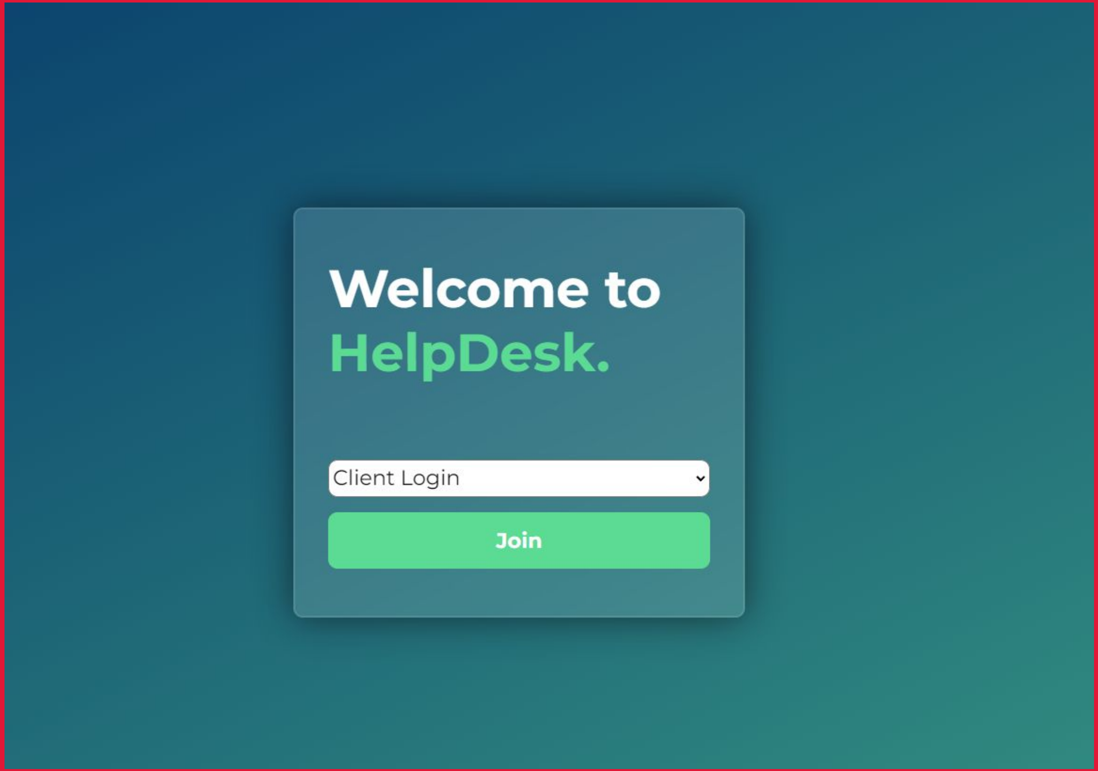


EECS 4481 - HelpDesk App Presentation

Derui Liu, Harsh Patel, Jihal Patel, and Lukas Rose
Team 3

YORK U

High Level Description

System Features

- Authentication
- Joining a Chat Room
- Room Selection

Authentication

Description and Priority

- Authentication is responsible for verifying a help-desk user’s identity
- This feature had a high priority as without it, we would not be able to discern between a help-desk user and an anonymous client

Stimulus/Response Sequences

- The user action required consist of the help-desk user clicking on the login button from the homepage
- Upon entering correct credentials and authenticating, the user should be redirected to the appropriate page

Authentication

Functional Requirements

- REQ-1: User is a help-desk user
- REQ-2: User has correct login credentials
- REQ-3: User clicks on the help-desk agent login button from the home page
- REQ-4: User enters the correct credentials
- REQ-5: User clicks the login button

Joining a Chat Room

Description and Priority

- The join chat room button is a feature that allows anonymous users to access a chat room
- This feature had a high priority as without it, anonymous users would not be able to join chat rooms to talk with help-desk users

Stimulus/Response Sequences

- The user action required consist of the client entering their display name and clicking on the login button from the homepage
- The waiting queue or an empty chat room should appear if there is an available help-desk agent

Joining a Chat Room

Functional Requirements

- REQ-1: User is a anonymous user (client)
- REQ-2: User clicks on the join button
- REQ-3: User waits for an available room/help-desk agent

Room Selection

Description and Priority

- Room selection is a feature for help-desk users to choose the room they want to enter
- This feature had a medium priority as without it, the help-desk user would automatically get assigned an empty room but would not be able to join other help-desk users to chat with them

Stimulus/Response Sequences

- The user action required consist of the help-desk user clicking on the login button from the homepage
- Upon entering correct credentials and authenticating, the user should be redirected to the room selection page
- Upon selection of a room, the help–desk user can click join and enter the chosen room

Room Selection

Functional Requirements

- REQ-1: User is a help-desk user
- REQ-2: User has correct login credentials
- REQ-3: User clicks on the help-desk agent login button from the home page
- REQ-4: User enters the correct credentials
- REQ-5: User clicks the login button
- REQ-6: User selects a room
- REQ-7: User clicks join room

Tech Stack

Javascript

Node.js: We make use of javascript and the Node.js framework for our backend. This is primarily because our group members were familiar with Node.js when it came to web development.

Express: We make use of the express library which gave us the tools needed for URL routing and handling HTTP requests and responses.

Socket: The socket library is used to create and manage TCP websockets to facilitate the sending of messages and commands in real time between users

MongoDB

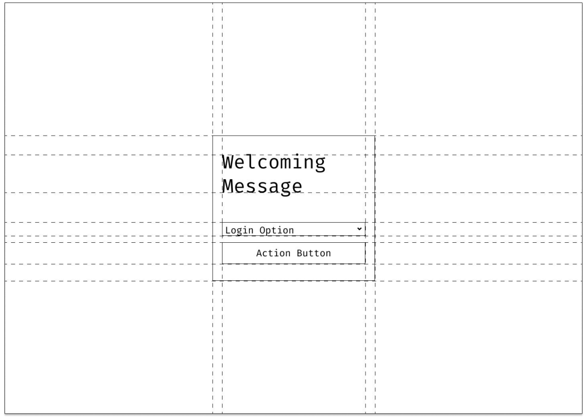
- The help-desk application connects to a cloud-based MongoDB database hosted on MongoDB Atlas

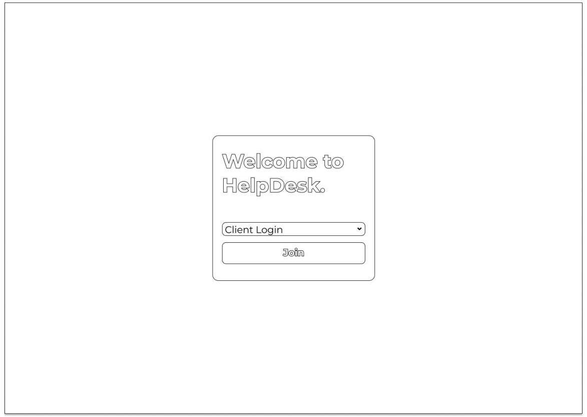
- The database contains login information for help-desk users including:
  - Usernames
  - Hashed passwords
  - ID numbers

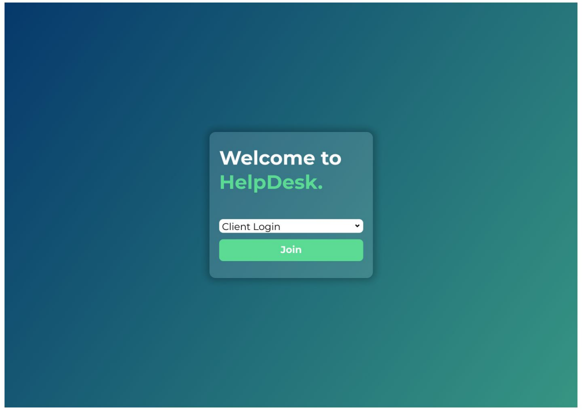
- Upon a login attempt, the system sends a query to the database for entries with matching usernames and password hashes
  - The database returns entries that match

Figma

- Helped with very initial designing.
- Enabled us to make styling changes easily before implementation.
- Having properties of each element defined made it easy to exactly match the design.

