

GAME OF GO

Secure Software Design and Programming

ISA 681

Programming Project Report

By

Rahul Gunasekaran(G01148123)

Varun Ravindra Babu(G01150085)

(Under Professor Dr. David A. Wheeler)

December 13, 2019



Table of Contents

1. INTRODUCTION.....	3
2. RULES OF THE GAME:	3
3. ARCHITECTURE OF THE PROJECT.....	4
3.1.1 Django version 2.2.8	4
3.1.2 Django Channels	5
3.1.3 Django Rest Framework.....	5
3.1.4 React JS	5
3.1.5 PostgreSQL 11.5	5
3.1.6 Django Webpack Loader	5
3.1.7 Node Package Manager	5
3.1.8 Nginx Reverse Proxy	6
3.2.1 Model.....	6
3.2.2 View	6
3.2.3 Template	7
3.2.4 Websocket	7
4. INSTALLATION AND OPERATING INSTRUCTIONS	8
5. ASSURANCE CASE.....	10
6. VARIOUS REQUIREMENTS OF THE PROJECT:.....	32
7. REFERENCES:.....	34

1. INTRODUCTION

Go is a strategic board game which can be played by two players. The primary goal of the game is to capture more area or territory than your opponent. The game was invented by the Chinese. In the board game, the pieces with which the game is played are “stones”. Players can either choose a white stone or a black stone. Players play the game by taking turns and placing the stones at empty points. Once a stone is placed, it cannot be moved. A capture occurs when one stone is being surrounded by stones of the opponent on all 4 adjacent sides. The game ends when both players do not wish to make more moves. The winner of the game is the player with more territory and calculating the captured stones. This is an interesting game which can be played in 19x19 or 9x9 or a 13x13 board. The objective of this project is to create the game with various security features and requirements learnt during the course.

2. RULES OF THE GAME:

The game can be played in either a 19x19 grid, or 13x13 or a 9x9 board (We’ve implemented a 9x9 board for this project).

Initially the board is empty and completely vacant.

The game starts by players making moves on the empty positions in the board.

The objective of the game is to capture more territory.

A single stone located at the center of a board which is empty, has 4 liberties (vacant surroundings).

A stone can be captured if the opponent’s stones surrounds all of it’s 4 liberties.

The captured stone becomes the opponent’s player’s stone and is removed from the board.

A player can wish to pass his turn by not making any move and passing his turn and the opponent can make his move.

Consecutive passes from both the players indicate the end of the game.

The player who has captured more territory wins the game.

3. ARCHITECTURE OF THE PROJECT

This section discusses about the architecture used in our project and also the major components and functionalities used to build our project. The architecture model is a client-server architecture. The client or the front end uses React JS and Django templates for the user to interact with the game. The front end is connected to the back end where the game is built using Django framework which is connected to the PostgreSQL database. Nginx is configured and is an intermediary reverse proxy service through which the client interacts with the back end and all requests pass through the Nginx reverse Proxy.

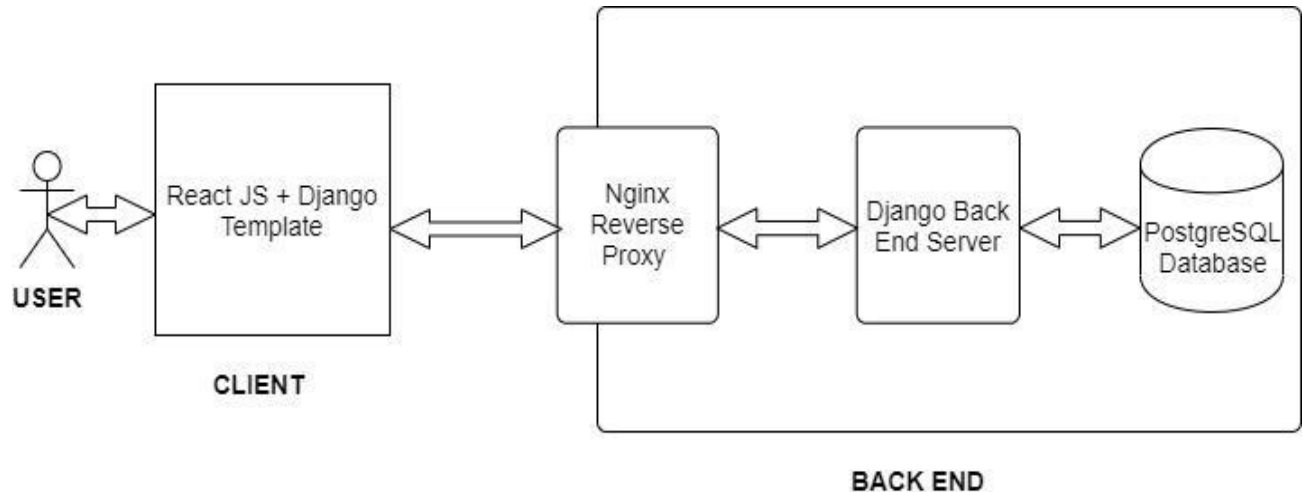


Figure 1. The client server architecture used in our Project.

3.1. COMPONENTS OF THE PROJECT

3.1.1 Django version 2.2.8

Django is a web application framework which is written using Python. It is based on the Model View Template pattern. It is a software pattern used for designing and developing a web application.

Model: It is responsible for maintaining data and it is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySQL, Postgres) [GEEKSDJANGO2019].

View: The View is the user interface — what you see in your browser when you render a website [GEEKSDJANGO2019].

Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted [GEEKSDJANGO2019].

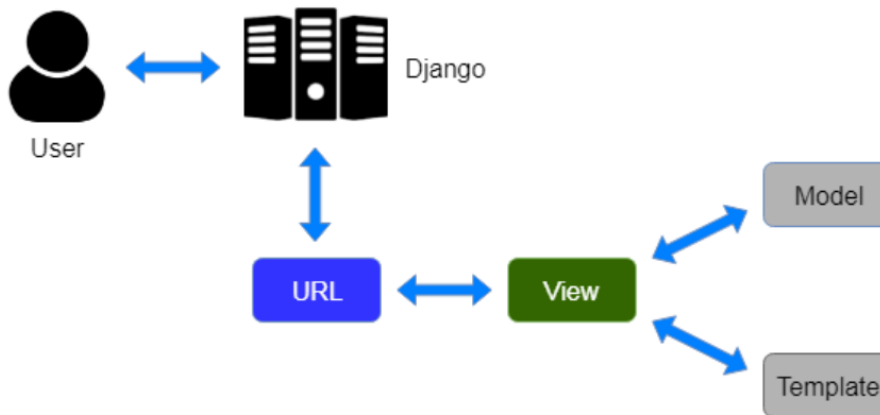


Figure 2. MVT architecture of Django [JAVAPoint2019].

3.1.2 Django Channels

Channels is a project that takes Django and extends its abilities beyond HTTP - to handle WebSockets, chat protocols, IoT protocols, and more and is built on a Python specification called ASGI [CHANNELS2019]. Django channels are used to enable web socket connection in the game and makes 2 player connection in the game possible.

In order to get a reference on how to build a game with web sockets using Django and React JS, we used a sample pen and paper game as a reference. [CODYPARKER2017].

3.1.3 Django Rest Framework

Django Rest Framework is a powerful framework which is used to build the web APIs. It is used in the application to develop the rest api calls and is used by front end framework to obtain the appropriate information.

3.1.4 React JS

React JS is a library using Javascript and is used to build front end interfaces. React JS is used in our project for the front end interface including the game, lobby, login, register pages.

3.1.5 PostgreSQL 11.5

PostgreSQL has been used as the database server. Django interacts with the database with its default ORM.

3.1.6 Django Webpack Loader

The webpack loader has been used to build the React JS files and create a static file for Django to use.

3.1.7 Node Package Manager

The node package manager is used for installing the dependencies for the node JS.

3.1.8 Nginx Reverse Proxy

Nginx is a reverse proxy which is an intermediary proxy service through which the clients interact with the back end server. All requests pass through the Nginx proxy and it delivers the requests to the server and passes the response to the client. Nginx is configured and sits right in front of the Django Back end server. There are various benefits using Nginx reverse proxy.

Load Balancing : It provides Load balancing service, where it helps distribute the requests from the client and distributes it to the server in an even fashion. This greatly helps by reducing the load and prevents the server from getting overloaded with a lot of requests.

Security : A reverse proxy also acts as a line of defense for your backend servers[KEYCDN2018]. A reverse proxy can protect the identity of the back end server and it remains completely unknown. The server is protected from attacks such as DDoS. The backend server is completely isolated from the public and users. Nginx reverse proxy handles all the HTTPS connections that are coming in, decrypts the requests, and passes the decrypted requests to the web server. This is a huge advantage as this eliminates the need to install the certificate in the back end. Nginx acts as a single point where configuration, management of TLS can be done. TLS in this project is configured for and using Nginx.

3.2 FUNCTIONALITIES

3.2.1 Model

- models.py

The models.py file contains the model classes for the three database tables such as Game, GameSquare and GameLog. This file handles all the database operations with PostgreSQL using the Django ORM.

3.2.2 View

- views.py

The views.py file has the display component for the front-end. It uses the template files to render the information on the website. It has view components for home, lobby and game page.

api_views.py

The api_views.py file has two REST API calls such as current-user and game-for-id. The current-user REST API is used to get the information about the current user to display it on the web application. The game-for-id REST API is used to the information about the game by the game_id value.

3.2.3 Template

- base.html

The base.html file has header and footer of the website. This base.html is extended to every page and the respective page content is rendered into this page.

