Fox and Hounds: A Board Game

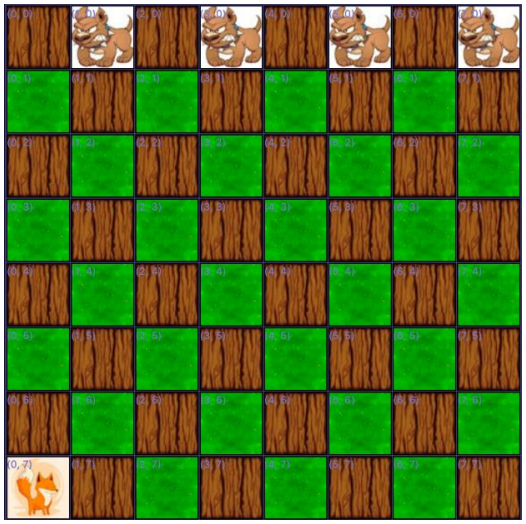
[redacted]

1. Introduction

In this section, we will introduce the game and the rules of the game.

1.1. Fox and Hounds

Fox and Hounds is played on an 8×8 chess/checkerboard and only the dark squares are used (in our case the grass aka green squares). The four hounds are initially placed on the dark squares at one end of the board; the fox is placed on any dark square on the opposite end. The objective of the fox is to cross from one side of the board to the other, arriving at any one of the hounds' original squares. The hounds' objective is to prevent the fox from doing so by trapping it.



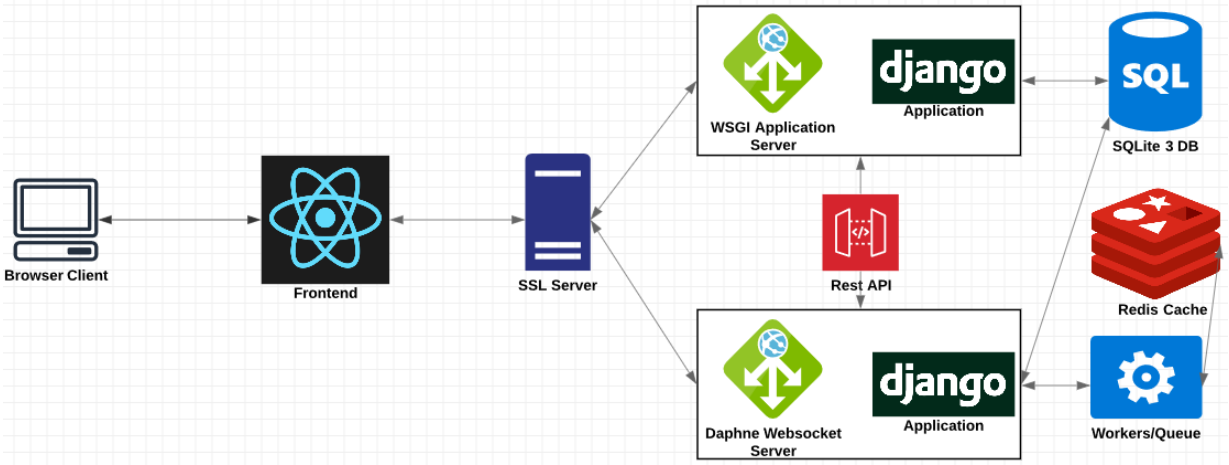
1.2. Game Rules

The hounds can only move diagonally forward one square. The fox can move diagonally forward or backward one square. There is no jumping/eliminating/promoting of pieces in this game. The game starts with the fox moving first. The player controlling the hounds may move only one of them each turn. The fox is trapped when it can no longer move to a vacant square. It is possible for two hounds to trap the fox against an edge of the board (other than their original home-row) or even one corner where a single hound may do the trapping. Should a hound reach the fox's original home row it will be unable to move further.

2. Design and Architecture

In this section, we will discuss the architecture of the application and the different components of the application to under the purpose they serve. To understand how Django and Channels work with ReactJS to build our, we referenced a tutorial by Cody Parker for another game [Parker2017] and blog post by Jacob Moss on Django Channels [Moss2016]. Our implementation for our game, Fox and Hounds, is heavily modified and adds features that are non-existent in these tutorials.

2.1. Architecture

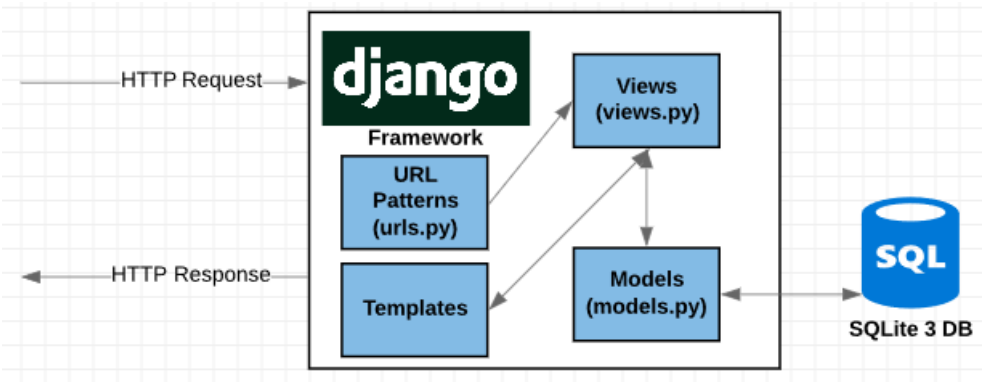


Our game application uses a client server architecture and consists of Django backend, ReactJS frontend, REST API for link between front and backend, SSL server for TLS traffic, SQLite database for long term storage, and Redis cache for temporary storage/WebSocket connections. When a browser client connects to the game application, he interacts with Django HTML templates and ReactJS to play the game. The interactions are then passed onto the Django backend and WebSocket server to be handled. The main Django backend is connected to the SQLite 3 database for long term storage and the WebSocket interactions are cached per connection using Redis.

2.2. Architecture Components

2.2.1. Django Framework

Django is an open-source web application framework written in Python and essentially consists of a collection of modules that aim to make development easier. Django framework is based on the Model-View-Template pattern. In this model, the developer is required to create the Model, View, and Template and then map it to a URL, then Django takes care of the rest.