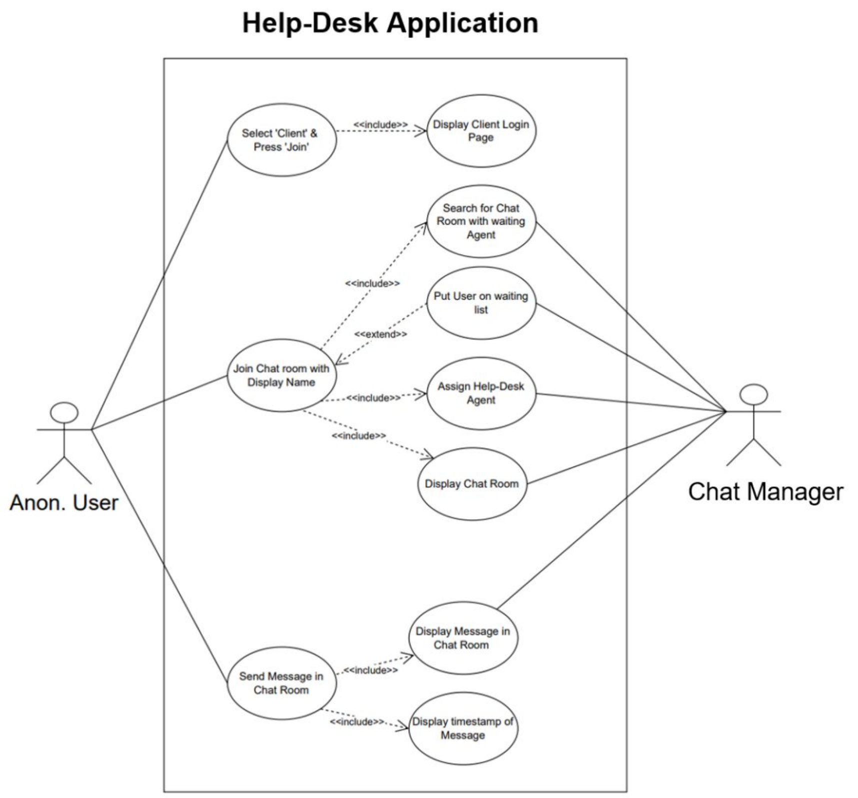




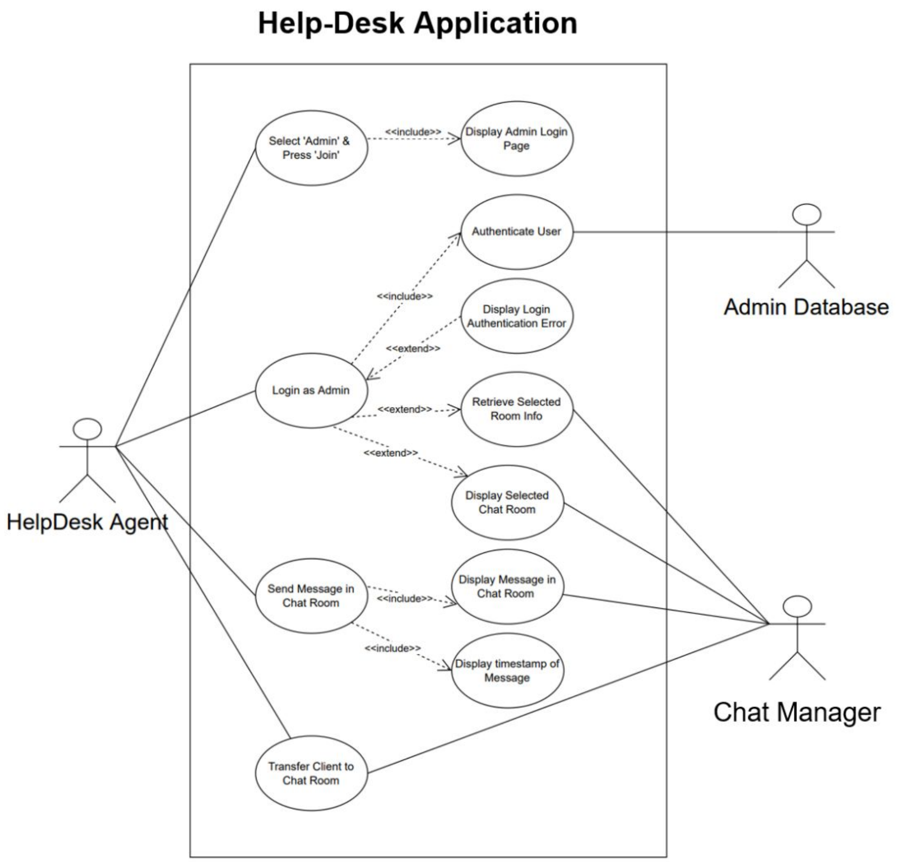


Use Cases

Anonymous User Use Case Diagram


Help-Desk Application
Select 'Client' & Press 'Join'
<<include>>
Display Client Login Page
Search for Chat Room with waiting Agent
<<include>>
Put User on waiting list
<<extend>>
Join Chat room with Display Name
Assign Help-Desk Agent
<<include>>
<<include>>
Display Chat Room
Anon. User
Chat Manager
Display Message in Chat Room
Send Message in Chat Room
<<include>>
<<include>>
Display timestamp of Message

Help-desk User Use Case Diagram


Help-Desk Application
Select 'Admin' & Press 'Join'
<<include>>
Display Admin Login Page
Authenticate User
Admin Database
<<include>>
Display Login Authentication Error
<<extend>>
Login as Admin
<<extend>>
Retrieve Selected Room Info
<<extend>>
Display Selected Chat Room
HelpDesk Agent
Send Message in Chat Room
<<include>>
Display Message in Chat Room
<<include>>
Display timestamp of Message
Chat Manager
Transfer Client to Chat Room

Live Demo

Penetration Testing

Penetration Testing

Password Cracking

Cross-Site Request Forgery (CSRF)

S.Q.L Injection Attack

Cross-Site Scripting Attack (XSS)

Static Code Analysis

- Cleartext submission of password
- Password field with autocomplete enabled
- Unencrypted communications

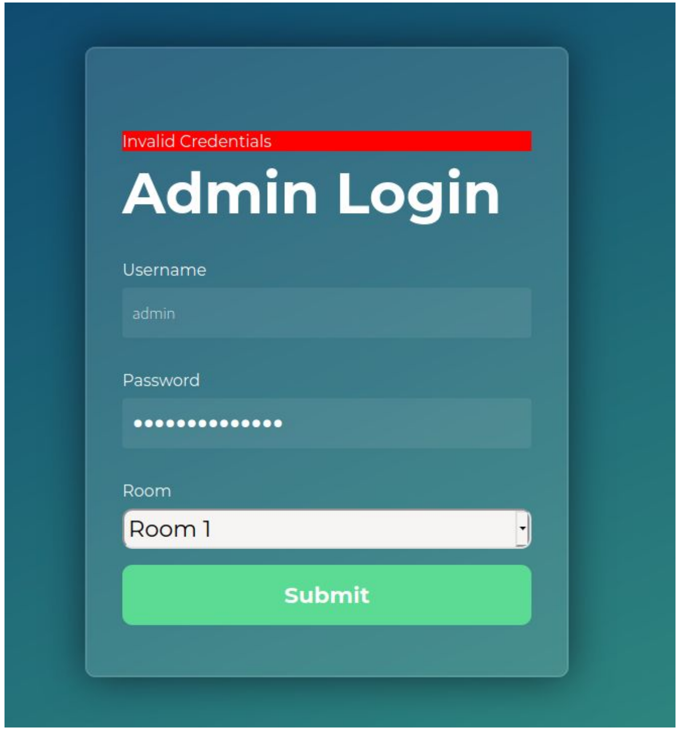
Password Cracking

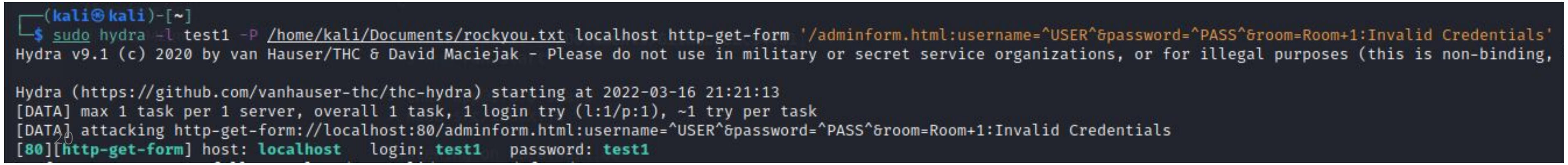
We used **Hydra** to perform dictionary attack on the admin login page of HelpDesk App.