- home.html

The home.html is the homepage of the web application which displays basic information about the game.

- login.html

The login.html has the form for the user login which accepts two inputs username and password.

- register.html

The register.html has the form for the user registration which uses CustomValidotor.py for the password validation.

- game.html

The game.html has the base template for the game page. It uses three react JS files such as GameBoard.jsx, GameLog.jsx, GameSquare.jsx that are used to render the go game board, game log and the player information.

- lobby.html

The lobby.html has the base template for the lobby page. It uses three react JS files such as AvailableGames.jsx, PastGames.jsx, PlayerGames.jsx to load the grid that displays available games, past game and current games on the lobby page.

3.2.4 Websocket

- consumer.py

The consumer.py handles the websocket calls which is enabled by the django channels package. The websocket connections are used to facilitate the two player functionality of the application.

4. INSTALLATION AND OPERATING INSTRUCTIONS

- First, install Python3:

brew install python3 // On Mac

sudo apt-get install python3 // On Ubuntu

- Normally, pip comes with python3 if you're downloading the latest version (or any version above 3.4). If that is not the case, install pip by running the following:

curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

python3 get-pip.py

- Install virtualenv:

pip3 install virtualenv

- Change directory to the project

cd ISA681_Go_Game

- To get the service running, run the following commands:

virtualenv -p `which python3` venv

source venv/bin/activate //for linux based systems

pip3 install -r requirements.txt

- Edit the database connection strings in the .env.example file

cp .env.example .env

- To install the npm webpack,

npm install

npm install webpack@2.7.0 --save-dev

- Execute the migrate command

python3 manage.py migrate

- If there is an error encountered(ERROR: ' No module named 'django.core.urlresolvers') in handler.py file (/venv/lib/python3.7/site-packages/channels/handler.py), replace the line

from django.core.urlresolvers import set_script_prefix

with

*from django.urls import **

- Run the client

npm run webpack

- Run the server

python3 manage.py runserver 8080

- Run the game by entering <https://localhost> in the browser.

5. ASSURANCE CASE

Protecting user data is one of the top priorities when designing any game or a web application for that matter. We have designed our game in such a way that a vast amount of the common threats related to web applications are eliminated. In the assurance case, claims and evidences are provided to describe and justify the security features implemented and to answer the primary question- “Why we believe the game is secure?”

5.1. SSL/TLS IMPLEMENTATION

CLAIM:

Both Secure Sockets Layer and Transport Layer Security are cryptographic protocols. Enabling these protocols ensures that the communication through the network is secure. Their main goals are to provide data integrity and communication privacy [ACUNETIXSSL 2019]. TLS is the successor of SSL. Modern browsers use TLS. Unencrypted communication can result in exposure of sensitive data ranging from user names, passwords and more. We have enabled TLS enabled connection in our game, which is a secure encryption channel, and communication from the client to the server is completely encrypted.

EVIDENCE:

TLS connection is configured, by using a self-signed certificate. Since our application is run locally, TLS certificate is created locally by creating a Certificate Authority and trusting it locally. So, eventually, any certificate created using the CA will be trusted locally. This is used to develop HTTPS connection for our local machine.

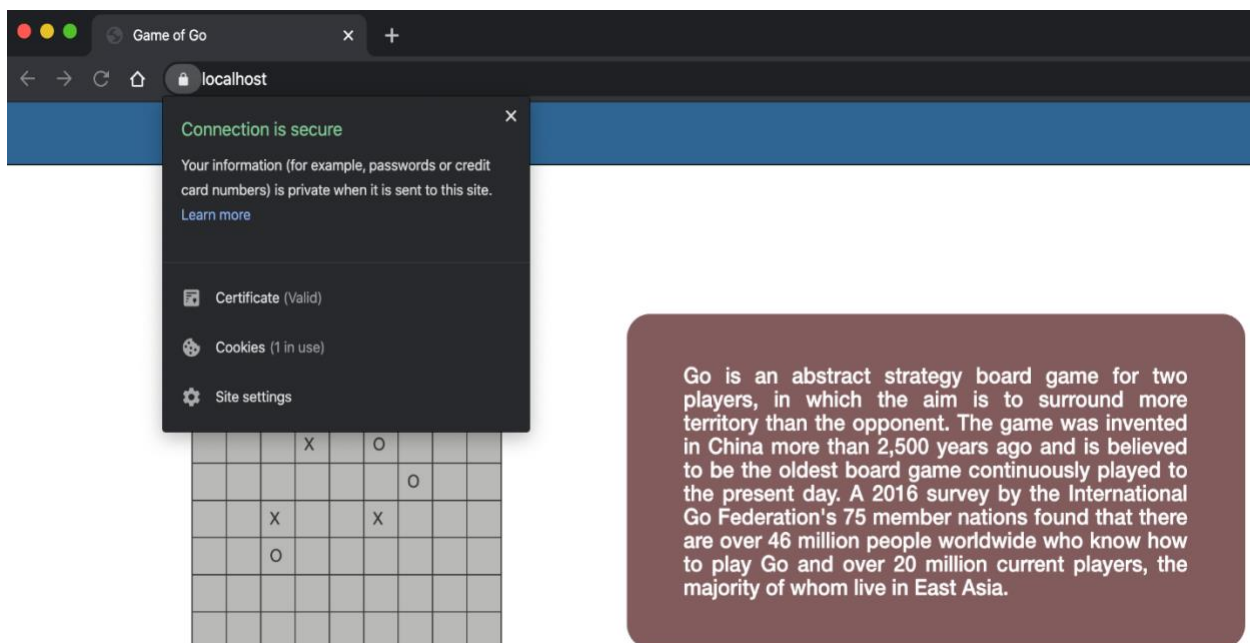


Figure 1. TLS Implementation and the connection is secure with HTTPS and a valid certificate.

Initially, we created a local certificate authority, and we trusted the certificate authority locally using Keychain access. Further, we created a local certificate for the CA, validated it and issued it using the CA we created earlier. The algorithm used here is RSA and the key size is 2048 bits. Using this method, a certificate, a local root CA and the private key for the certificate was obtained. We used these in hand to create a HTTPS connection locally.

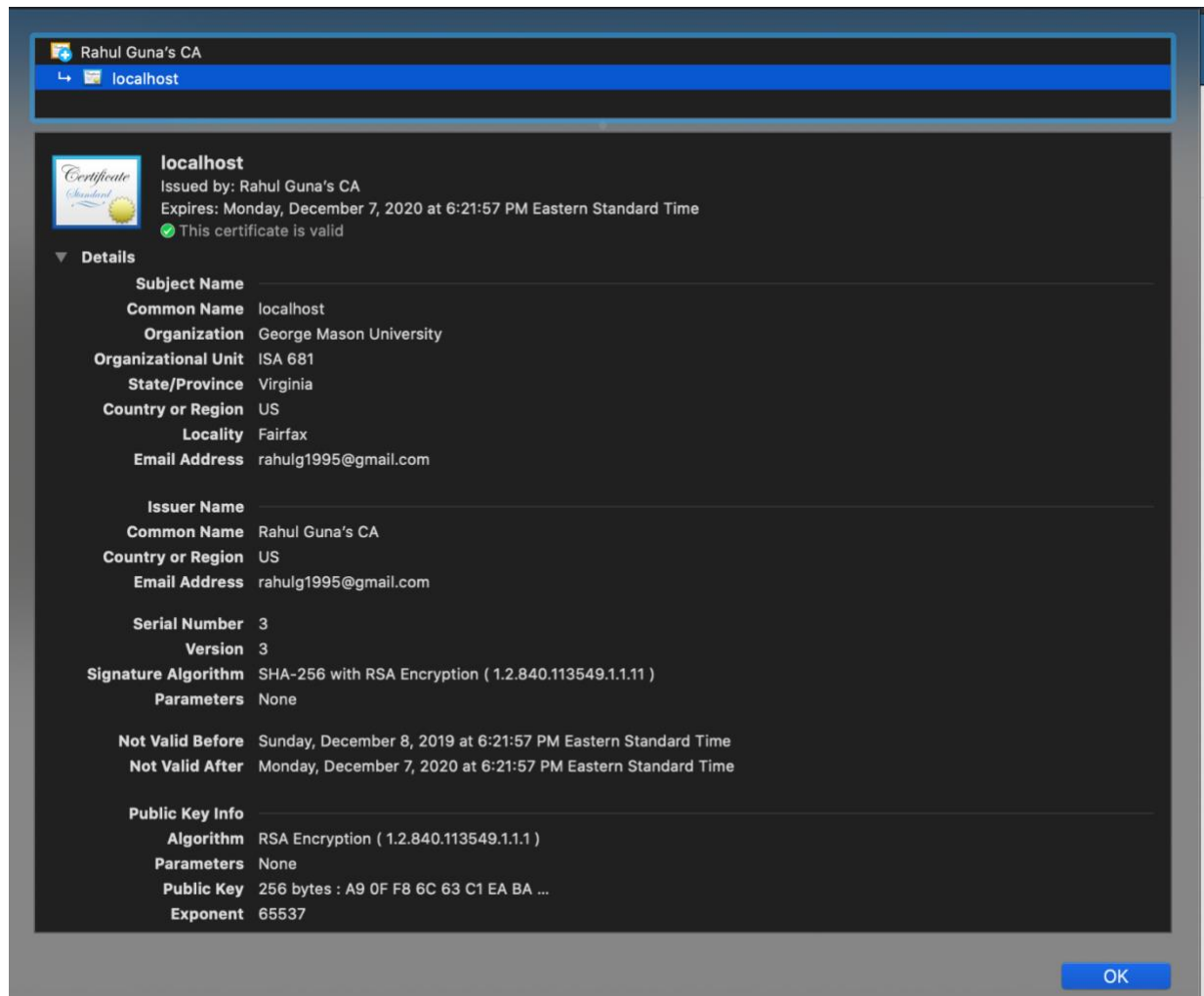


Figure 2. Details of the Certificate.