Model: A model is a class that represents a table or collection in our database, and where every attribute of the class is a field of the table or collection. Models are defined in the models.py file. [TutorialP2020]

View: A view is where the logic of the application is written to process a request and respond to the user. Views are created in the views.py file. [TutorialP2020]

Template: A HTML file that contains both static and dynamic components that are rendered to the user. HTML files are defined it the templates folder. [TutorialP2020]

2.2.2. Django Channels

Django channels are an extension the Django framework that adds a new layer and allows for WebSocket handling and perform background tasks. It separates Django into two process types: one that handles HTTP and WebSocket and another that runs the views. It adds an additional server that needs to be spun up so that it can handle these WebSocket request. This is done though an ASGI server (we use Daphne), a Django worker, and Redis. [Channels2017]

2.2.3. ReactJS

React is a framework for JavaScript and is used to create web applications. ReactJS is used as the frontend and Django as the backend when using together. They need a framework to interact between the two so this is where we used the REST framework.

2.2.4. Django REST Framework

Django REST framework is used to build Web APIs which can be used to transfer data to and from out backend. Below is a sample data from a browsable API in our application.

2.2.5. Django SSL Server

We use the Django SSL Server to allow for HTTPS in our application. It is a lightweight server that is used instead of the regular runserver deployment in Django.

2.2.6. Redis Cache

Django Channels channel layer requires a way to route its requests for WebSocket so we use the popular Redis as its backing store. Redis is an open source in-memory data structure store and allows us to serve requests quickly.

2.3. Code Components

2.3.1. Game

For the game, the main files that are important are consumers.py, models.py, views.py, api_views.py, gameboard.jsx, gamelog.jsx, and gamesquare.jsx. First, the urls.py will route the request to display the appropriate view for the game which are in the views.py. It also helps route the api views for our REST framework so ReactJS components can get data when rendering the game. Gameboard and gamesquare jsx files are used to render the game based on its current state and bind clickable events to each of the squares so the players can play. They host all the logic for game mechanics including prevention of invalid moves, determining the winner, and determining valid moves.

2.3.2. Lobby

For the lobby (homepage), the main files that are import are urls.py, views.py, availablegames.jsx, completedgames.jsx, lobbybase.jsx, and playergames.jsx. First, the urls.py will route the request to display the appropriate view for the home page which are in the views.py. Then availablegames.jsx, completedgames.jsx, lobbybase.jsx, and playergames.jsx grabs the appropriate game lists based on the states and displays them on the page. From here the player can join, view, or create a new game.

2.3.3. Users

There are many HTML template pages and separate models for the users so that we can manage them. We are not going to list all the files under the Users folder but all of them provide login, logout, register, and password reset functionality.

2.4. Included Functionality

2.4.1. User login/logout/register/reset/update

Users can perform login, logout, register, reset, and update functionalities. Users are allowed to read about the game and understand the rules, but they are required to login to do anything else on the website (other than register of course). Once the user has created an account, they are allowed to update their profile which includes their e-mail address, username, and profile picture. If the user forgets their password, they can also opt to get a password reset email and reset their password.

2.4.2. Join/create/view a game

Logged in users can create as many new games as they want, and the game will not start until another logged in user decides to join an existing game. Logged in users are also allowed to view their completed games and the results of the game. Users who aren't part of a game and not allowed to view that game.

2.4.3. Play

Logged in users who created a new game can join it and wait until an opponent decides to join the game and start playing. Players are not allowed to make any moves until both player positions are filled or if the game is over. Players can make a move and exit the game then return before the timeout happens to continue playing. Players need to first select the game piece they want to move and then select a valid free square they want to move that piece to. If the player chooses an invalid move, nothing happens and the game waits until he makes a valid move.

3. Source Code Installation and Setup

In this section, we will go over how to acquire the source code for the game and run it on your own machine, this will allow you to have a running local copy.

3.1. Instructions for macOS or Linux:

1. Acquire a copy of the source code from GitHub: `git clone https://github.com/pmarella2/Fox-and-Hounds.git`
2. Install Homebrew: `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"`
3. Install Node12 and Python3 using Homebrew: `brew install python3` then `brew install node@12`
4. Install virtualenv for python3: `pip3 install virtualenv`
5. Change into the cloned git repo: `cd Fox-and-Hounds`

6. Create a new virtual environment for python3: `virtualenv -p python3 venv`
7. Activate your new virtual environment: `source venv/bin/activate`
8. Install required packages for python: `pip install -r requirements.txt`
9. Install required modules for react: `npm install`
10. Set a secret key for django in your bash profile: `nano ~/.bash_profile` then paste `export SECRET_KEY="your generated secret key"`
    - You can create a secret key here: https://miniwebtool.com/django-secret-key-generator/
11. Migrate Django models into database schema: `python manage.py makemigrations` then `python manage.py migrate`
12. Now open five terminal windows, change directory into the project folder, and activate the python virtual environment
13. Run one of each command in a separate terminal: `redis-server` and `npm run webpack` and `python manage.py runsslserver --certificate (path to your localhost.crt) --key (path to your localhost.key) localhost:8080` and `daphne -b 127.0.0.1 -p 8888 fox_and_hounds.asgi:channel_layer` and `python manage.py runworker`
    - You can create your own self-signed certificates for localhost following this guide: https://deliciousbrains.com/ssl-certificate-authority-for-local-https-development/
14. Now navigate to: https://localhost:8080
15. Play the game and enjoy!

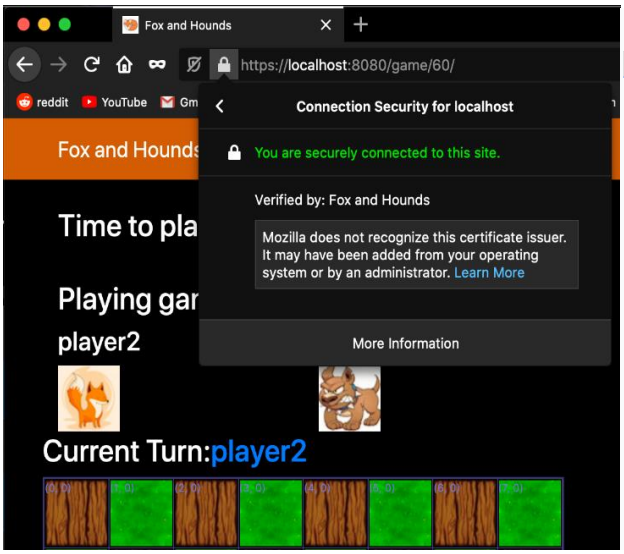
4. Security (Assurance Cases)