We employed a 5 letter password for user “test1” and password test1.

Dictionary attack using rockyou.txt was easily able to crack it.

Discovered Password Cracking Vulnerability.





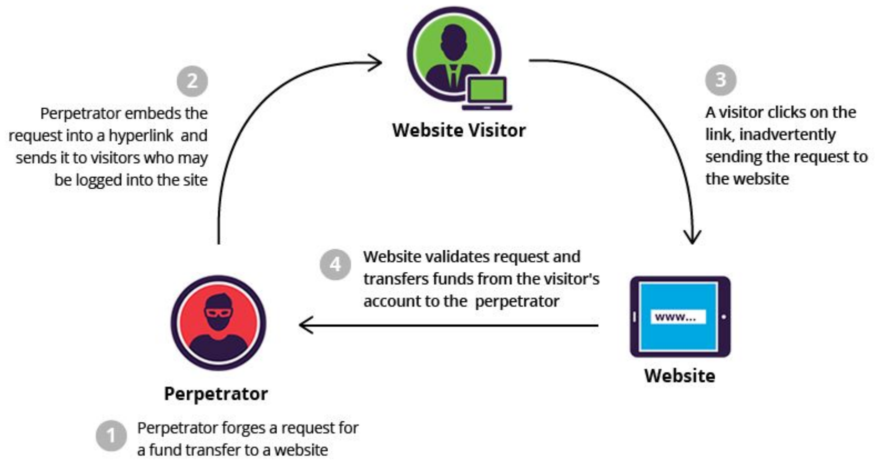
```
┌──(kali㉿kali)-[~]
└─$ sudo hydra -l test1 -P /home/kali/Documents/rockyou.txt localhost http-get-form '/adminform.html:username=^USER^&password=^PASS^&room=Room+1:Invalid Credentials'
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding,

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-03-16 21:21:13
[DATA] max 1 task per 1 server, overall 1 task, 1 login try (l:1/p:1), ~1 try per task
[DATA] attacking http-get-form://localhost:80/adminform.html:username=^USER^&password=^PASS^&room=Room+1:Invalid Credentials
[80][http-get-form] host: localhost   login: test1   password: test1
```

Cross-Site Request Forgery (CSRF)

No CSRF Vulnerability was found due to lack of cookies being employed in our application before part 3.

After Part 3, we implemented cookies with session control functionality (session token issued by server). We ran **burp suite** to test for CSRF vulnerability and it came back as negative.

Hence, we concluded that our application does not have CSRF vulnerability.


2
Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
Website Visitor
3
A visitor clicks on the link, inadvertently sending the request to the website
4
Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Website
Perpetrator
1
Perpetrator forges a request for a fund transfer to a website

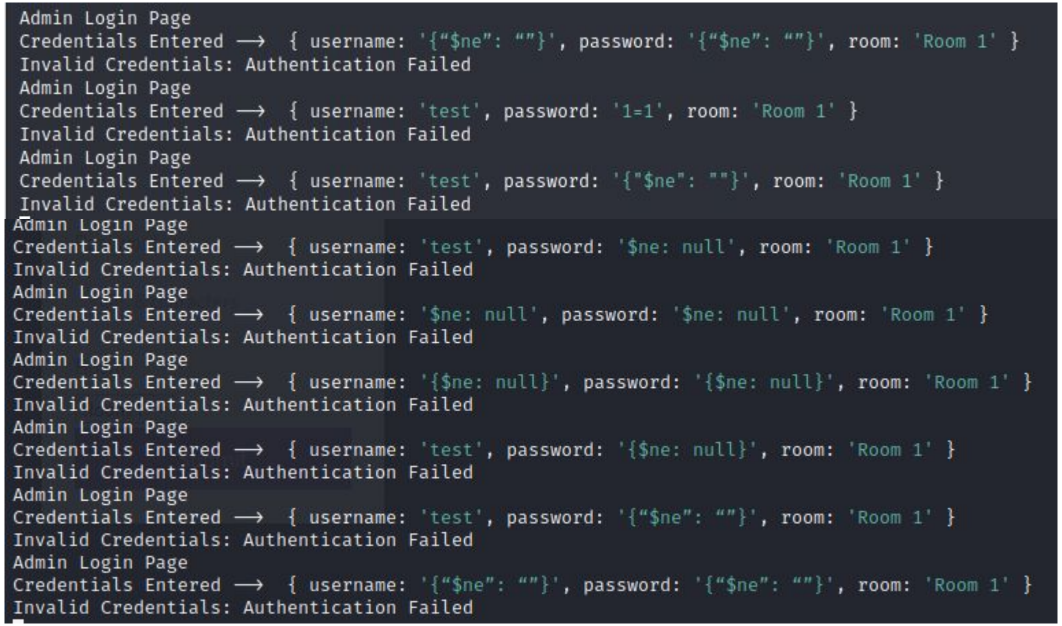
S.Q.L Injection Attack

In order to make our application more secure we decided to use NoSQL with MongoDB instead of traditional SQL. This was done to provide a layer of protection against SQL injections.

So we performed NoSQL Injections.
Ex: {$ne: null} {$ne: 1} {$gt: ’’} {$gt: ''} [$ne]=1 { $ne: 1 } {“$ne”: “”} where ‘1=1’ etc

Our tests revealed that MongoDb, would automatically convert all queries to Strings. rendering Tautologies & Piggybacked injections useless.

MongoDb also does not return an error for a conversion error, so we can not apply Illegal/Logically Incorrect Queries.



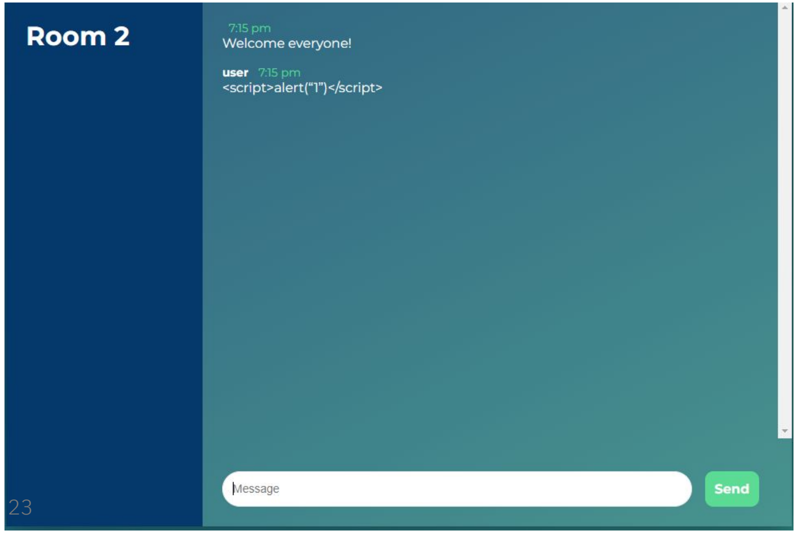
```
Admin Login Page
Credentials Entered ⟶ { username: '{“$ne”: “”}', password: '{“$ne”: “”}', room: 'Room 1' }
Invalid Credentials: Authentication Failed
Admin Login Page
Credentials Entered ⟶ { username: 'test', password: '1=1', room: 'Room 1' }
Invalid Credentials: Authentication Failed
Admin Login Page
Credentials Entered ⟶ { username: 'test', password: '{"$ne": ""}', room: 'Room 1' }
Invalid Credentials: Authentication Failed
Admin Login Page
Credentials Entered ⟶ { username: 'test', password: '$ne: null', room: 'Room 1' }
Invalid Credentials: Authentication Failed
Admin Login Page
Credentials Entered ⟶ { username: '$ne: null', password: '$ne: null', room: 'Room 1' }
Invalid Credentials: Authentication Failed
Admin Login Page
Credentials Entered ⟶ { username: '{$ne: null}', password: '{$ne: null}', room: 'Room 1' }
Invalid Credentials: Authentication Failed
Admin Login Page
Credentials Entered ⟶ { username: 'test', password: '{$ne: null}', room: 'Room 1' }
Invalid Credentials: Authentication Failed
Admin Login Page
Credentials Entered ⟶ { username: 'test', password: '{“$ne”: “”}', room: 'Room 1' }
Invalid Credentials: Authentication Failed
Admin Login Page
Credentials Entered ⟶ { username: '{“$ne”: “”}', password: '{“$ne”: “”}', room: 'Room 1' }
Invalid Credentials: Authentication Failed
```