Nginx was also used in front of the Django back end server, which is a reverse proxy, it increases security by acting as a line of defense for the backend server. Configuring this reverse proxy makes sure that the identity of our Django back end server stays unknown. This helps in protecting the server from DDOS attacks. TLS was configured for and using Nginx in this case.

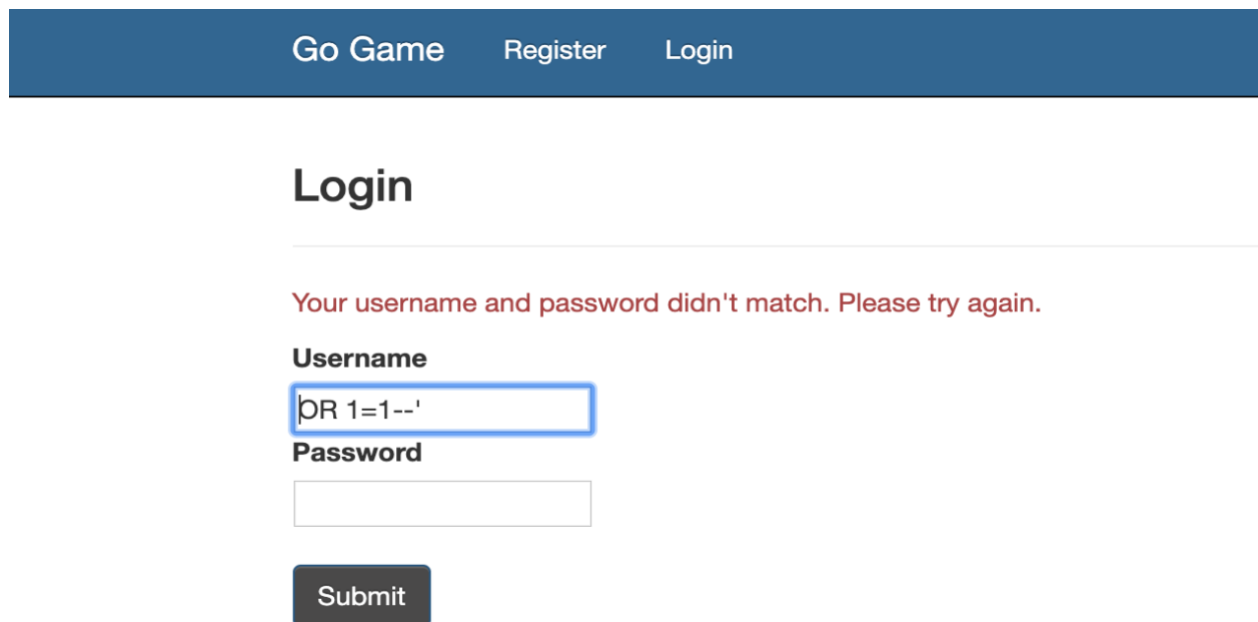
5.2 SQL INJECTION PROTECTION

CLAIM:

SQL Injection is a type of an attack where the intruder inserts data into the web application which will be given as input to a SQL interpreter at the back end. This is done in a way where data will be misunderstood. SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application [SQLACUNETIX2019]. We have implemented mechanisms to prevent SQL injection attacks.

EVIDENCE:

Our framework uses Django's Object-relational mapper (ORM) Layer, all the query sets are protected from the SQL injection attack. The queries here are developed and constructed using query parameterization. A query's SQL code is defined separately from the query's parameters and since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver [DJANGOSECURITY2019]. SQL Injection is best prevented through the use of parameterized queries [OWASPSQLCHEATSHEET].



The screenshot shows a web application interface with a dark blue header bar containing three links: "Go Game", "Register", and "Login". Below the header, the word "Login" is displayed in a large, bold, black font. A red error message reads: "Your username and password didn't match. Please try again." Below this message, there are two input fields. The "Username" field contains the text "OR 1=1--'". The "Password" field is empty. A dark grey "Submit" button is located below the input fields.

Figure 3. SQL Injection Prevention. Example attack attempted and blocked.

5.3. Cross-Site Request Forgery Protection

CLAIM:

Cross Site Request Forgery is a type of vulnerability that is extensively exploited in web applications. This vulnerability uses the server's trust in a client and the vulnerability exploits that trust. In Cross Site Request Forgery, the attacker fools and tricks the user to send some

data to server and the server acts upon the request and executes that action. Here, the server believes that the client truly intended to perform that particular action. We have implemented features and mechanisms to prevent this type of vulnerability

EVIDENCE:

Django has protection mechanisms against CSRF attacks. CSRF protection in Django works by using a check for a secret token in every POST request. This means that an adversary cannot perform “replay” actions on a form that triggers a POST to our web server and also prevents a logged in user from performing unintentional actions, for example submitting the form unintentionally. The malicious user would have to know the secret, which is user specific (using a cookie) [DJANGOSECURITY2019]. This is done through the MIDDLEWARE setting in Django. ‘django.middleware.csrf.CsrfViewMiddleware’ is included in the Middleware setting.

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django_session_timeout.middleware.SessionTimeoutMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
]
```

Figure 4. ‘django.middleware.csrf.CsrfViewMiddleware’ configured in middleware settings

CSRF cookies are based on secret random value, which is unique and no one else will have access to. This cookie will be set by the CsrfViewMiddleware. Cookies are sent with each response which calls the Django.middleware.csrf.get_token(), which is the function that is used to get the CSRF token. The token is just not secret, random salts will be added to the secret.

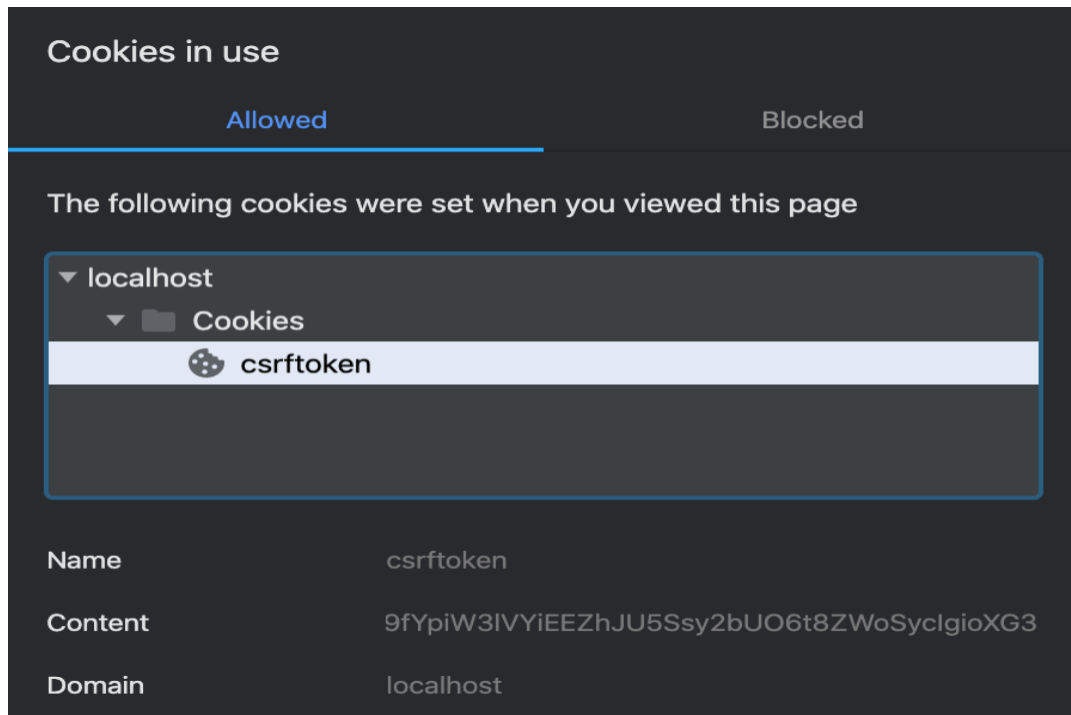
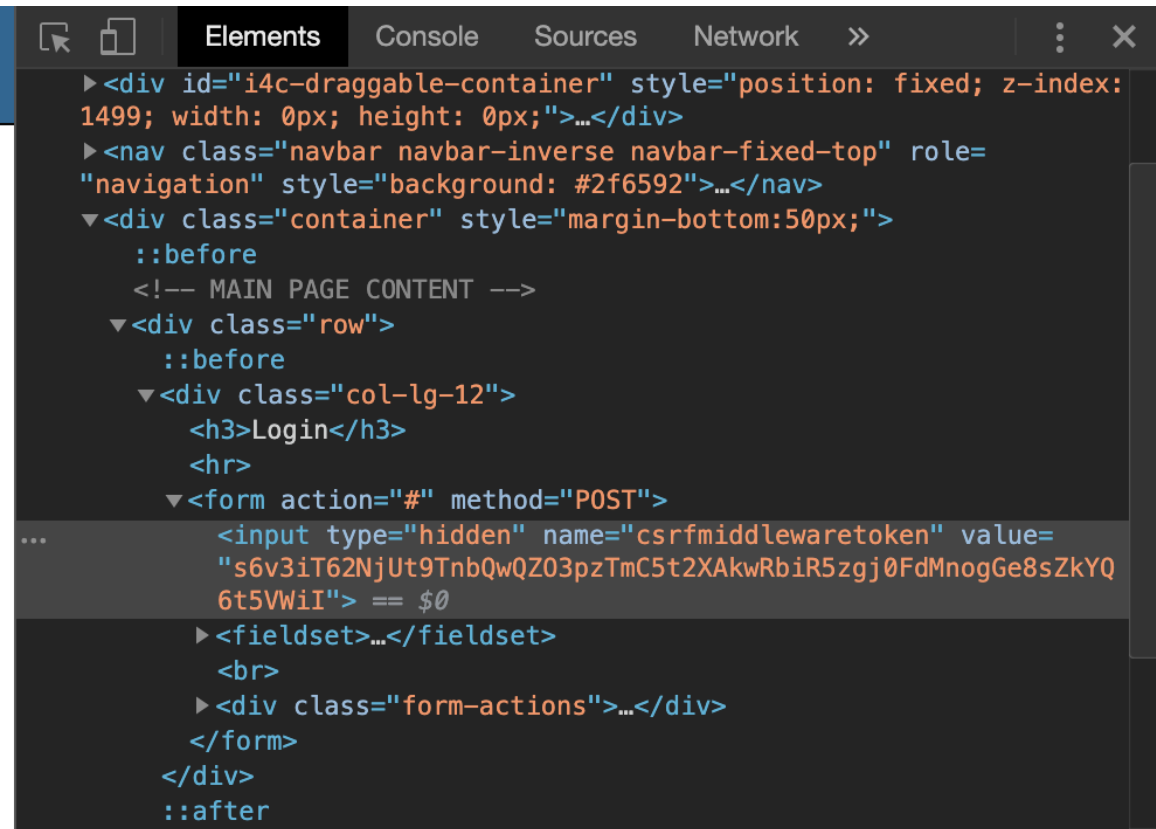


Figure 5. Csrf Token present in cookies for verification

Also, a hidden field in the form which is named 'csrfmiddlewaretoken' will be present in all the POST outgoing forms. The value of the field is secret and a random salt is added to it to make it even more random. Each time the call to get_token is done, the salt is regenerated. For all incoming requests that are not using HTTP GET, HEAD, OPTIONS or TRACE, a CSRF cookie must be present, and the 'csrfmiddlewaretoken' field must be present and correct. User will get a 403 error if it is not present. All this is being done by CsrfViewMiddleware.

For all HTTPS requests, referrer checking is strictly done by CsrfViewMiddleware.