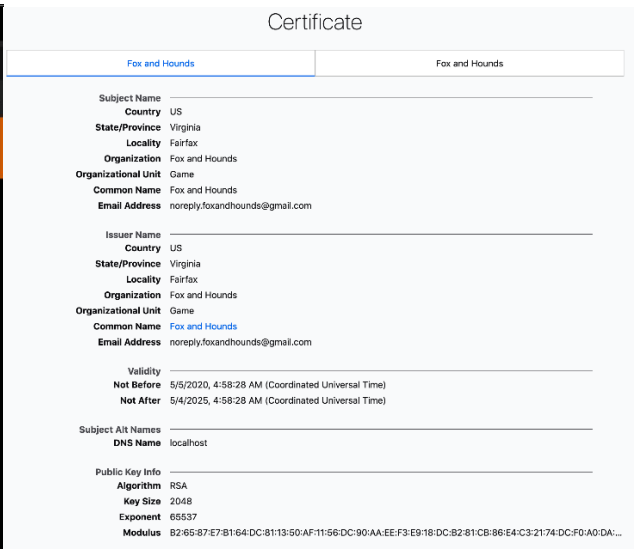
In this section we will discuss the security consideration we took in our web application and provide evidence to our claims. We will primarily focus on the OWASP Top Ten Web Application Security Risks. [OWASP2017]

4.1. Secure communication via HTTPS (TLS/SSL)

"Implementing TLS in a web application can provide a number of security benefits like protection against an attacker from reading the contents of traffic (confidentiality), protection against an attacker modifying traffic (integrity), and allowing the client to verify they are connected to the real server (authentication)." [OWASP2017]

In our web application, implementing TLS will secure the transport protocol so we can transmit the game and user data securely. To configure our web application to use TLS for TCP/IP transport, we generated a self-signed certificate using openssl and RSA public key (with 2048 bits size) cryptography and used those keys with our Django sslserver to run our web application.





The evidence for the implementation of TSL is shown above. We see the connection to our web application is secure and looking at the certificate details, we see that it is issued to the localhost and generated using RSA with 2048 bits key size. For the purposes of this project we have used a self-signed certificate, but we would have used a recognized certificate authority to generate the certificates and use them if this was for production.

4.1.1. HTTP Strict Transport Security

Additionally, OWASP TLS Cheat Sheet [OWASP2020] suggests using HTTP Strict Transport Security (HSTS) to instruct browsers to always request the site over HTTPS. HSTS is an optional feature but we chose to implement based on the security recommendations.

Django provides support for enabling HSTS through their security middleware [Django2020]. We set the HTST seconds to 1 year, which means that the browser will refuse to communicate non-securely with my domain for that long. We chose to include all the subdomains since we serve everything over HTTPS. Below are images for the settings in Django and header info showing the implementation working.

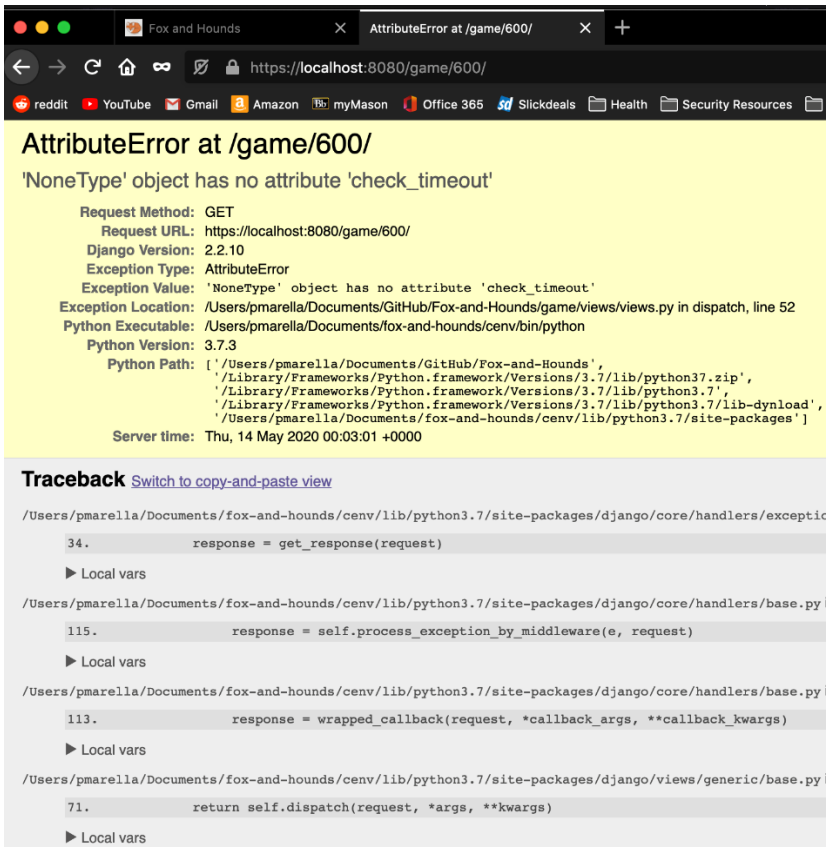
```
CSRF_COOKIE_SECURE = True
SESSION_COOKIE_SECURE = True
SECURE_HSTS_SECONDS = 31556952
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True
SECURE_SSL_REDIRECT = True
SECURE_REFERRER_POLICY = "same-origin"
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True
SECURE_PROXY_SSL_HEADER = ("HTTP_X_FORWARDED_PROTO", "https")
```

```
Content-Length: 4642
Content-Security-Policy: script-src 'self' https://code...ap/4.0.0/css/bootstrap.min.css
Content-Type: text/html; charset=utf-8
Date: Wed, 13 May 2020 23:54:03 GMT
Server: WSGIServer/0.2 CPython/3.7.3
Strict-Transport-Security: max-age=31556952; includeSubDomains; preload
```

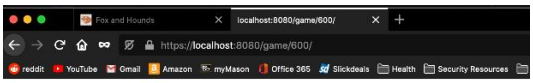
4.2. Security Misconfiguration

Attackers will often attempt to exploit unpatched flaws or access default accounts, unused pages, unprotected files and directories, etc to gain unauthorized access or knowledge of the system. [OWASP2017]

In Django if the debug flag is set to true, we will see a stack trace whenever there is a request that causes an error. We can see an example of that below:



To configure it properly, we set the debug flag to false and also serve a custom 404 and 500 error pages so no additional/unintended information is ever leaked.