Cross-Site Scripting Attack (XSS)

Persistent XSS

<script>alert(1);</script>

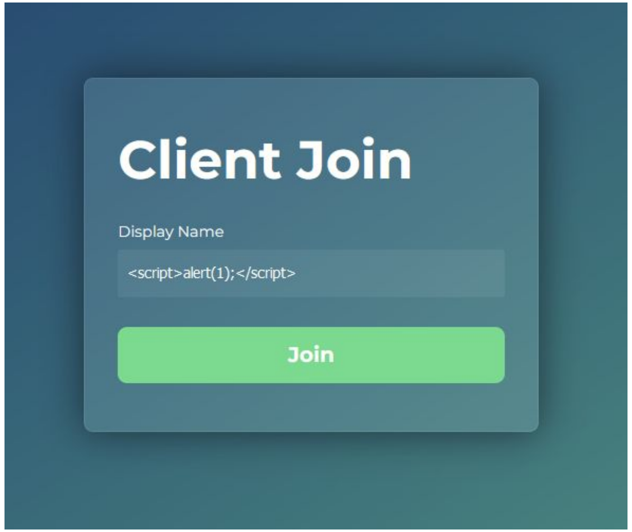
In Chat Window


Room 2

7:15 pm
Welcome everyone!

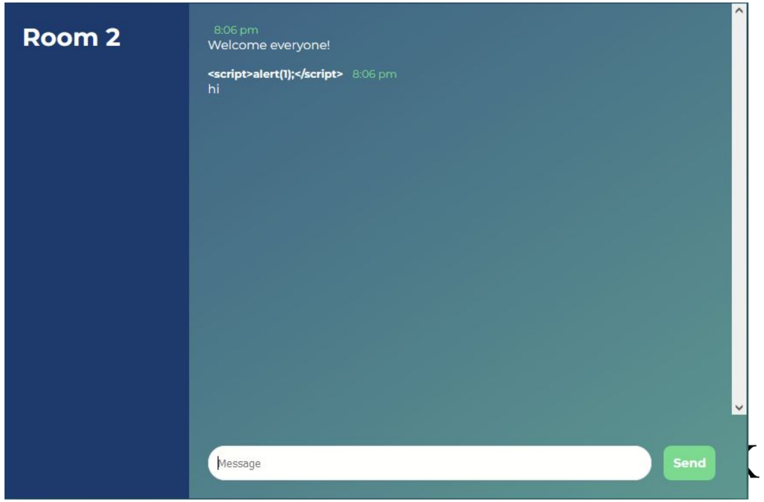
user 7:15 pm
<script>alert("1")</script>

In Client Name


Client Join

Display Name

<script>alert(1);</script>

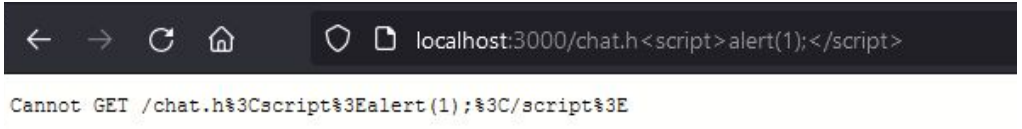

Room 2

8:06 pm
Welcome everyone!

<script>alert(1);</script> 8:06 pm
hi

Reflected XSS

This does not work. The node.js server correctly encodes the special characters in the string, such that the information reflected in the error message is no longer a script block, and does not execute.

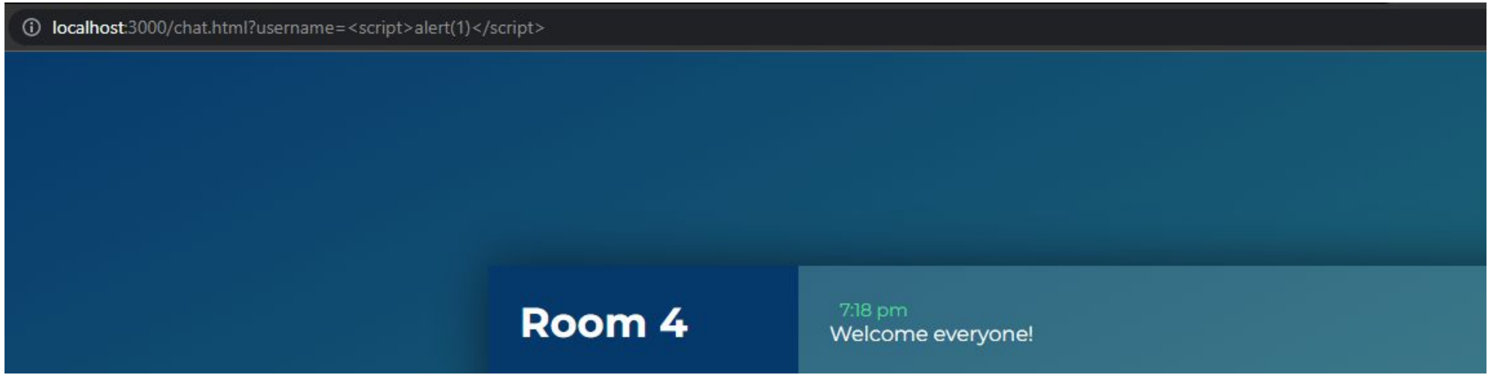


```
localhost:3000/chat.h<script>alert(1);</script>

Cannot GET /chat.h%3Cscript%3Ealert(1);%3C/script%3E
```

DOM-based XSS

We performed a DOM-based XSS by using this modified URL: http://localhost:3000/chat.html?username=<script>alert(1);</script>

This does not work. The document object model is unaffected by this url, as the Node.js server interprets this as an incoming connection by a user with the name: "<script>alert(1);</script>" joining the chat. The application safely renders the script into text, and protects the user from an XSS attack.



```
localhost:3000/chat.html?username=<script>alert(1)</script>
```

Room 4

7:18 pm
Welcome everyone!

Static Code Analysis (Burp Suite Pro)

We Performed Static Code Analysis using Burp Suite Pro.

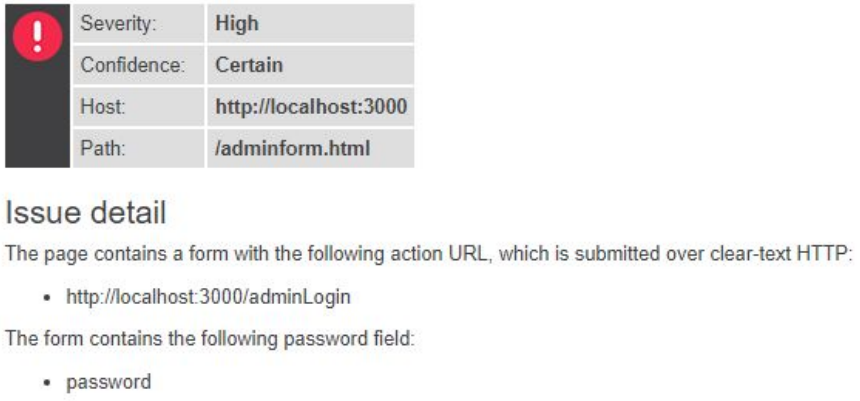
Cleartext submission of password

The Password is being sent to the database for authentication in clear/plain text.

This vulnerability should be handled by first encrypting/hashing the password on the client's device before being sent to the database for authentication.

This however is not enough, hackers usually have a stored database of common passwords and their respective hashes. So a strong password policy needs to be implemented.

The application should also use HTTPS to further protect traffic between the client and server.