```
><div id="i4c-draggable-container" style="position: fixed; z-index: 1499; width: 0px; height: 0px;">...</div>
><nav class="navbar navbar-inverse navbar-fixed-top" role="navigation" style="background: #2f6592;">...</nav>
▼<div class="container" style="margin-bottom:50px;">
  ::before
  <!-- MAIN PAGE CONTENT -->
  ▼<div class="row">
    ::before
    ▼<div class="col-lg-12">
      <h3>Login</h3>
      <hr>
      ▼<form action="#" method="POST">
        ...
        <input type="hidden" name="csrfmiddlewaretoken" value="s6v3iT62NjUt9TnbQwQZ03pzTmC5t2XAkWRbIR5zgJ0FdMnogGe8sZkYQ6t5VWii"> == $0
        <fieldset>...</fieldset>
        <br>
        <div class="form-actions">...</div>
      </form>
    </div>
  ::after
```

Figure 6. Csrfmiddlewaretoken hidden in forms.

5.4. Sensitive Data Exposure

More frequently known as a data breach, sensitive data exposure ranks as one of the top 10 most dangerous cyberthreats by OWASP (Open Web Application Security Project) because of the damage it can do to its victims [SITELOCK2019]. Sensitive data exposure happens when the application does not protect user data or sensitive information adequately. This involves confidential data ranging from credit card, passwords, and much more. We have implemented the following measures to prevent exposure of sensitive data.

5.4.1. Password Storage

CLAIM:

User's passwords are stored in the database using a salted hash system and they are not stored as the clear text.

EVIDENCE:

The passwords get stored in the database using PBKDF2 which is a password based key derivation function, which is a strong salted hashing function. PBKDF2 uses a pseudo random

function, for example HMAC(Hash based message authentication code), which is applied to the passphrase and adds a salt value and repeats the process multiple times to generate a derived key.

The password is stored as a string in the following format

<algorithm>\$<iterations>\$<salt>\$<hash>

Separated by the dollar sign, the format contains the hashing algorithm, number of iterations, and the random salt value, also the password hash which gets generated.

Data Output	Explain	Messages	Notifications		
id	password	last_login	is_superuser	username	
[PK] integer	character varying (128)	timestamp with time zone	boolean	character varying (150)	
1	4 pbkdf2_sha256\$150000\$EyEhtvySZpXB\$ZjB0EgeVShmVS2r...	2019-12-12 13:47:16.586521-05	false	Bharat1	
2	5 pbkdf2_sha256\$150000\$LQsh59lq6aYj\$Q9dyW+YnzZsIME0...	2019-12-12 13:49:30.430255-05	false	Rahul1	
3	2 pbkdf2_sha256\$150000\$HlxdHGYOkApf\$NqYIUL9r1126c3C...	2019-12-12 16:29:44.464135-05	false	Varun	
4	1 pbkdf2_sha256\$150000\$wugAT77fTBg7\$hA12b17ug5DoW...	2019-12-12 16:45:39.542694-05	false	Rahul	
5	6 pbkdf2_sha256\$150000\$EMrrvxnCYiN9\$ARKpXAUhPSaD9u...	2019-12-12 16:46:12.663671-05	false	Intruder	
6	3 pbkdf2_sha256\$150000\$HHEmGY8DNpJX\$w/wZXWjy6mXn...	2019-12-09 02:14:28.436063-05	false	Bharat	

Figure 7. Passwords are salted and hashed in the database.

By default, Django uses the PBKDF2 algorithm with a SHA256 hash, a password stretching mechanism recommended by NIST [PASSDJANGO2019]. OWASP recommends the following, Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt, PBKDF2 [OWASPSENSITIVE2019].

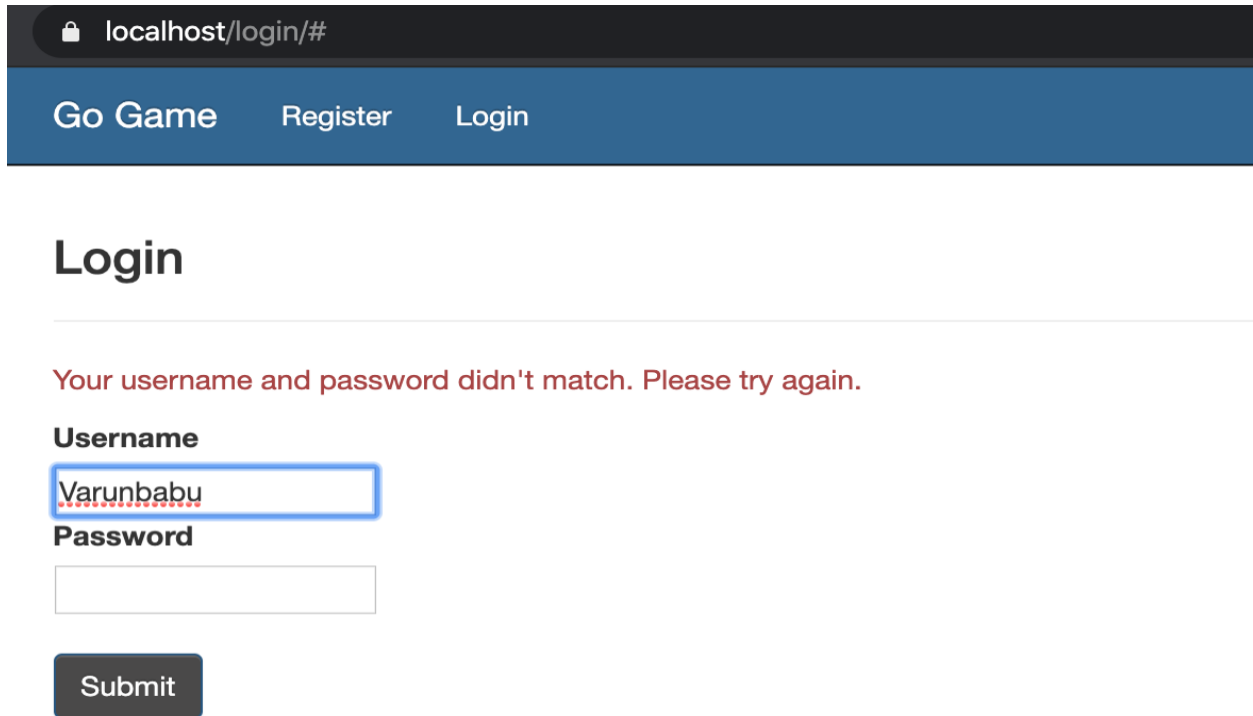
```
PASSWORD_HASHERS = [  
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',  
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',  
    'django.contrib.auth.hashers.Argon2PasswordHasher',  
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',  
]
```

Figure 8. PASSWORD_HASHERS setting in Django.

This is done by using the PASSWORD_HASHERS setting in Django. It includes the list of various hashing algorithms that Django supports. We enabled the setting to use the PBKDF2 hashing algorithm on passwords .

5.4.2. Details of Failed Login

If a user provides an incorrect username and password, the web application must not reveal sensitive details about the failure, for example if a user enters the wrong password, the web application must not reveal that the username is correct and the password is wrong. Instead, generic information must be provided to the user, stating the credentials are incorrect.



The screenshot shows a web application interface. At the top, there is a dark header bar with a lock icon and the text 'localhost/login/#'. Below this is a blue navigation bar with three links: 'Go Game', 'Register', and 'Login'. The main content area has a large heading 'Login'. Below the heading, a red error message reads: 'Your username and password didn't match. Please try again.' Underneath the message, there are two input fields. The first is labeled 'Username' and contains the text 'Varunbabu'. The second is labeled 'Password' and is empty. A dark 'Submit' button is located below the password field.

Figure 9. Generic Error message displayed instead of specifics.

5.5. Audit Trail and Game-move validation

CLAIM:

An audit trail of every game move is created and recorded, these game moves and history of win/loss statistics along with every move in a particular game are available to the users who played the game. This is not available to the users who did not play the game. Also, these moves are only available to the ones who are authenticated.

The game is designed in such a way that players cannot make illegal moves. If a player makes a move at a specific location, the opponent player cannot make the same move at the exact location since it is an invalid move. This prevents the players from making illegal moves and cheating through move validation.

EVIDENCE:

One the user logs in, he is provided with the following:

- Start a new game
- Join an existing game through available games
- Look at win/loss history (Flag sign) of previous games.
- View Past games and game moves of completed games.

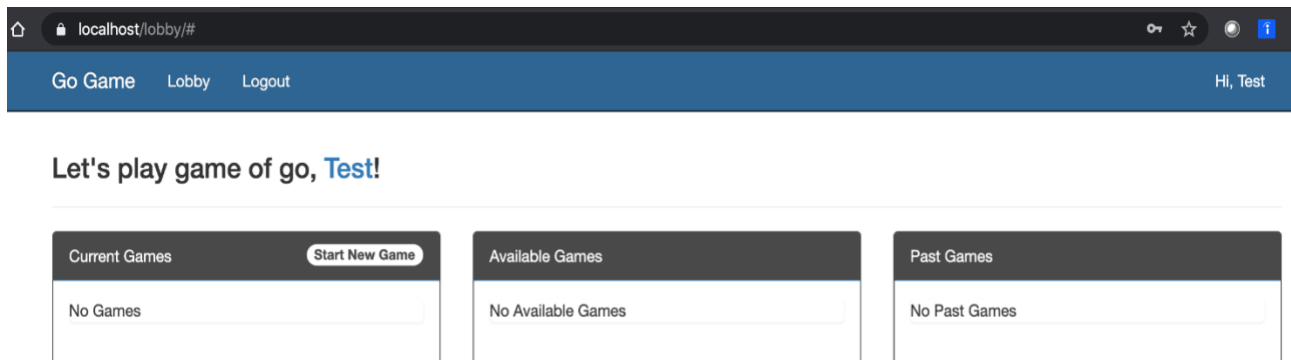


Figure 10. User provided with the above options in the Lobby.

Once 2 players start playing a game, every single game move made by both the users are logged in a table in the database. Also, it is available to the user while in the game screen to track their previous moves within the game. The data in the database shows which player has claimed which square with a time stamp and time zone.

ip	Data Output	Explain	Messages	Notifications
ip_permissions				
nission				
groups				
_user_permissions				
dmin_log				
ontent_type				
migrations				
ession				
me				
melog				
mesquare				
ins				
raints				
as				
rs				
tions				