500 Server Error

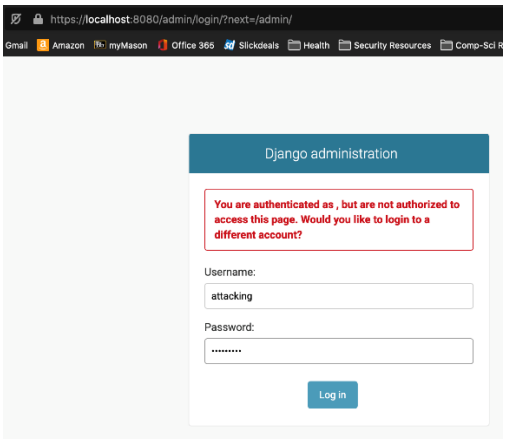
Something is wrong on our end. We will figure it out!

```
DEBUG = False
ALLOWED_HOSTS = ["localhost", "127.0.0.1"]
```

4.2.1. Default admin page configuration

In addition, we also decided to protect access to our default admin page. In Django, by default navigating to domain/admin/ will take you the login page for admin. We changed it so that the actual admin page is served at another custom page and implanted an authentication check before allowing

access to that page. If a user does not have a staff status in their user profile, they will be served with an error when trying to access our custom admin page. On top of this, we also setup a honey pot at the default admin site so we log anyone attempting to log into the admin site in our web application. Below is the evidence for the honey pot at the default admin page.

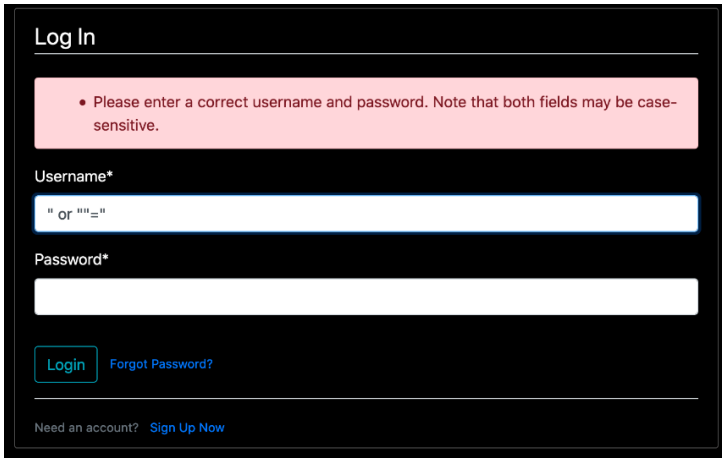


attacking	127.0.0.1	<a href="?session_key=zj541dk1m1vq5gp54da1ik8q5b7cnk3v">zj541dk1m1vq5gp54da1i

4.3. SQL Injection

SQL injection is a type of attack where a malicious user can execute arbitrary SQL code on a database. This can result in records being deleted or data leakage. "SQL Injection flaws are introduced when software developers create dynamic database queries that include user supplied input. To avoid SQL injection flaws is simple. Developers need to either: a) stop writing dynamic queries; and/or b) prevent user supplied input which contains malicious SQL from affecting the logic of the executed query." [OWASP2020]

Django provides built-in SQL injection protection because we use Django's object-relational mapper layer and using query sets though query parameterization. "The query's SQL code is defined separately from the query's parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver." [Django2020]



4.4. Cross Site Scripting (XSS)

Cross Site Scripting (XSS) is a type of attack that allows attackers to inject scripts in the browser. Typically, this script is stored in the database and when another user loads the web application, that script will be loaded from the database and then executed.

Django has built in protections again XSS which is achieved by escaping various characters that are considered dangerous. We can also enable the XSS protection header with block mode to provide additional protection against XSS. This is included in the security middleware provided by Django and it needs to be manually enabled. Below you can see the option being enabled and the header including the X-XSS-Protection Block.

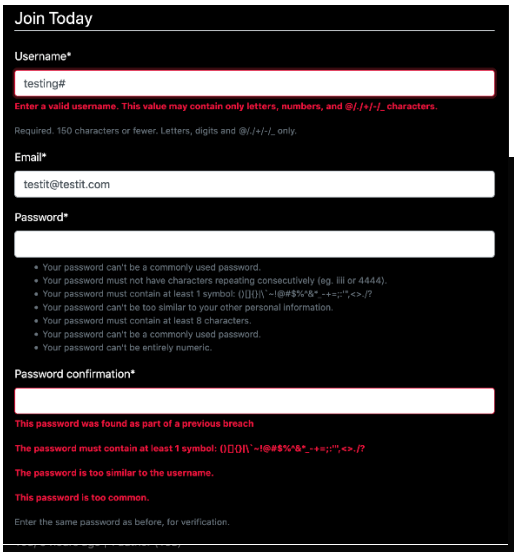
```
SECURE_SSL_REDIRECT = True
SECURE_REFERRER_POLICY = "same-origin"
SECURE_BROWSER_XSS_FILTER = True
SECURE_CONTENT_TYPE_NOSNIFF = True
SECURE_PROXY_SSL_HEADER = ("HTTP_X_FORWARDED_PROTO", "https")
```

```
X-Frame-Options: SAMEORIGIN
Content-Length: 77076
Vary: Cookie, Origin
Strict-Transport-Security: max-age=31556952; includeSubDomains; preload
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
```

Other ways to ensure protection against XSS is to use input validation and configure a Content Security Policy.

4.4.1. Input Validation

We have added input validation to the username and password fields of the forms used in registration and logging in pages of our web application. This input validation is provided by default with Djangovalidators but we also chose to include a couple of custom validators.




```
AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "pwned_passwords_django.validators.PwnedPasswordsValidator",
        "OPTIONS": {"error_message": ("This password was found as part of a previous breach")},
    },
    {
        "NAME": "fox_and_hounds.validators.SameCharacterRepeatingValidator",
    },
    {
        "NAME": "fox_and_hounds.validators.RequiredSymbolValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
        "OPTIONS": {"min_length": 8},
    },
    {"NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",},
    {"NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",},
]
```

```
class SameCharacterRepeatingValidator:
    def __init__(self, min_length=3):
        self.min_length = min_length

    def validate(self, password, user=None):
        for char in password:
            if password.count(char) >= self.min_length:
                char_check = char * self.min_length
                if char_check in password:
                    raise ValidationError(
                        _(
                            "The password has too many characters repeating consecutively (eg. iiii or 4444)"
                        ),
                        code="password_repeating",
                    )

    def get_help_text(self):
        return _(
            "Your password must not have characters repeating consecutively (eg. iiii or 4444)."
        )
```

```
You, 9 hours ago | 1 author (You)
class RequiredSymbolValidator:
    def validate(self, password, user=None):
        if not re.findall(r"[()[\]{}|\\`~!@#$%^&*_\-+=;:'\",<>./?]", password):
            raise ValidationError(
                _(
                    "The password must contain at least 1 symbol: "
                    + "()[]{}|\`~!@#$%^&*_-+=;:'\",<>./?"
                ),
                code="password_no_symbol",
            )

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

As we can see above if the user input characters that are not part of the allowed characters, then the user is not allowed to register. The list of validators that we use in Django are pwned_passwords (looks for passwords that are part of previous breach lists), requiredsymbolvalidator (uses regex to make sure the user is using atleast one symbol in their password), samecharacterrepeatingvalidator (looks for repeated characters in a password), userattributesimilarityvalidator (looks for elements in the password that are also part of the username or email id), minimumlenghtvalidator (looks for atleast the specified number of characters in the password field), commonpasswordvalidators (checks the input password against a list of commonly used passwords), and numericpasswordvalidator (checks if the entire password is only numers).

4.4.2. Content Security Policy

Another method of protecting against XSS is to configure a CSP (content security policy). CSP allows us to configure a list of valid sources that our web application can grab executable scripts or content from. This reduces the risk vectors for us by blocking potentially malicious sources loading scripts onto our web application. Below you can see the CSP for our web application.