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

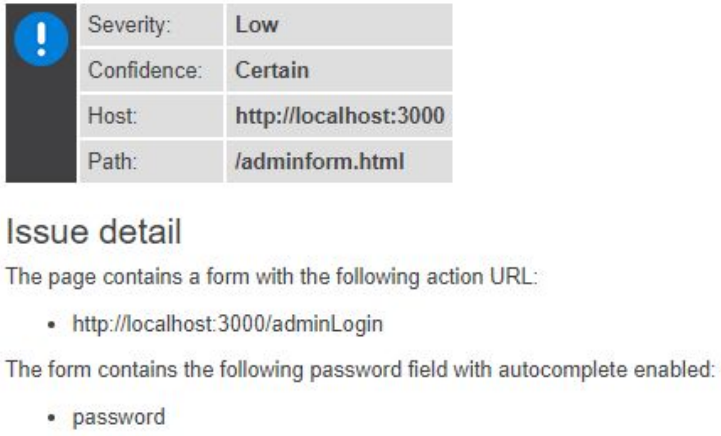
The form contains the following password field:

- password

Password field with autocomplete enabled

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.

These stored credentials can be captured by an attacker.



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

Unencrypted communications

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

The application should also use HTTPS to further protect traffic between the client and server.

Patching Vulnerabilities

Weak Passwords

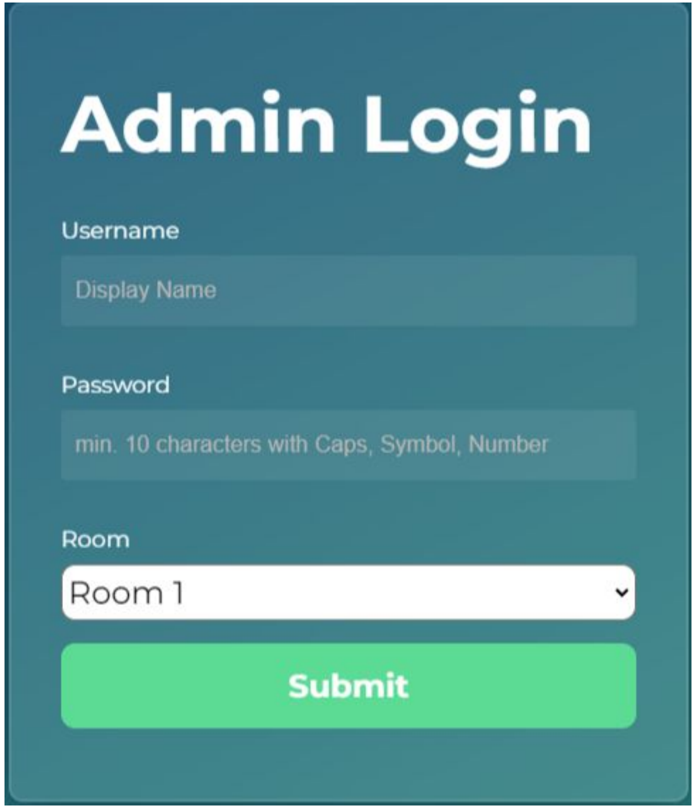
Static Code Analysis

- Cleartext submission of password
- Password field with autocomplete enabled
- Unencrypted communications

Weak Passwords

Previously we used Hydra to perform dictionary attacks on our web login portal and discovered that weak passwords made our portal susceptible to dictionary attacks.

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

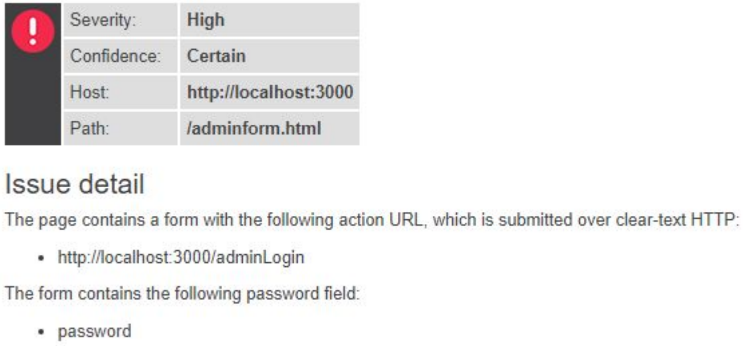


Static Code Analysis

Cleartext submission of password

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 cannot be eavesdropped.



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

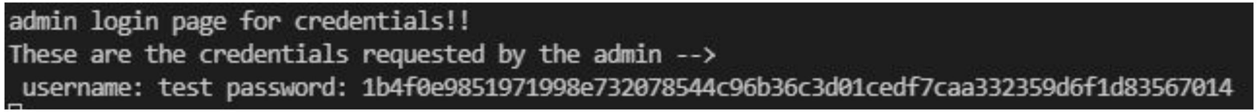
Issue detail

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:

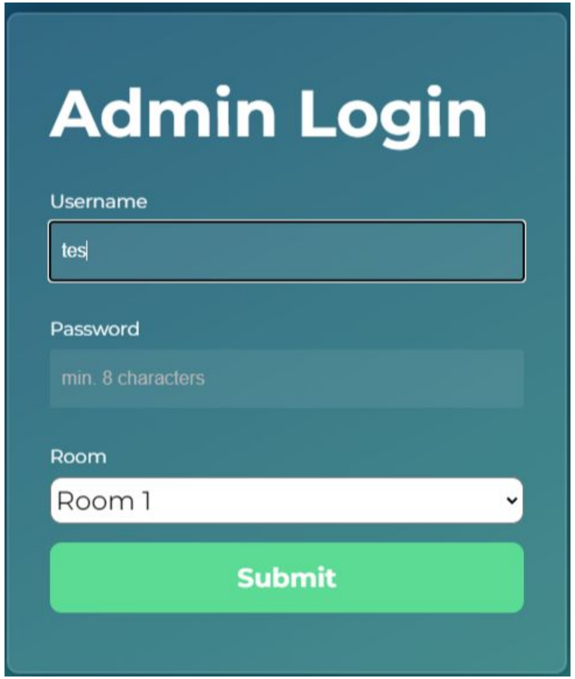
- password

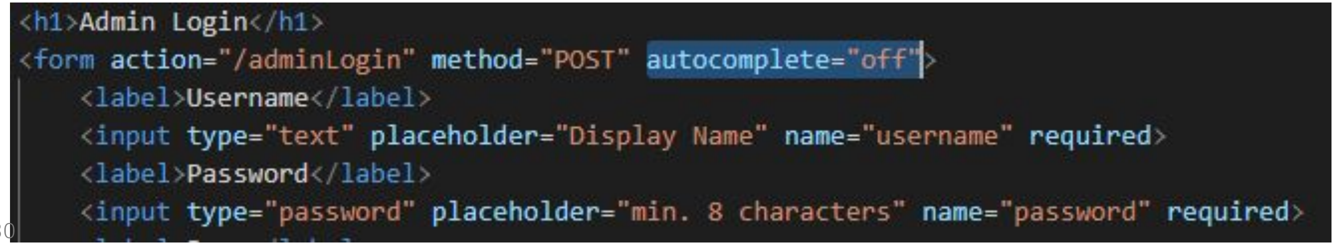


```
admin login page for credentials!!
These are the credentials requested by the admin -->
 username: test password: 1b4f0e9851971998e732078544c96b36c3d01cedf7caa332359d6f1d83567014
```