	id	text	created	game_id
	[PK] integer	character varying (300)	timestamp with time zone	integer
	3	Square claimed at (6, 6) by Test2	2019-12-12 17:23:55.795075-05	1
	4	Square claimed at (4, 2) by Test	2019-12-12 17:23:58.662995-05	1
	5	Square claimed at (1, 6) by Test2	2019-12-12 17:24:02.500552-05	1
	6	Square claimed at (6, 1) by Test	2019-12-12 17:24:06.102893-05	1
	7	Square claimed at (5, 4) by Test2	2019-12-12 17:24:18.838701-05	1
	8	Square claimed at (2, 5) by Test	2019-12-12 17:24:23.235344-05	1
	9	Square claimed at (7, 7) by Test2	2019-12-12 17:24:26.52952-05	1
	10	Square claimed at (3, 4) by Test	2019-12-12 17:24:29.888386-05	1
	11	Square claimed at (4, 4) by Test2	2019-12-12 17:24:33.89994-05	1
	12	Square claimed at (5, 5) by Test	2019-12-12 17:24:37.837882-05	1
	13	Square claimed at (5, 7) by Test2	2019-12-12 17:24:40.76286-05	1
	14	Square claimed at (6, 4) by Test	2019-12-12 17:24:43.719965-05	1
	15	Square at (5, 4) is captured by Test2	2019-12-12 17:24:51.930315-05	1
	16	Square claimed at (5, 3) by Test2	2019-12-12 17:24:51.931463-05	1

Figure 11. Each player move and square capture is recorded in the database.

Test
(O)

VS

Test2
(X)

Game Info

Current Turn: Test

Pass Your turn: Pass

Log

Say It

System	Square claimed at (5, 3) by Test2
System	Square at (5, 4) is captured by Test2
System	Square claimed at (2, 8) by Test
System	Square claimed at (6, 4) by Test2
System	Square claimed at (3, 7) by Test
System	Square claimed at (5, 5) by Test2
System	Square claimed at (5, 8) by Test

Figure 12. Each move appears in the Log section for both the players to track previous moves in an active game.

Once a game is completed, the past games are available to the user in the lobby, given that he is authenticated. In the past games, a small flag is included beside the games he has won. No flag indicates he has lost the game. When he clicks on the past game, history of moves made are available to the user.

Here, the authenticated user has read-only access, he cannot modify his past scores and moves in any way.

Hi, player1

Available Games

No Available Games

Past Games

1

player2 vs player1

|

View

Figure 13. Past games section available for the authenticated user. The flag indicates the user has won that game. No flag indicates the user has lost that game.

Go Game
Lobby
Logout
Hi, Test

Playing Game of Go!

Test
(O)

VS

Test2
(X)

Game Info

Winner: **Test2**

Score: **6**

Log

- System** Square claimed at (6, 6) by Test2
- System** Square claimed at (4, 2) by Test
- System** Square claimed at (1, 6) by Test2
- System** Square claimed at (6, 1) by Test
- System** Square claimed at (5, 4) by Test2
- System** Square claimed at (2, 5) by Test
- System** Square claimed at (7, 7) by Test2

Figure 14. Authenticated user can view his past games, track moves and scores.

Also, each move made by every player is updated in the database. Each position captured by a player is updated in the database and this makes validation easier and hence a position acquired by player 1 cannot be undone or over-written by player 2.

Data Output		Explain	Messages	Notifications				
	id [PK] integer	status character varying (25)	row integer	col integer	created timestamp with time zone	modified timestamp with time zone	game_id integer	owner_id integer
143	153	Free	7	8	2019-12-12 17:33:45.826144-05	2019-12-12 17:33:45.826157-05	2	[null]
144	154	Free	8	0	2019-12-12 17:33:45.827029-05	2019-12-12 17:33:45.827054-05	2	[null]
145	155	Free	8	1	2019-12-12 17:33:45.827928-05	2019-12-12 17:33:45.827946-05	2	[null]
146	157	Free	8	3	2019-12-12 17:33:45.830011-05	2019-12-12 17:33:45.830029-05	2	[null]
147	158	Free	8	4	2019-12-12 17:33:45.830915-05	2019-12-12 17:33:45.830932-05	2	[null]
148	160	Free	8	6	2019-12-12 17:33:45.832503-05	2019-12-12 17:33:45.83252-05	2	[null]
149	161	Free	8	7	2019-12-12 17:33:45.833192-05	2019-12-12 17:33:45.833209-05	2	[null]
150	162	Free	8	8	2019-12-12 17:33:45.833875-05	2019-12-12 17:33:45.833892-05	2	[null]
151	121	Selected	4	3	2019-12-12 17:33:45.791585-05	2019-12-12 17:33:45.791601-05	2	1
152	142	Selected	6	6	2019-12-12 17:33:45.818871-05	2019-12-12 17:33:45.818885-05	2	2
153	104	Selected	2	4	2019-12-12 17:33:45.769815-05	2019-12-12 17:33:45.769832-05	2	1
154	147	Selected	7	2	2019-12-12 17:33:45.822259-05	2019-12-12 17:33:45.822273-05	2	2
155	122	Selected	4	4	2019-12-12 17:33:45.792989-05	2019-12-12 17:33:45.793005-05	2	2

Figure 15. Data for each square recorded in the database, free status denotes the square is free, selected status denotes moves made on that square(row, column) in specific game_id which is made by user (owner_id). Users cannot cheat and make illegal moves.

5.6. Session Management

CLAIM:

Session management is a process where multiple requests to a web application from a user are securely handled. Browsers and websites use HTTP to communicate, and a web session is a series of HTTP requests and response transactions created by the same user and Since HTTP is a stateless protocol, where each request and response pair is independent of other web interactions, each command runs independently without knowing previous commands [WHITEHATSEC2019]. Incorrect session management may lead to confidential data being compromised. Session management is implemented in our game to overcome these problems.

EVIDENCE:

Each time a user is authenticated and logs in to the game, a session is created and the data is being stored in cookies. The cookies contain the session ID. Each time the user logs out, the session gets expired and the session data is cleared. Using Django, it stores the data on the server side of things and completely abstracts the send/receive process of cookies.

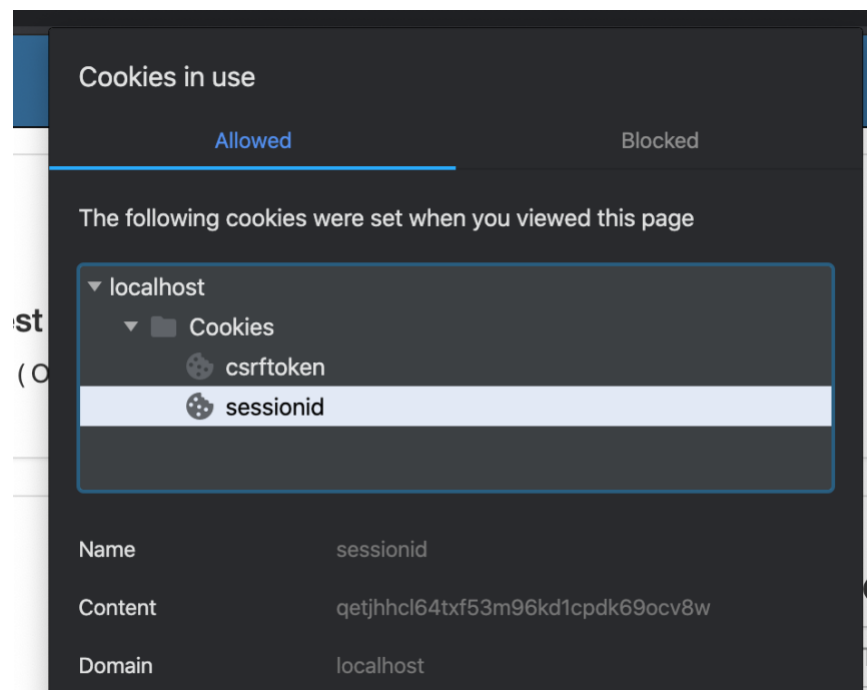


Figure 16. Session id generated upon login.

Cookies are enabled by using the middleware setting, by using 'django.contrib.sessions.middleware.SessionMiddleware' in the settings. The session is also set to expire in 180 seconds (3 minutes) of inactivity. This means if the user is inactive for 180 seconds, the session is expired and the user is navigated back to the login page upon clicking

the page. This is done by using `SESSION_EXPIRE_SECONDS` and setting it as 180, and also using `SESSION_EXPIRE_AFTER_LAST_ACTIVITY` and setting it to `TRUE`.

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    .... 'django_session_timeout.middleware.SessionTimeoutMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Figure 17. Session configured and enabled in Middleware settings.