```
CSP_DEFAULT_SRC = (
    "'self'",
    "https://o374711.ingest.sentry.io/",
    "ws://localhost:8888/"
)
CSP_STYLE_SRC = (
    "'self'",
    "https://stackpath.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css",
)
CSP_SCRIPT_SRC = (
    "'self'",
    "https://code.jquery.com/jquery-3.5.0.slim.min.js",
    "https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js",
    "https://stackpath.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js",
)
CSP_IMG_SRC = ("'self'",)
CSP_FONT_SRC = ("'self'",)
```

4.5. Cross Site Request Forgery (CSRF)

“Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data since the attacker has no way to see the response to the forged request.” [OWASP2020] It is dangerous because if an attacker submits a malicious request on behalf of an authenticated user, it can lead to unintended actions. In our game, it could mean someone making a move that the user actually didn’t want to which leads to cheating.

Django has built-in protection against this type of attacks. “CSRF protection works by checking for a secret in each POST request. This ensures that a malicious user cannot simply “replay” a form POST to your website and have another logged in user unwittingly submit that form. The malicious user would have to know the secret, which is user specific (using a cookie).” [Django2020] We enable this protection just by including the middleware that as part of the settings. Below is the evidence of the CSRF protection in action.

```
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
```

csrftoken	C5wmnoLQhNmsw5YDgKO3oQimZ49naRHI1UuBipgoaDiEf1LJwQZA0...	localhost	/	Wed, 12 May 2021...	73
sessionid	h5zgj9nh68axy8r02fcda5b4jbwlksb8	localhost	/	Session	41

4.6. Session Management

"A web session is a sequence of network HTTP request and response transactions associated to the same user. Modern and complex web applications require the retaining of information or status about each user for the duration of multiple requests. Therefore, sessions provide the ability to establish variables – such as access rights and localization settings – which will apply to each and every interaction a user has with the web application for the duration of the session." [OWASP2020] Once an authenticated session has been established, the session ID (or token) is temporarily equivalent to the strongest authentication method used by the application. Not handling session management properly can lead to the user's session data being compromised or the attacker using the session ID to authenticate as the user.

Django had built in functionality to handle session management and we also configured some optional settings to secure the session management conducted in our web application. As part of our session management policy, we have ensured that the user's session expires after five minutes of inactivity and after the user closes the browser. This forces the user to have a newly generated session id upon relogging in or visiting our web application. We have also ensured the session id cookie is secure. Below is the evidence of our configuration.

```
"django_session_timeout.middleware.SessionTimeoutMiddleware",
"django.middleware.security.SecurityMiddleware",
"django.contrib.sessions.middleware.SessionMiddleware",
```

```
SESSION_EXPIRE_SECONDS = 300
SESSION_EXPIRE_AFTER_LAST_ACTIVITY = True
SESSION_TIMEOUT_REDIRECT = "home"
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
```

```
CSRF_COOKIE_SECURE = True
SESSION_COOKIE_SECURE = True
```


csrftoken	C5wmnoLQhNmsw5YDgKO3oQimZ49naRHI1UuBipgoaDiEf1LJwQZA0...	localhost	/	Wed, 12 May 2021...	73
sessionid	h5zgj9nh68axy8r02fcda5b4jbwlksb8	localhost	/	Session	41

We have also implemented the timeout for the user's being inactive in a game session using the following enforcement code which gets executed on every load into the game. If the user is inactive on his turn for more than five minutes, then the other player is automatically declared the winner and the game will be ended.

```
def check_timeout(self):      You, 19 hours ago • [Completed] Timeout and Completed Gamelist
    if (self.opponent is not None) and (self.game_over is False):
        last_log = GameLog.objects.filter(game=self).reverse()[0]
        #timenow = datetime.now()
        timenow = datetime.now(timezone.utc)
        lastactivity = last_log.modified
        diff = (timenow - lastactivity).total_seconds()

        if (diff > 300):
            if self.current_turn == self.creator:
                self.mark_complete(winner=self.opponent)
                self.update_game_status(game_over=True)
                self.add_log("{0} won the game, {1} forfeited!".format(self.opponent, self.creator))
                self.send_game_update()
            elif self.current_turn == self.opponent:
                self.mark_complete(winner=self.opponent)
                self.update_game_status(game_over=True)
                self.add_log("{1} won the game, {0} forfeited!".format(self.opponent, self.creator))
                self.send_game_update()
```

4.7. Sensitive Data Exposure

Sensitive data exposure occurs when the application is implementing the necessary encryptions in place for sensitive data handled in an application.

The most sensitive data we handle in our application is our user's password. To protect against sensitive data exposure, we use the PBKDF2 – a key derivation function on our passwords before we store them in our database. PBKDF2 uses a pseudorandom function to input the password along with the sale value and repeats the process many times to produce a derived key. The derived key function is DK = PBKDF2 (PRF, Password, Salt, iterations, dkLen), the larger the iterations, the more work is required to derive the key and makes it harder for attackers to crack it. Below you will find out implementation of this using the built-in hashers Django providers with a custom modification to the iteration count used by PBKDF2 in our application.