Password field with autocomplete enabled

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





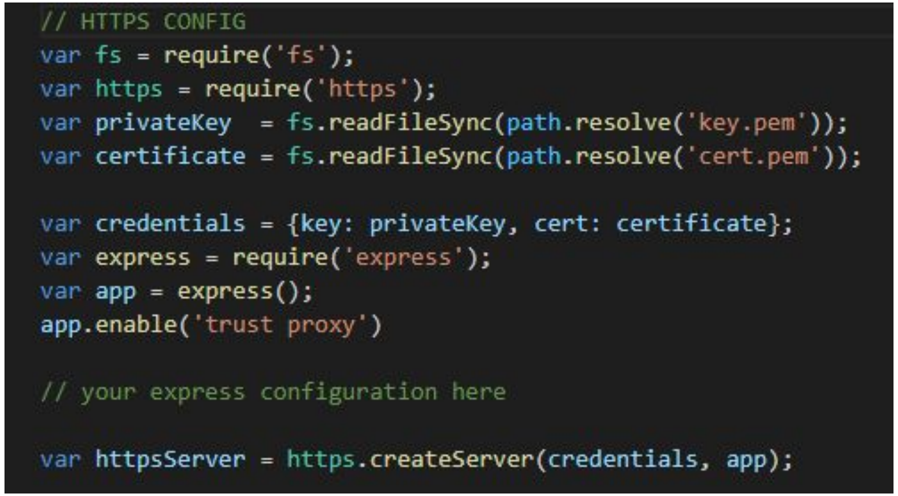
```
<h1>Admin Login</h1>
<form action="/adminLogin" method="POST" autocomplete="off">
    <label>Username</label>
    <input type="text" placeholder="Display Name" name="username" required>
    <label>Password</label>
    <input type="password" placeholder="min. 8 characters" name="password" required>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

Unencrypted communications

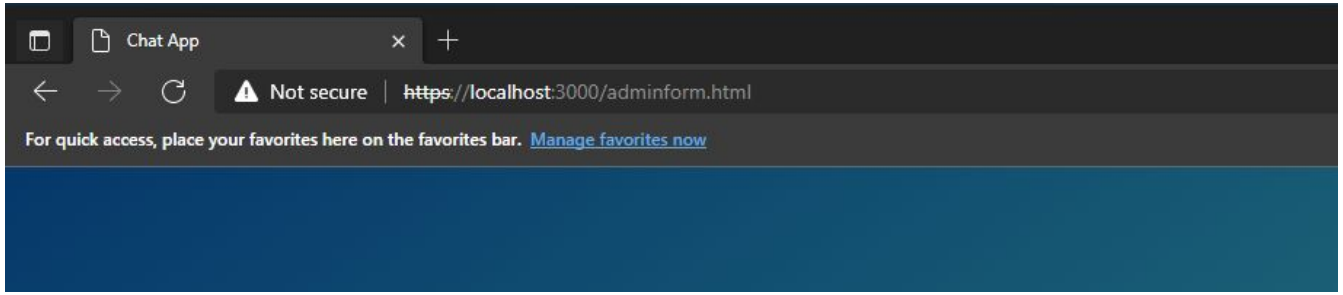
We removed HTTP and instead employed HTTPS with a private key and certificate. Also created HTTP redirect to HTTPS.

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.