```
SESSION_EXPIRE_SECONDS = 180
SESSION_EXPIRE_AFTER_LAST_ACTIVITY = True
```

Figure 18. Session expire settings configured.

Also, each time the user starts playing a game, all the moves he makes are being stored in the database. Once a player1 makes a move, now it is the turn of the player2 to make the move. The time limit for the player2 to make the move is 180 seconds. This is done by calculating the time difference after player 1 completes the move and the time taken by player 2 to make the next move. If player 2 does not make the move in 180 seconds, the session gets expired and when he logs back in, the game will be forfeited and the opponent wins the game

```
def claim(self, status_type, user):
    """
    Claims the square for the user
    """
    log = GameLog.objects.filter(game=self.game).aggregate(Max('id'))
    print(log)
    if(log['id_max'] != None):
        log2 = GameLog.objects.get(id=log['id_max'])

        datetimeFormat = '%Y-%m-%d %H:%M:%S.%f'
        date1 = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')
        date2 = log2.created.strftime('%Y-%m-%d %H:%M:%S.%f')
        diff = datetime.datetime.strptime(date1, datetimeFormat) - datetime.datetime.strptime(date2, datetimeFormat)

        print("Seconds:", diff.seconds)

        if(diff.seconds>180):
            print("inside the loop ")
            self.game.current_turn = self.game.creator if self.game.current_turn != self.game.creator else self.game.opponent
            self.game.add_log('Game was forfeited because of session timeout. {0} has won the game'.format(self.game.current_turn))
            self.game.send_game_update()
            self.game.mark_complete(self.game.current_turn)
            return

    self.owner = user
    self.status = status_type
    self.save(update_fields=['status', 'owner'])
```

Figure 19. Implementing Time difference Calculation.

Also, if the user experiences any connectivity problems, he can log back in and continue the game, given that the time frame to log back in the game is within 180 seconds. If it pasts the time mark, then the game is forfeited and the opponent wins here as well. This is how we designed the game.

In another case, if the user is idle for 3 minutes, or if he is wandering around in the lobby without making a move inside the game within 3 minutes, the session expires and the game is forfeited automatically and the opponent wins.

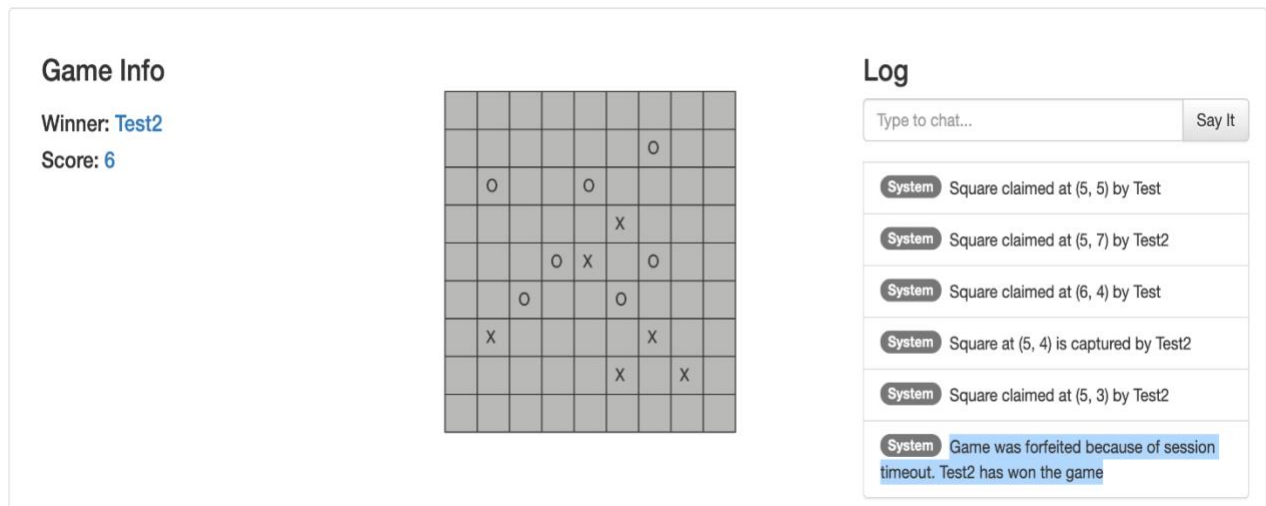


Figure 20. Game forfeited due to logout/inactivity for 180 seconds caused by session expiry.

Once the user logs out of the game, the session gets expired and all session data is cleared. After logging out, the same URL used when the user was logged in cannot be used to get in since the session is expired. The web application would ask the user to log back in.

Also, the URL used by player 1 and player 2 to play the game cannot be used by a third person (player 3 – intruder) to intrude into the game which is being played by 2 other different players.

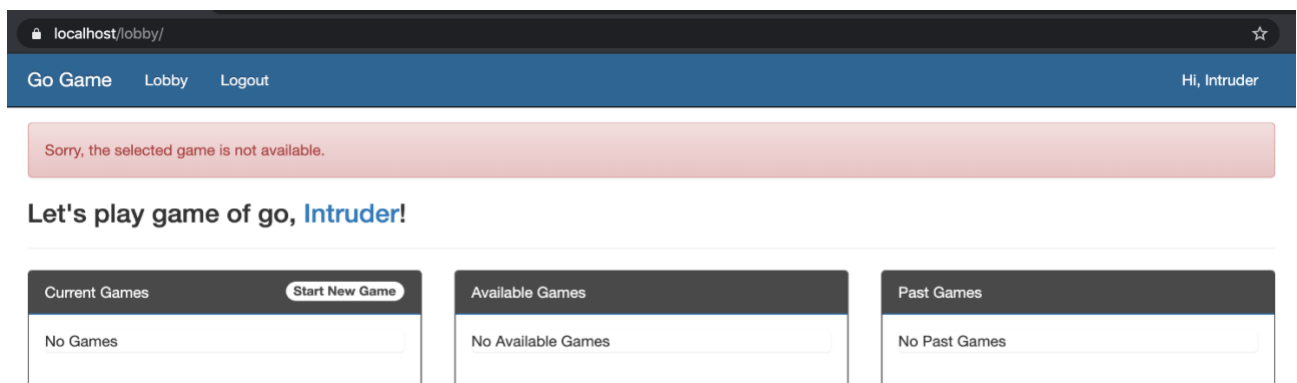


Figure 21. Error Message - Third player cannot join an existing game using the game URL.

5.7. Click Jacking Protection

CLAIM:

Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. This attack can occur when a website fools the user in to clicking on an element which belongs to another site which is loaded through a hidden frame or an iframe. We have implemented measures to overcome this problem of Clickjacking

EVIDENCE:

Django uses a middleware for clickjacking for protection against clickjacking attacks. Modern browsers have the X-Frame-Options HTTP header which shows if a resource can be loaded within an iframe or frame. If the response contains SAMEORIGIN as the value in the header, then the browser will load it in a frame only if the request is being originated from the same site. If it contains DENY, then any kind of resource loading in a frame will be blocked, no matter the origin of the request. In Django, the middleware is only responsible for setting the X-Frame-Options HTTP header. We added 'django.middleware.clickjacking.XFrameOptionsMiddleware' to the Middleware settings in our project. Also, the X_FRAME_OPTIONS is set to 'Deny' for all outgoing responses.

This can protect our web application from being rendered through a concealed layer in a frame.

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'django_session_timeout.middleware.SessionTimeoutMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]  
  
X_FRAME_OPTIONS = 'DENY'
```

Figure 22. Middleware settings for XFrameOptions and options set to 'DENY'.

