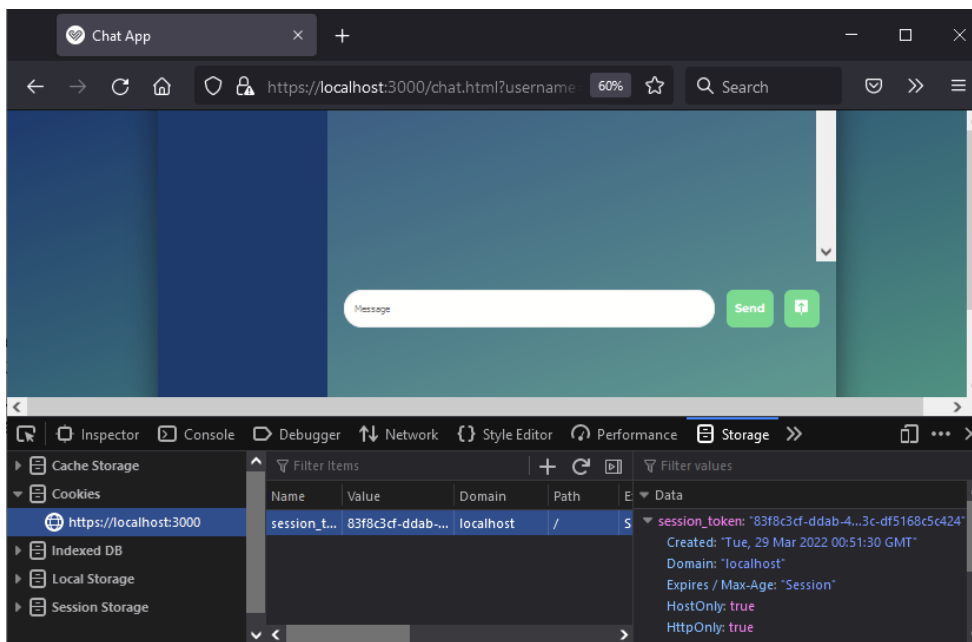
Logging in as an admin user:



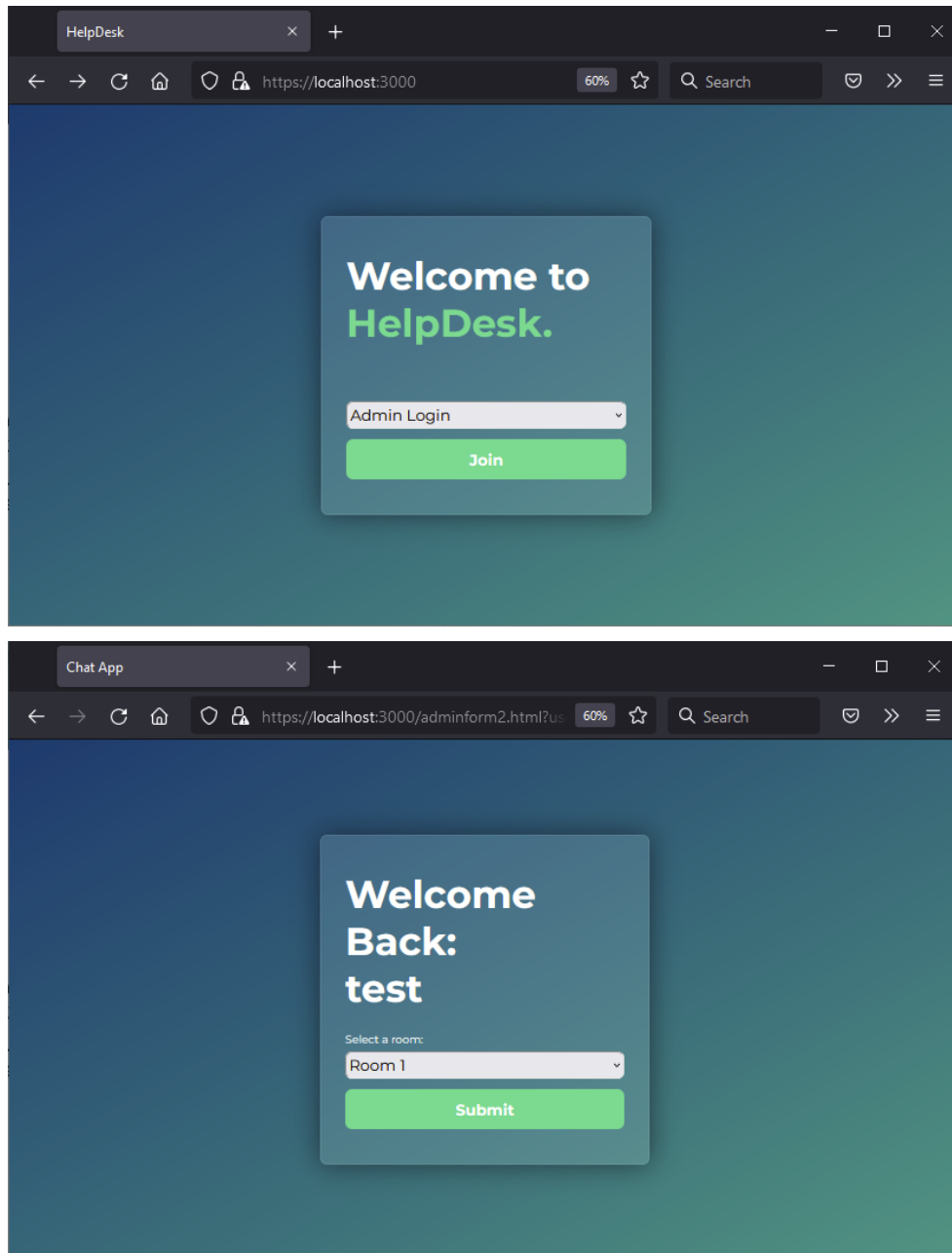
Entered into the chat room:



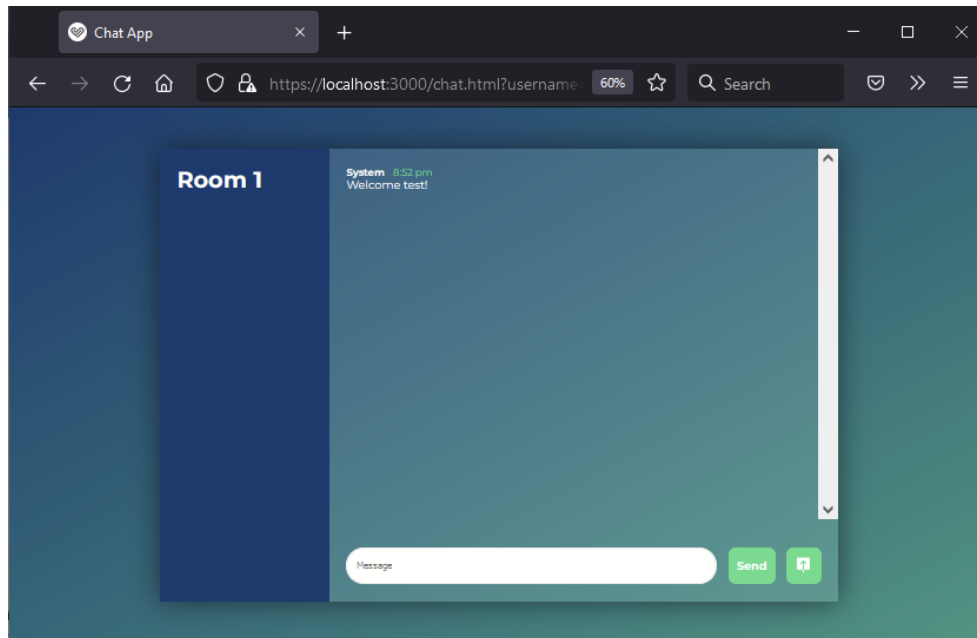
Session cookie:



Skipping the login page, going to a page that only requires room selection:



Entering into the chat room again:



4. Source Code

All of our source code is provided in the GitHub repository. The GitHub repository for the help-desk chat software can be found at the following link:

<https://github.com/Harsh-B-Patel/HelpDeskApp/tree/main>. Please make sure that you utilize the **main branch** in the repository. The Readme file contains installation instructions.