```
// 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);
```



Chat App
Not secure
~~https~~://localhost:3000/adminform.html
For quick access, place your favorites here on the favorites bar. Manage favorites now

Optimizations

Optimizations

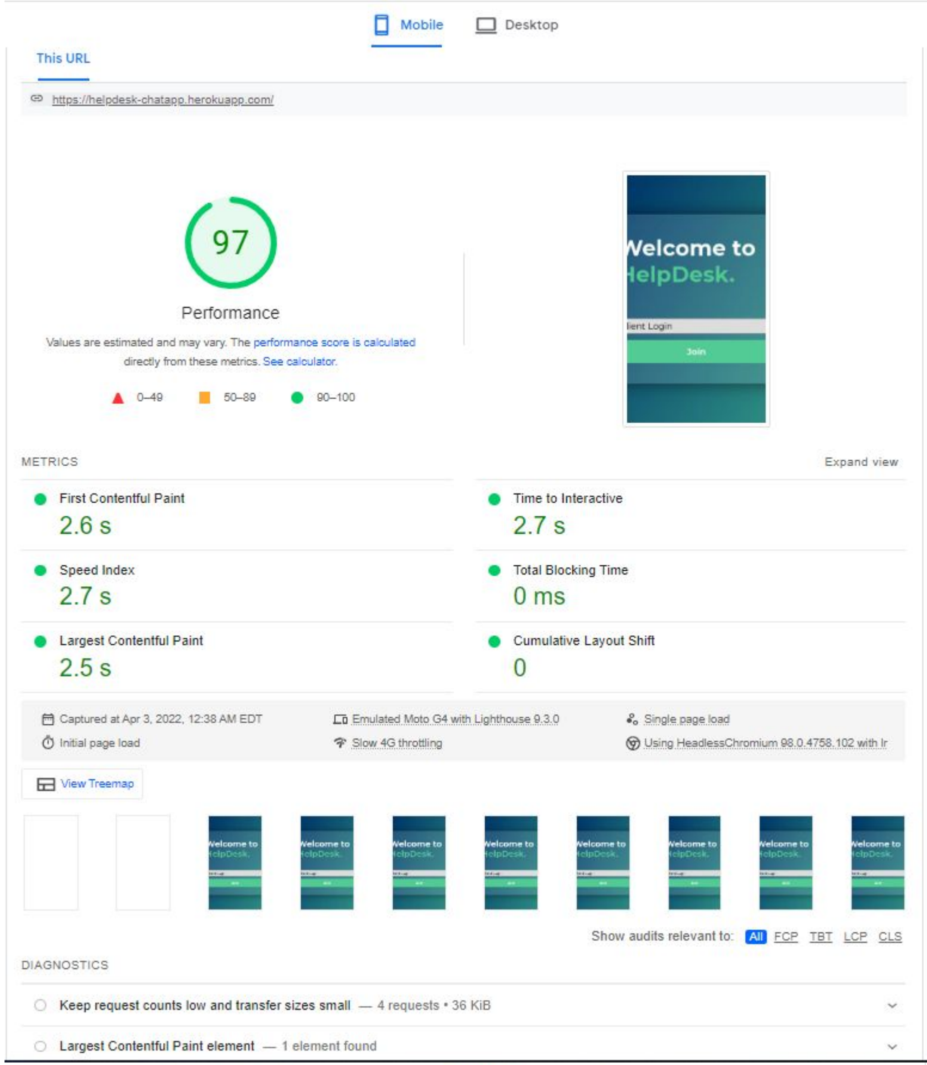
Google PageSpeed

Desktop: 99%

Mobile: 83% ⇨ 97%

Google

PageSpeed Insights


Mobile Desktop

This URL

https://helpdesk-chatapp.herokuapp.com/

97

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

● First Contentful Paint 2.6 s	● Time to Interactive 2.7 s
● Speed Index 2.7 s	● Total Blocking Time 0 ms
● Largest Contentful Paint 2.5 s	● Cumulative Layout Shift 0

Captured at Apr 3, 2022, 12:38 AM EDT
Initial page load
Emulated Moto G4 with Lighthouse 9.3.0
Slow 4G throttling
Single page load
Using HeadlessChromium 98.0.4758.102 with lr

View Treemap

Show audits relevant to: All FCP TBT LCP CLS

DIAGNOSTICS

Keep request counts low and transfer sizes small — 4 requests • 36 KiB

Largest Contentful Paint element — 1 element found

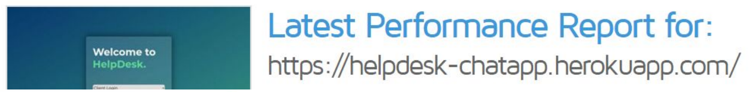
Optimizations

GTmetrix

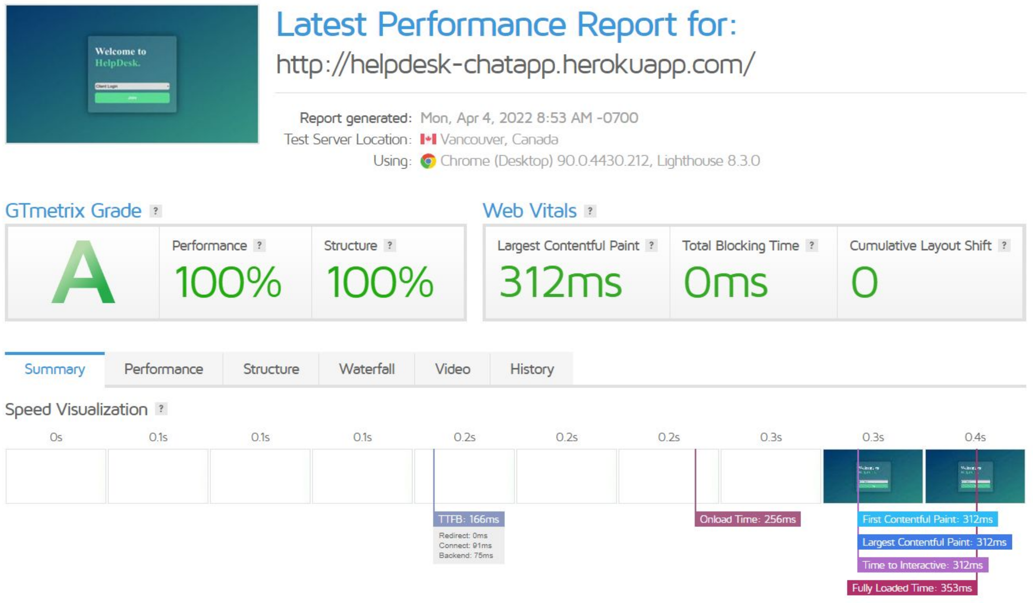
GTmetrix Grade: **A**

Performance Score: **100%**

Structure Score: **95%** → **100%**



Latest Performance Report for:
https://helpdesk-chatapp.herokuapp.com/

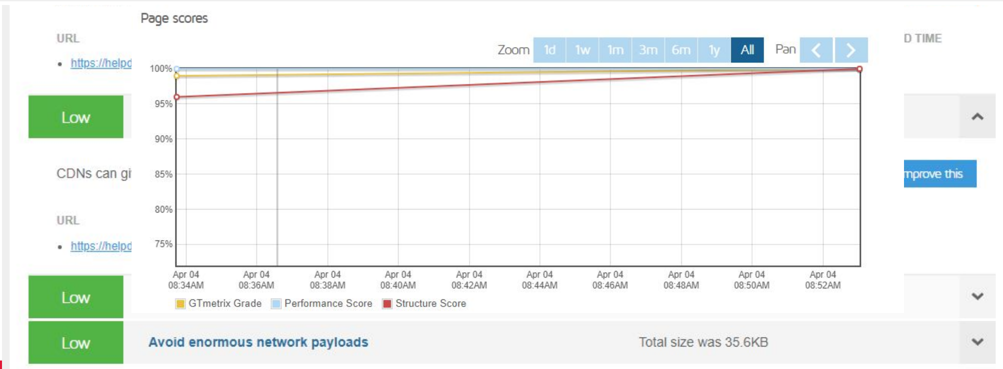


Latest Performance Report for:
http://helpdesk-chatapp.herokuapp.com/

Report generated: Mon, Apr 4, 2022 8:53 AM -0700
Test Server Location: Vancouver, Canada
Using: Chrome (Desktop) 90.0.4430.212, Lighthouse 8.3.0

GTmetrix Grade: A — Performance 100% — Structure 100%

Web Vitals: Largest Contentful Paint 312ms — Total Blocking Time 0ms — Cumulative Layout Shift 0



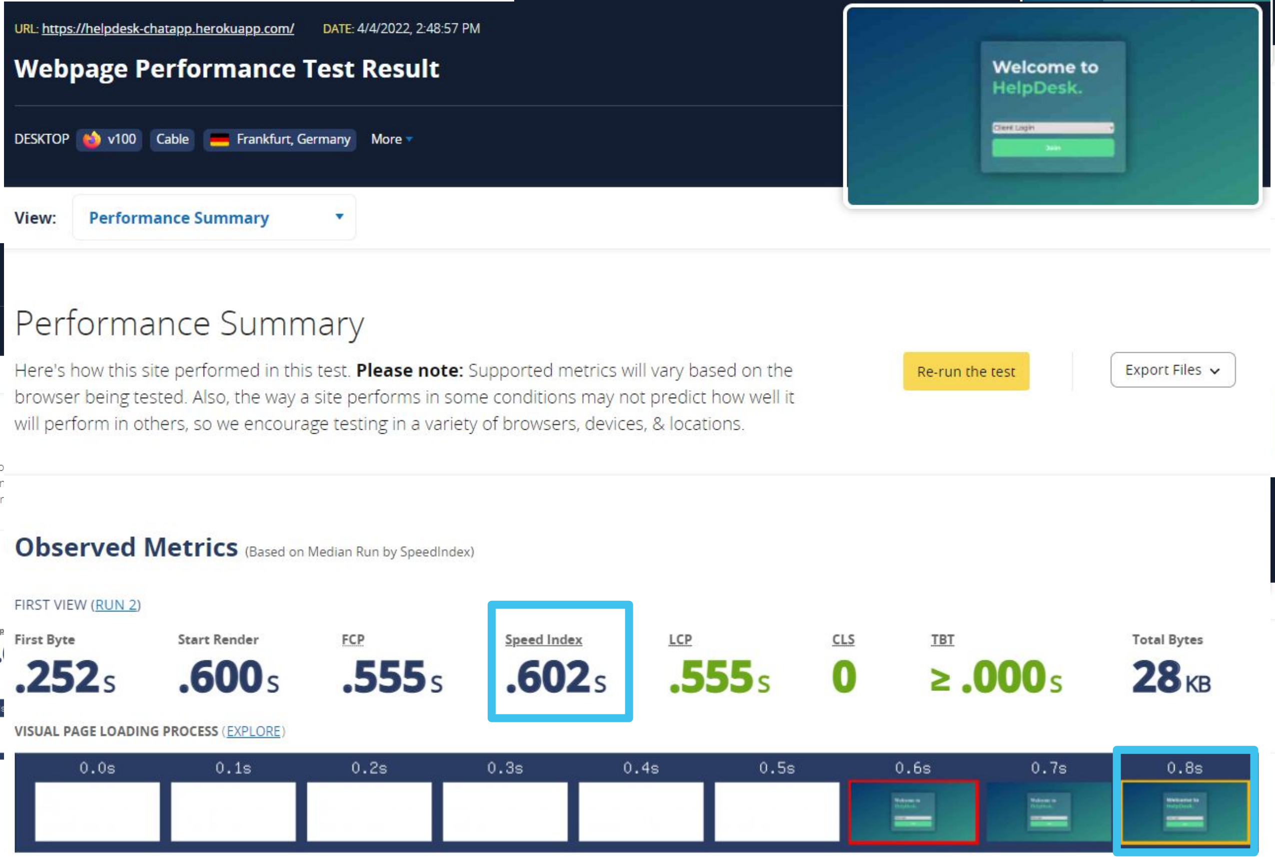
Low — Avoid enormous network payloads — Total size was 35.6KB

Optimizations

Webpage Test

URL: https://helpdesk-chatapp.herokuapp.com/ DATE: 4/4/2022, 2:46:57 PM

Webpage Performance Test Result



URL: https://helpdesk-chatapp.herokuapp.com/ DATE: 4/4/2022, 2:48:57 PM

Webpage Performance Test Result

DESKTOP v100 Cable Frankfurt, Germany More

View: Performance Summary

Performance Summary

Here's how this site performed in this test. **Please note:** Supported metrics will vary based on the browser being tested. Also, the way a site performs in some conditions may not predict how well it will perform in others, so we encourage testing in a variety of browsers, devices, & locations.

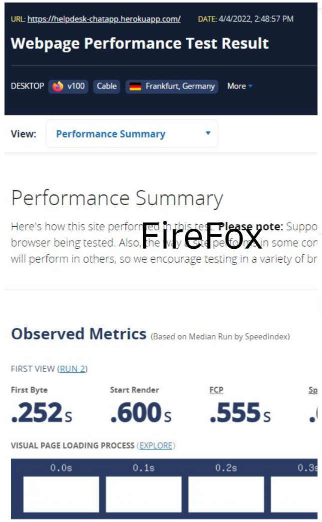
Observed Metrics (Based on Median Run by SpeedIndex)

FIRST VIEW (RUN 2)

First Byte	Start Render	FCP	Speed Index	LCP	CLS	TBT	Total Bytes
.252s	.600s	.555s	.602s	.555s	0	≥ .000s	28KB

VISUAL PAGE LOADING PROCESS (EXPLORE)

FireFox



URL: https://helpdesk-chatapp.herokuapp.com/ DATE: 4/4/2022, 2:48:57 PM

Webpage Performance Test Result

DESKTOP v100 Cable Frankfurt, Germany More

View: Performance Summary

Performance Summary

Here's how this site performed in this test. **Please note:** Suppo
browser being tested. Also, the way a site performs in some con
will perform in others, so we encourage testing in a variety of br

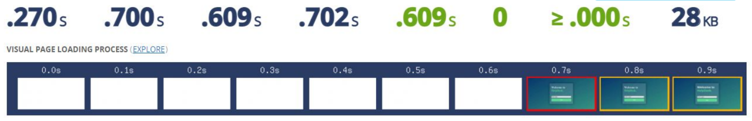
Observed Metrics (Based on Median Run by SpeedIndex)

FIRST VIEW (RUN 2)

First Byte	Start Render	FCP	Sp
.252s	.600s	.555s	.

VISUAL PAGE LOADING PROCESS (EXPLORE)

FireFox



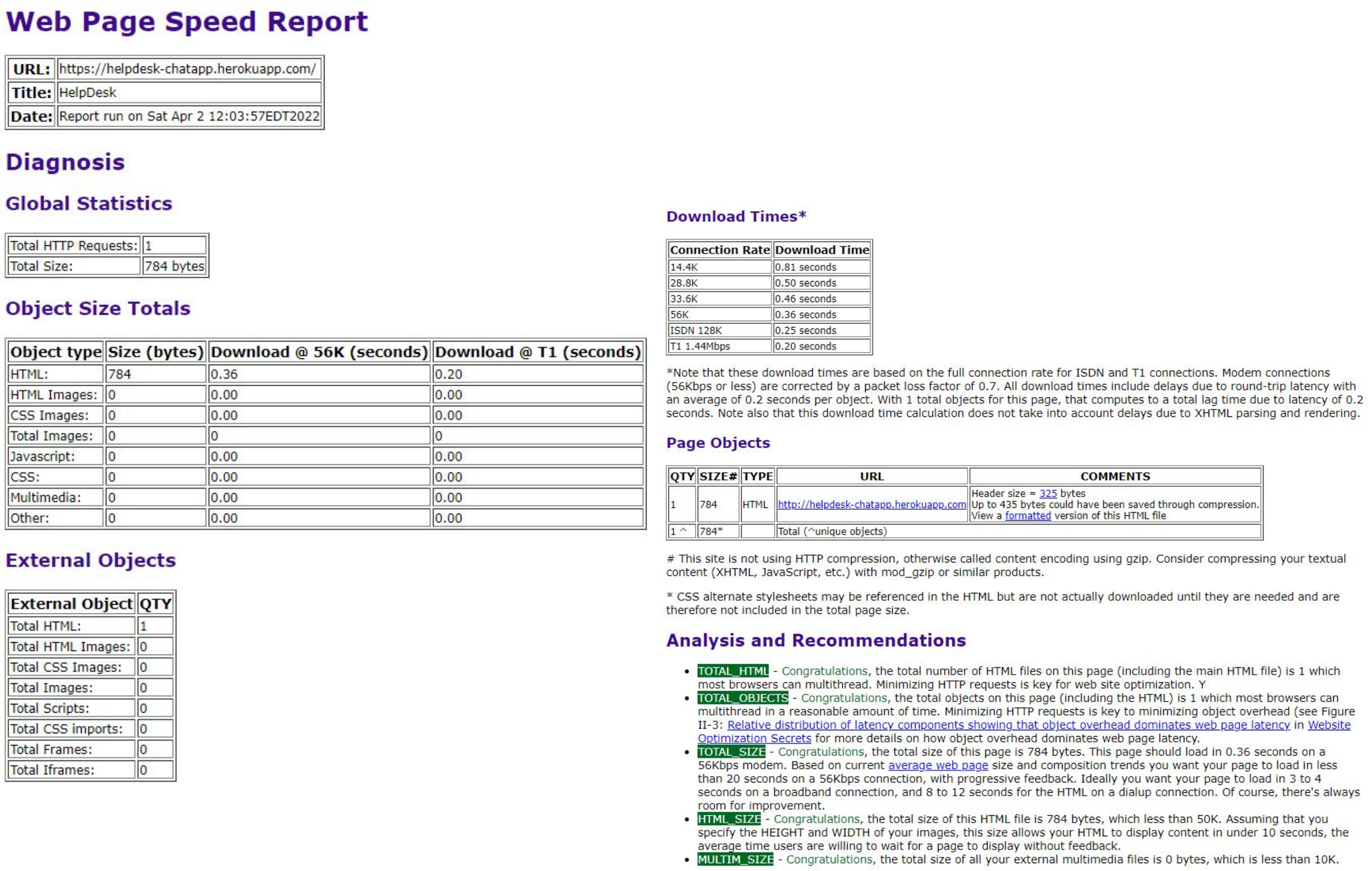
.270s	.700s	.609s	.702s	.609s	0	≥ .000s	28KB

VISUAL PAGE LOADING PROCESS (EXPLORE)

WEBPAGETEST by Catchpoint®

Optimizations

Optimization.com



Web Page Speed Report

URL:	https://helpdesk-chatapp.herokuapp.com/