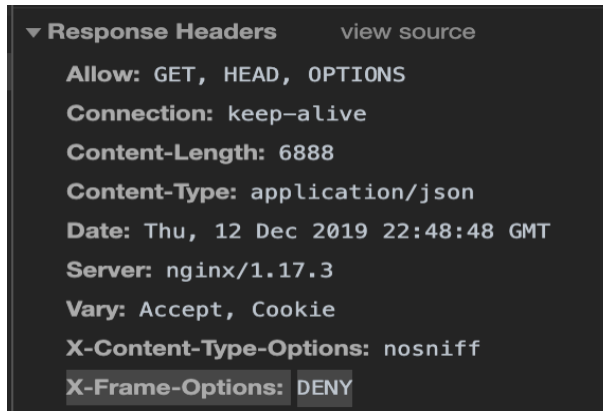


Figure 23. X-Frame-Options set to DENY in HTTP header.

5.8. Cross-Site Scripting and Input Validation

CLAIM:

Cross site scripting is a type of an attack which gives the user the capability to inject scripts in the browsers in the client side of other users. This attack occurs when these malicious scripts are stored in the database, and ultimately retrieved and displayed to the legitimate users. There are other ways where users are tricked in to clicking malicious links which will cause the malicious script to be executed in the browser of the user. This type of attack fools the user and the client (browser), the client believes that the server intended to send whatever data it sent. We've implemented measures to counter this type of vulnerability.

EVIDENCE:

Using Django templates protects you against the majority of XSS attacks[DJANGOSECURITY2019]. Using Django templates helps since it automatically escapes various characters which are specifically dangerous to Hyper Text Markup Language. The XSS Protection header along with the block mode provides some security against cross-site scripting attacks. Although modern web browsers have a good Content security policy, it is a good practice to have multiple security measures and defense in breadth. Using this method results in the browser preventing rendering of a web page in case of an attack.

To activate this header, we used the 'django.middleware.security.SecurityMiddleware' and included it in the middleware settings. Also, we set `SECURE_BROWSER_XSS_FILTER = True` and by doing this, xss protection header can be seen in the response headers.

```
X_FRAME_OPTIONS = 'DENY'
SECURE_BROWSER_XSS_FILTER = True
```

Figure 24. Security Middleware setting and XSS filter set to true in Settings.

Input validation is also implemented as a protection measure and validate all input from the user to help eliminate cross site scripting attacks.

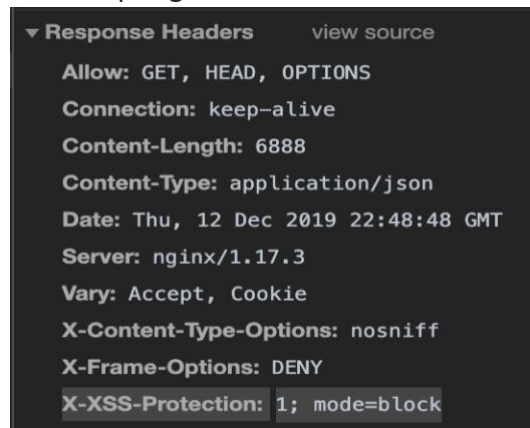


Figure 25. XSS-Protection enabled and block mode set in HTTP header.

5.8.1 INPUT VALIDATION

CLAIM:

Improper input validation is one of the common vulnerabilities in today's world of web applications and security. All input from the user or a source must be validated and filtered. All input must be treated as untrusted input. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application[CWE20MITRE2019].

We've implemented input validation mechanism in our application to validate all input from the user and making sure we counter attacks caused by improper input validation such as buffer overflow, cross-site scripting and directory traversal.

EVIDENCE:

We have implemented regular expressions to validate input from the user. Using regular expressions with whitelisting is a better option for a lot of reasons. Few of them being, the attacker does not get much room to work his attacks, using characters that we know can make our lives of validating input a lot more easier. A white list is a pattern which tells us and defines about all input which must be allowed and other input which are not allowed must be escaped or rejected.

Using regular expressions as whitelisting mechanism, we have set criteria for input validation as shown below:

```

class LowercaseValidator(object):
    def validate(self, password, user=None):
        if not re.findall('[a-z]', password):
            raise ValidationError(
                _("The password must contain at least 1 lowercase letter, a-z."),
                code='password_no_lower',
            )

    def get_help_text(self):
        return _(
            "Your password must contain at least 1 lowercase letter, a-z."
        )

class SymbolValidator(object):
    def validate(self, password, user=None):
        if not re.findall('[()[]{}|\\"~!@#%&*_-+\\',./?]', password):
            raise ValidationError(
                _("The password must contain at least 1 symbol: " +
                  "()[ ]{|~!@#%&*_-+\\',./?")
            )

    def get_help_text(self):
        return _(
            "Your password must contain at least 1 symbol: " +
            "()[ ]{|~!@#%&*_-+\\',./?"
        )

```

Figure 26. Regular expressions defined to validate input.

In the above image, the first validation is for passwords to have one lower case letter at-least. The second validation is for the password to have atleast one special character or symbol which includes characters shown above.

```

import re

from django.core.exceptions import ValidationError
from django.utils.translation import ugettext as _

class NumberValidator(object):
    def validate(self, password, user=None):
        if not re.findall('[0-9]', password):
            raise ValidationError(
                _("The password must contain at least 1 digit, 0-9."),
                code='password_no_number',
            )

    def get_help_text(self):
        return _(
            "Your password must contain at least 1 digit, 0-9."
        )

class UppercaseValidator(object):
    def validate(self, password, user=None):
        if not re.findall('[A-Z]', password):
            raise ValidationError(
                _("The password must contain at least 1 uppercase letter, A-Z."),
                code='password_no_upper',
            )

    def get_help_text(self):
        return _(
            "Your password must contain at least 1 uppercase letter, A-Z."
        )

```

Figure 27. Regular Expressions to validate Passwords.

In the above image, our 3rd and 4th password validations ensures the password must include at-least one digit from 0-9 and at-least one upper case letter.

In our application, Django allows us to configure password validations through the AUTH_PASSWORD_VALIDATORS setting. Django has a few included built in password validators which are the following :

- UserAttributeSimilarityValidator – this checks for similarity between the user name and the password.
- MinimumLengthValidator – this checks if the password meets the minimum length. The default value is 8 characters.
- CommonPasswordValidator – this checks if the password provided matches a list of 20,000 common passwords which comes included with Django by default.

These validators can be configured in the AUTH_PASSWORD_VALIDATORS setting in the settings file. We have included the above-mentioned validators for validation along with our custom created validator created using Regular Expressions (Regex). Django gives the flexibility to create as many custom validators. The custom validators are added to the AUTH_PASSWORD_VALIDATORS setting by including the path for the validator.

```
AUTH_PASSWORD_VALIDATORS = [  
    {  
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',  
    },  
    {  
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',  
    },  
    {'NAME': 'game.validators.CustomPasswordValidator.NumberValidator', },  
    {'NAME': 'game.validators.CustomPasswordValidator.UppercaseValidator', },  
    {'NAME': 'game.validators.CustomPasswordValidator.LowercaseValidator', },  
    {'NAME': 'game.validators.CustomPasswordValidator.SymbolValidator', },  
]
```

Figure 28. In-built and custom validators added to the AUTH_PASSWORD_VALIDATORS setting.

Register

Username

Password

Password confirmation

This password is too short. It must contain at least 8 characters. The password must contain at least 1 digit, 0-9. The password must contain at least 1 uppercase letter, A-Z. The password must contain at least 1 symbol: `~!@#\$%^&*._+,-/?`

Submit

Figure 29. Input Validation seen in the application.

5.9. Static and Dynamic Tools for Vulnerability Scanning

Bandit, which is a comprehensive code vulnerability scanner for Python, was used to scan for vulnerabilities in code. Bandit is a tool used to scan for common security vulnerabilities in code.

To do this Bandit processes each file, builds an AST from it, and runs appropriate plugins against the AST nodes. Once Bandit has finished scanning all the files it generates a report [PYPIBANDIT2019].

Bandit was installed using the 'pip install bandit' command.

Our whole project repository was examined using Bandit.

```
Go_Game — -bash — 190x42
[Rahuls-MBP:Go_Game rahulguna$ bandit -r ISA681_Go_Game/
[main] INFO profile include tests: None
[main] INFO profile exclude tests: None
[main] INFO cli include tests: None
[main] INFO cli exclude tests: None
[main] INFO running on Python 2.7.16
Run started:2019-12-12 19:43:09.983625

Test results:
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: '76t_o&oi@e!^@9age&8)1=@0bb71a_(19=)%l4k-k&um96l+w3'
Severity: Low Confidence: Medium
Location: ISA681_Go_Game/go_game/settings.py:10
More Info: https://bandit.readthedocs.io/en/latest/plugins/b105_hardcoded_password_string.html
9 # SECURITY WARNING: keep the secret key used in production secret!
10 SECRET_KEY = '76t_o&oi@e!^@9age&8)1=@0bb71a_(19=)%l4k-k&um96l+w3'
11
12 # SECURITY WARNING: don't run with debug turned on in production!
13 DEBUG = True

-----

Code scanned:
Total lines of code: 809
Total lines skipped (#nosec): 0

Run metrics:
Total issues (by severity):
Undefined: 0
Low: 1
Medium: 0
High: 0
Total issues (by confidence):
Undefined: 0
Low: 0
Medium: 1
High: 0

Files skipped (0):
Rahuls-MBP:Go_Game rahulguna$
```

Figure 29. Initial Test Result of Bandit – One low risk error.