```
PASSWORD_HASHERS = [
    "fox_and_hounds.hashers.MyPBKDF2PasswordHasher",
    "django.contrib.auth.hashers.PBKDF2PasswordHasher",
    "django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher",
    "django.contrib.auth.hashers.Argon2PasswordHasher",
    "django.contrib.auth.hashers.BCryptSHA256PasswordHasher",
]
```

```python
from django.contrib.auth.hashers import PBKDF2PasswordHasher


You, 6 hours ago | 1 author (You)
class MyPBKDF2PasswordHasher(PBKDF2PasswordHasher):
    # iterations = PBKDF2PasswordHasher.iterations * 5
    iterations = 100000      You, 6 hours ago • [Update] Hashs
```

Our hashed passwords stored in the database:

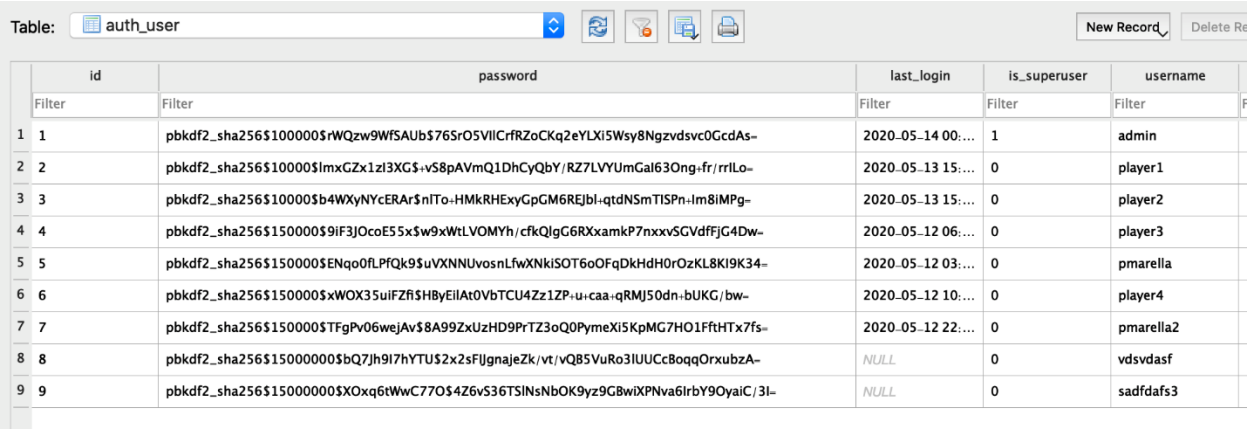


Table: auth_user

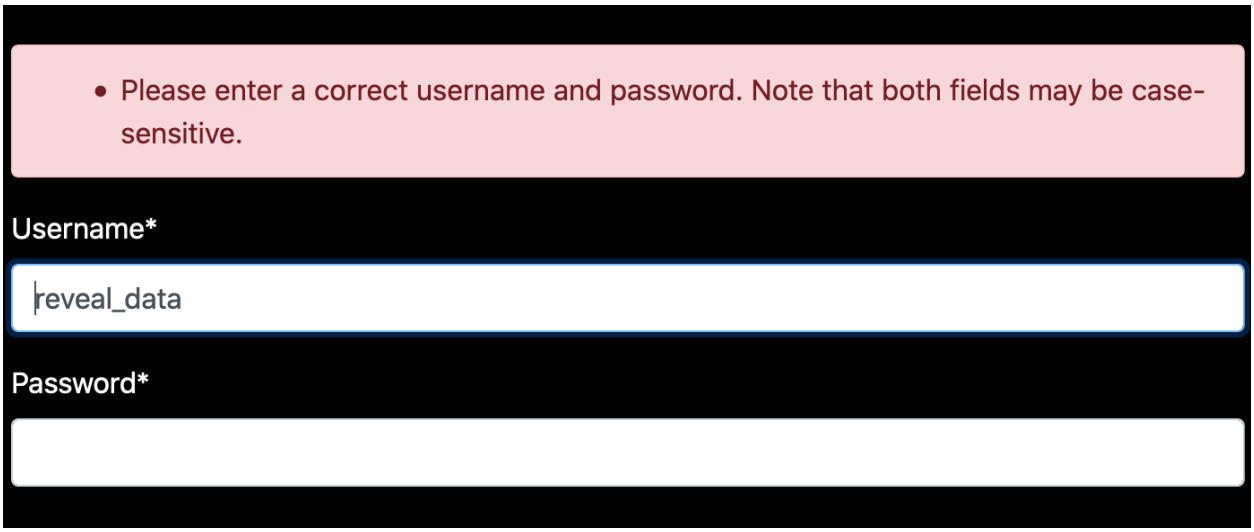
New Record | Delete Re

	id	password	last_login	is_superuser	username
	Filter	Filter	Filter	Filter	Filter
1	1	pbkdf2_sha256$100000$rWQzw9WfSAUb$76SrO5VIlCrfRZoCKq2eYLXi5Wsy8Ngzvdsvc0GcdAs=	2020-05-14 00:...	1	admin
2	2	pbkdf2_sha256$10000$lmxGZx1zI3XG$+vS8pAVmQ1DhCyQbY/RZ7LVYUmGaI63Ong+fr/rrILo=	2020-05-13 15:...	0	player1
3	3	pbkdf2_sha256$10000$b4WXyNYcERAr$nITo+HMkRHExyGpGM6REJbl+qtdNSmTISPn+Im8iMPg=	2020-05-13 15:...	0	player2
4	4	pbkdf2_sha256$150000$9iF3JOcoE55x$w9xWtLVOMYh/cfkQlgG6RXxamkP7nxxvSGVdfFjG4Dw=	2020-05-12 06:...	0	player3
5	5	pbkdf2_sha256$150000$ENqo0fLPfQk9$uVXNNUvosnLfwXNkiSOT6oOFqDkHdH0rOzKL8KI9K34=	2020-05-12 03:...	0	pmarella
6	6	pbkdf2_sha256$150000$xWOX35uiFZfi$HByEilAt0VbTCU4Zz1ZP+u+caa+qRMJ50dn+bUKG/bw=	2020-05-12 10:...	0	player4