Title:	HelpDesk
Date:	Report run on Sat Apr 2 12:03:57EDT2022

Diagnosis

Global Statistics

Total HTTP Requests:	1
Total Size:	784 bytes

Object Size Totals

Object type	Size (bytes)	Download @ 56K (seconds)	Download @ T1 (seconds)
HTML:	784	0.36	0.20
HTML Images:	0	0.00	0.00
CSS Images:	0	0.00	0.00
Total Images:	0	0	0
Javascript:	0	0.00	0.00
CSS:	0	0.00	0.00
Multimedia:	0	0.00	0.00
Other:	0	0.00	0.00

External Objects

External Object	QTY
Total HTML:	1
Total HTML Images:	0
Total CSS Images:	0
Total Images:	0
Total Scripts:	0
Total CSS imports:	0
Total Frames:	0
Total Iframes:	0

Download Times*

Connection Rate	Download Time
14.4K	0.81 seconds
28.8K	0.50 seconds
33.6K	0.46 seconds
56K	0.36 seconds
ISDN 128K	0.25 seconds
T1 1.44Mbps	0.20 seconds

*Note that these download times are based on the full connection rate for ISDN and T1 connections. Modem connections (56Kbps or less) are corrected by a packet loss factor of 0.7. All download times include delays due to round-trip latency with an average of 0.2 seconds per object. With 1 total objects for this page, that computes to a total lag time due to latency of 0.2 seconds. Note also that this download time calculation does not take into account delays due to XHTML parsing and rendering.

Page Objects

QTY	SIZE#	TYPE	URL	COMMENTS
1	784	HTML	http://helpdesk-chatapp.herokuapp.com	Header size = 325 bytes Up to 435 bytes could have been saved through compression. View a formatted version of this HTML file
1 ^	784*		Total (^unique objects)	

This site is not using HTTP compression, otherwise called content encoding using gzip. Consider compressing your textual content (XHTML, JavaScript, etc.) with mod_gzip or similar products.

* CSS alternate stylesheets may be referenced in the HTML but are not actually downloaded until they are needed and are therefore not included in the total page size.

Analysis and Recommendations

- TOTAL_HTML - Congratulations, the total number of HTML files on this page (including the main HTML file) is 1 which most browsers can multithread. Minimizing HTTP requests is key for web site optimization. Y
- TOTAL_OBJECTS - Congratulations, the total objects on this page (including the HTML) is 1 which most browsers can multithread in a reasonable amount of time. Minimizing HTTP requests is key to minimizing object overhead (see Figure II-3: Relative distribution of latency components showing that object overhead dominates web page latency in Website Optimization Secrets for more details on how object overhead dominates web page latency.
- TOTAL_SIZE - Congratulations, the total size of this page is 784 bytes. This page should load in 0.36 seconds on a 56Kbps modem. Based on current average web page size and composition trends you want your page to load in less than 20 seconds on a 56Kbps connection, with progressive feedback. Ideally you want your page to load in 3 to 4 seconds on a broadband connection, and 8 to 12 seconds for the HTML on a dialup connection. Of course, there's always room for improvement.
- HTML_SIZE - Congratulations, the total size of this HTML file is 784 bytes, which less than 50K. Assuming that you specify the HEIGHT and WIDTH of your images, this size allows your HTML to display content in under 10 seconds, the average time users are willing to wait for a page to display without feedback.
- MULTIM_SIZE - Congratulations, the total size of all your external multimedia files is 0 bytes, which is less than 10K.

Future Changes:

- Admin DashBoard with all online users shown
- Professional web hosting of our application
- Create an mobile app using Android JS

Thank You