An error with Low severity was found which said “Possible hardcoded password”. This is because the secret key for Django was present in the settings file and it was hardcoded.

Measures taken to eliminate this risk:

The hardcoded password was stored in an .env file. The .env file contains a key and value pair which is pulled from the .env file each time the application is stored. This is stored locally. Decouple is a package that can be used to retrieve the key name and value pair from the .env file. This is done using the `config('KEY_NAME')` command. The secret key is now stored in the .env file. So, this is a method to remove the hardcoded values in the settings file. The actual password is replaced by the config command as seen below in the image.

```
SECRET_KEY = config('DJANGO_SECRET_KEY')
```

Figure 30. Secret key no longer hardcoded in settings file.

After fixing this issue, we ran Bandit again, upon completing the scan, there were no issues to be found in the test results as shown in the image below.

```
[Rahuls-MBP:Go_Game rahulguna$ bandit -r ISA681_Go_Game/
[main] INFO profile include tests: None
[main] INFO profile exclude tests: None
[main] INFO cli include tests: None
[main] INFO cli exclude tests: None
[main] INFO running on Python 2.7.16
Run started:2019-12-12 19:52:26.799104

Test results:
    No issues identified.

Code scanned:
    Total lines of code: 809
    Total lines skipped (#nosec): 0

Run metrics:
    Total issues (by severity):
        Undefined: 0
        Low: 0
        Medium: 0
        High: 0
    Total issues (by confidence):
        Undefined: 0
        Low: 0
        Medium: 0
        High: 0
Files skipped (0):
Rahuls-MBP:Go_Game rahulguna$
```

Figure 31. Test results after fixing the vulnerability.

We also used OWASP ZAP as our dynamic scanning tool to scan for vulnerabilities. The Zed Attack Proxy tool is one of the most popular tools for scanning web applications for vulnerabilities. We performed scans on our web applications using both Standard Mode and Attack mode.

The results of the scans are as shown in the image below:

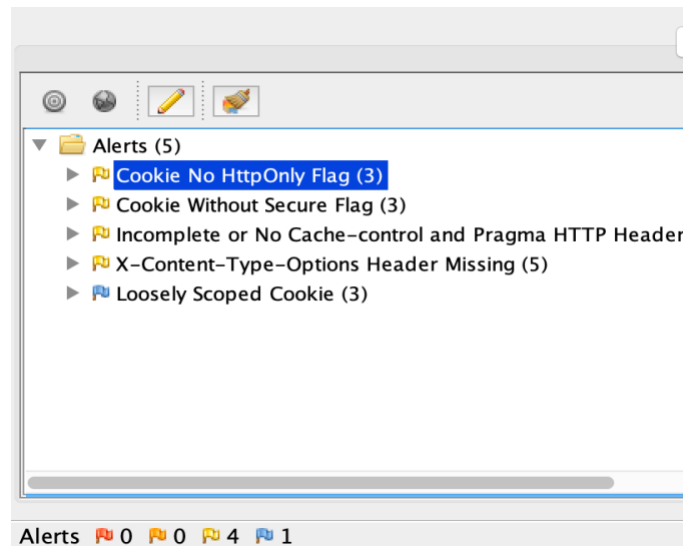


Figure 32. Test results of OWASP ZAP – 4 low risk vulnerabilities.

4 low risk vulnerabilities were found.

Measures taken:

- Cookie No HttpOnly Flag - The HttpOnly flag directs compatible browsers to prevent client-side script from accessing cookies and including the HttpOnly flag in the Set-Cookie HTTP response header helps mitigate the risk associated with Cross-Site Scripting (XSS) where an attacker's script code might attempt to read the contents of a cookie and exfiltrate information obtained [CWE-10042019].
CSRF_COOKIE_HTTPONLY=TRUE and SESSION_COOKIE_HTTPONLY = TRUE were set to eliminate this risk.
- Cookie without Secure Flag - If the secure flag is not set, then the cookie will be transmitted in clear-text if the user visits any HTTP URLs within the cookie's scope. An attacker may be able to induce this event by feeding a user suitable links, either directly or via another web site [PORTSWIGGER2019]. CSRF_COOKIE_SECURE =TRUE was set to eliminate this risk.
- X-Content-Type-Options Header Missing – There is a possibility that the web application is vulnerable to MIME sniffing since the Header is missing.
SECURE_CONTENT_TYPE_NOSNIFF = TRUE was set to eliminate this risk.

```
CSRF_COOKIE_SECURE = True
SECURE_CONTENT_TYPE_NOSNIFF = True
SESSION_COOKIE_HTTPONLY = True
|CSRF_COOKIE_HTTPONLY = True
```

Figure 33. Measures taken to fix the low- risk vulnerabilities.

The above settings were configured in the settings file of the project in order to eliminate the above mentioned vulnerabilities. A scan was run after these settings were configured and 3 of the 4 found vulnerabilities were fixed.

6. VARIOUS REQUIREMENTS OF THE PROJECT:

The game has a number of requirements and the table below is a summary of the requirements that we have met and the below table maps the requirements to the appropriate section in this report. The below security requirements have been met in our project. So, we believe its secure.

Requirements	Justification and mapping
1. A client-server architecture must be used.	Section 3 shows justification
2. Usage of TLS/ SSL for communication through an encrypted channel	Section 5.1 shows justification
3. Once a user logs in, they must be able to: <ul style="list-style-type: none">❖ see the past games statistics of themselves,❖ see game moves of completed games,❖ start a new game, and❖ join an existing game that needs players.	Section 5.5 shows justification
4. Store passwords using a salted hash system	Section 5.4 shows justification
5. Input must be validated from the user and invalid input must be rejected.	Section 5.8 shows justification
6. Sessions are created when a user logs in the game and expires once he is inactive for a specific time or if he logs out and sessions must be managed to prevent attacks.	Section 5.6 shows justification
7. Confidentiality requirement: An audit trail of every game move must be created, game moves must only be available to the ones authenticated. After completing the game, the past game moves must be	Section 5.5 shows justification

available only to the ones authenticated and the ones who have played the game.	
8. Integrity requirement: Only the current player can make a move in the game, he must only make a legal move. The opponent cannot make an illegal move by overwriting the previous player's move and cannot cheat. All game moves are recorded and cannot be altered/overwritten.	Section 5.5 shows justification
9. Availability requirement: The player cannot pause or be idle forever in a game. A player can logout of the game and log back in to continue the game he started(eg: Poor connection), but has to have a time frame to return back, after which the session expires and the game must be forfeited, opponent wins in this situation.	Section 5.6 shows justification
10. At-least one static/dynamic tool must be used to analyze code for any vulnerabilities, vulnerabilities must be handled.	Section 5.9 shows justification
11. Common vulnerabilities in the web application must be avoided.	Section 5.2, 5.4, 5.4, 5.7, 5.8 shows justification

7. REFERENCES:

- | | |
|----------------------|--|
| [GEEKSDJANGO2019] | Django Project MVT Structure
https://www.geeksforgeeks.org/django-project-mvt-structure/ |
| [JAVAPOINT2019] | Django MVT
https://www.javatpoint.com/django-mvt |
| [CHANNELS2019] | Django Channels
https://channels.readthedocs.io/en/latest/ |
| [KEYCDN2018] | Setting up Nginx Proxy Server
https://www.keycdn.com/support/nginx-reverse-proxy |
| [ACUNETIXSSL 2019] | Prodromou, Agathkolis, "TLS Security 3: SSL/TLS Terminology and Basics", 2019
https://www.acunetix.com/blog/articles/tls-ssl-terminology-basics-part-3/ |
| [SQLACUNETIX2019] | What is SQL Injection (SQLi) and How to Prevent It
https://www.acunetix.com/websitesecurity/sql-injection/ |
| [DJANGOSECURITY2019] | Security in Django
https://docs.djangoproject.com/en/3.0/topics/security/ |
| [OWASPSQLCHEATSHEET] | SQL_Injection_Prevention_Cheat_Sheet
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html |
| [SITELOCK2019] | Joyce, Tammany, "The OWASP Top 10: Sensitive Data Exposure", 2019
https://www.sitelock.com/blog/owasp-top-10-sensitive-data-exposure/ |
| [PASSDJANGO2019] | Password Management in Django
https://docs.djangoproject.com/en/3.0/topics/auth/passwords/ |

[OWASPSENSITIVE2019]	Top 10-2017 A3-Sensitive Data Exposure https://www.owasp.org/index.php/Top_10-2017_A3-Sensitive_Data_Exposure
[WHITEHATSEC2019]	Session Management https://www.whitehatsec.com/glossary/content/session-management
[CWE20MITRE2019]	CWE-20: Improper Input Validation https://cwe.mitre.org/data/definitions/20.html
[PYPIBANDIT2019]	Bandit https://pypi.org/project/bandit/
[CWE-10042019]	CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag https://cwe.mitre.org/data/definitions/1004.html
[PORTSWIGGER2019]	SSL cookie without secure flag set https://portswigger.net/kb/issues/00500200_ssl-cookie-without-secure-flag-set
[CODYPARKER2017]	Tutorial: Create a real-time web game with Django Channels and React https://codyparker.com/django-channels-with-react/