7	7	pbkdf2_sha256$150000$TFgPv06wejAv$8A99ZxUzHD9PrTZ3oQ0PymeXi5KpMG7HO1FftHTx7fs=	2020-05-12 22:...	0	pmarella2
8	8	pbkdf2_sha256$15000000$bQ7Jh9I7hYTU$2x2sFIJgnajeZk/vt/vQB5VuRo3lUUCcBoqqOrxubzA=	NULL	0	vdsvdasf
9	9	pbkdf2_sha256$15000000$XOxq6tWwC77O$4Z6vS36TSlNsNbOK9yz9GBwiXPNva6IrbY9OyaiC/3l=	NULL	0	sadfdafs3

4.7.1. Information Exposure during Failed Logins

When a user attempts into the application, we must not specify which login parameter the user has input incorrectly. Revealing the field that is incorrect will allow the user to focus on bruteforcing/crack the credentials.

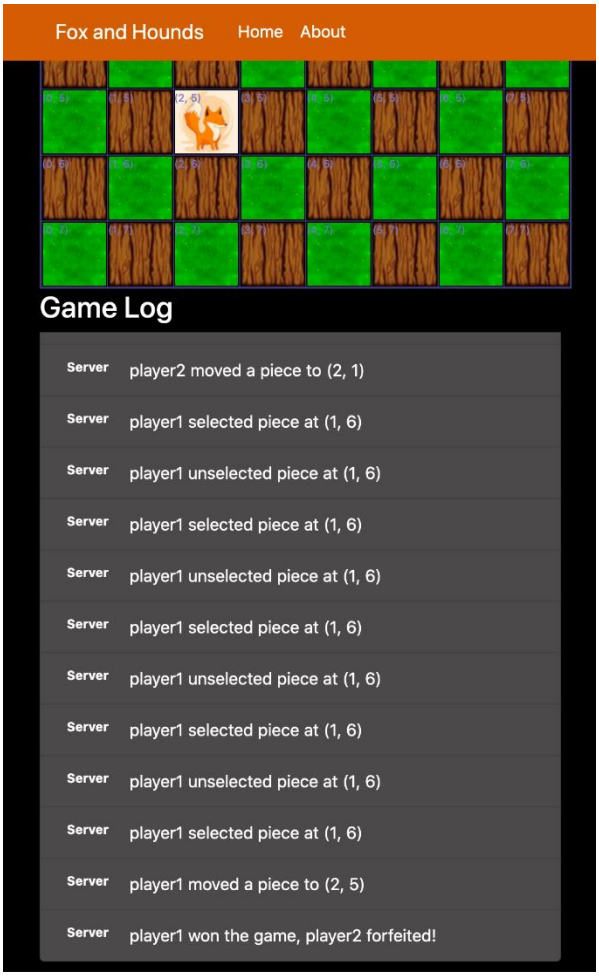
In our implementation, we provide a generic error so that the user is known being exposed to sensitive data leakage.



4.8. Logging and Monitoring

“Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data.” [OWASP2017] Audit log also allows us in our application to have a record of all player’s activity and we can use it help in response to accusing of cheating or etc..,

In our implementation, we log every move a play makes in the game and store in our database for a long-term storage. Players are not allowed access to modify this log and only admins have access to the master log which they may only access by logging into the admin portal via multi-factor authentication. Below you can see sample of logs that we store in the database and the log the players get to see in their game. We also provide a view of the admin login page that requires MFA to access the site and view the master logs.



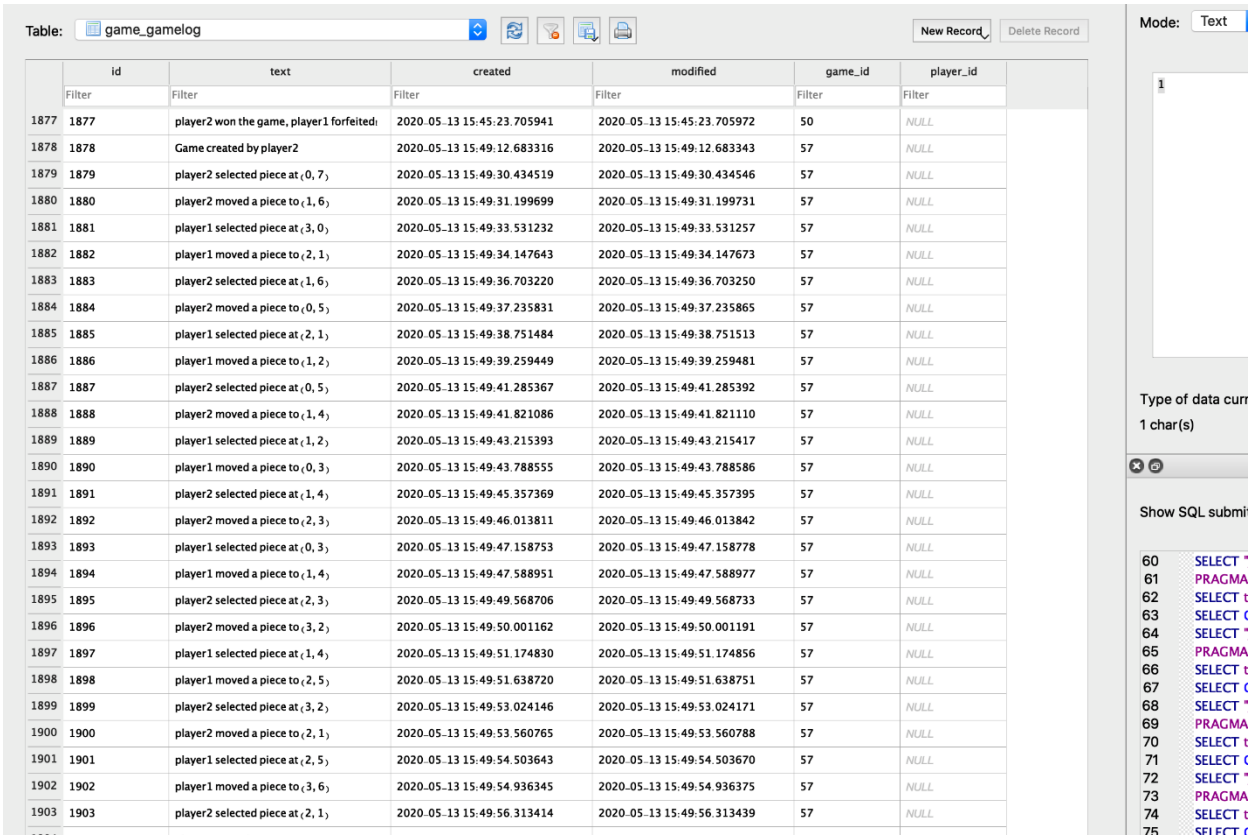
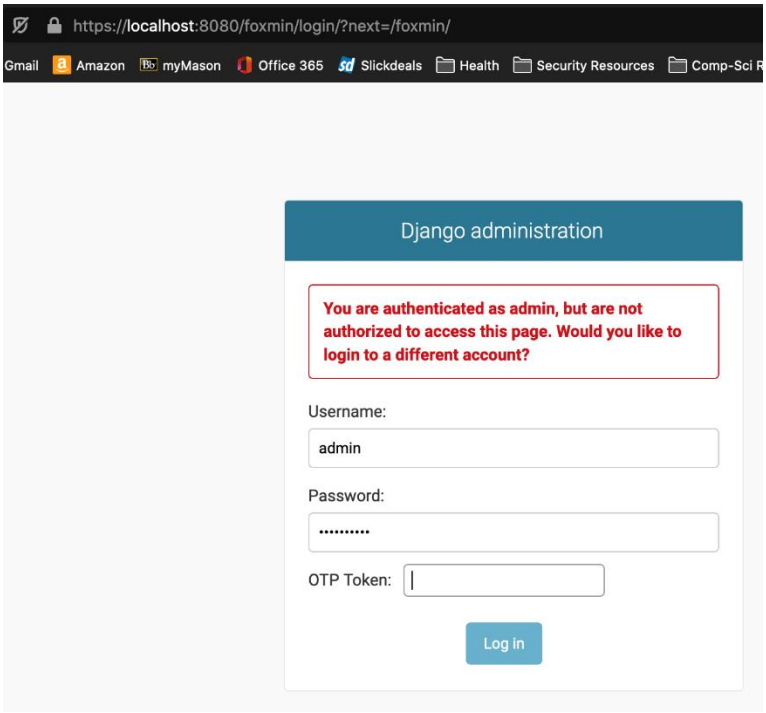
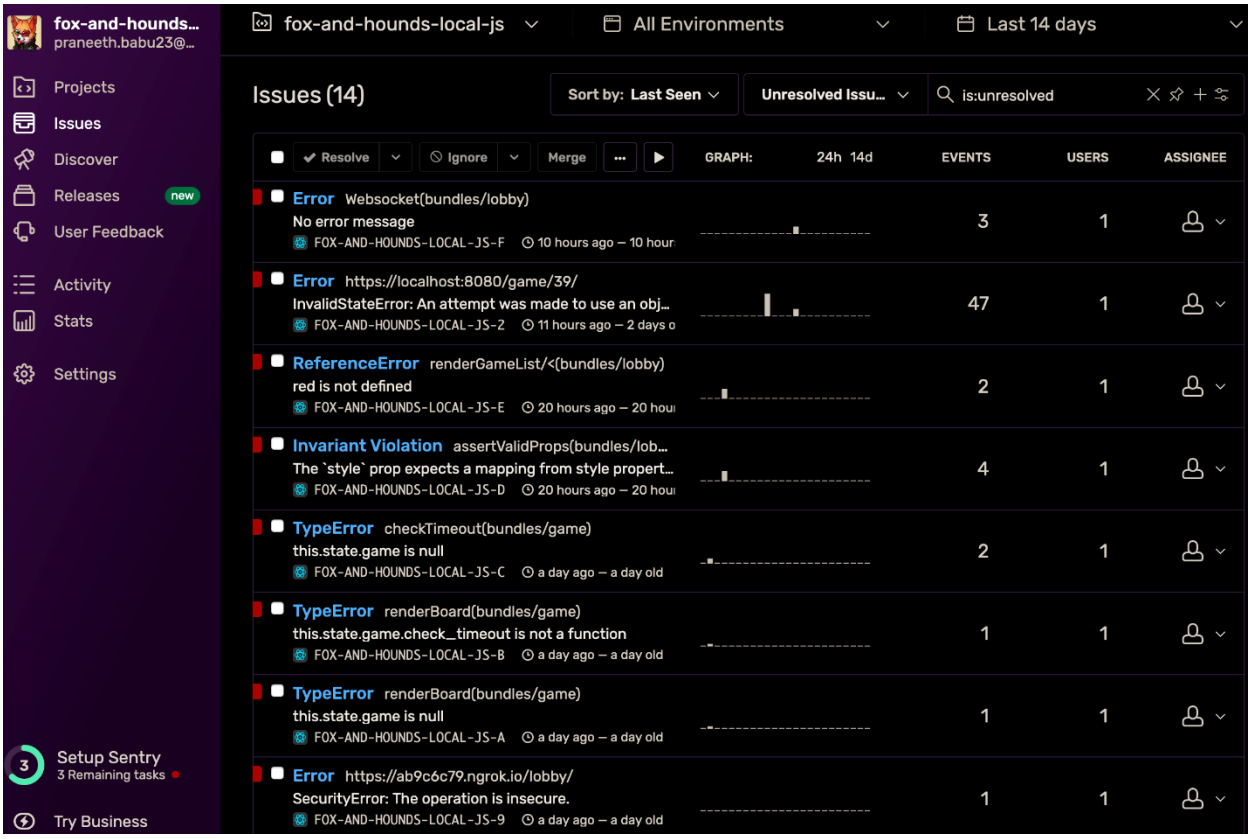


Table: game_gamelog
New Record
Delete Record
Mode: Text
id
text
created
modified
game_id
player_id
Filter
1877 1877 player2 won the game, player1 forfeited 2020-05-13 15:45:23.705941 2020-05-13 15:45:23.705972 50 NULL
1878 1878 Game created by player2 2020-05-13 15:49:12.683316 2020-05-13 15:49:12.683343 57 NULL
1879 1879 player2 selected piece at (0, 7) 2020-05-13 15:49:30.434519 2020-05-13 15:49:30.434546 57 NULL
1880 1880 player2 moved a piece to (1, 6) 2020-05-13 15:49:31.199699 2020-05-13 15:49:31.199731 57 NULL
1881 1881 player1 selected piece at (3, 0) 2020-05-13 15:49:33.531232 2020-05-13 15:49:33.531257 57 NULL
1882 1882 player1 moved a piece to (2, 1) 2020-05-13 15:49:34.147643 2020-05-13 15:49:34.147673 57 NULL
1883 1883 player2 selected piece at (1, 6) 2020-05-13 15:49:36.703220 2020-05-13 15:49:36.703250 57 NULL
1884 1884 player2 moved a piece to (0, 5) 2020-05-13 15:49:37.235831 2020-05-13 15:49:37.235865 57 NULL
1885 1885 player1 selected piece at (2, 1) 2020-05-13 15:49:38.751484 2020-05-13 15:49:38.751513 57 NULL
1886 1886 player1 moved a piece to (1, 2) 2020-05-13 15:49:39.259449 2020-05-13 15:49:39.259481 57 NULL
1887 1887 player2 selected piece at (0, 5) 2020-05-13 15:49:41.285367 2020-05-13 15:49:41.285392 57 NULL
1888 1888 player2 moved a piece to (1, 4) 2020-05-13 15:49:41.821086 2020-05-13 15:49:41.821110 57 NULL
1889 1889 player1 selected piece at (1, 2) 2020-05-13 15:49:43.215393 2020-05-13 15:49:43.215417 57 NULL
1890 1890 player1 moved a piece to (0, 3) 2020-05-13 15:49:43.788555 2020-05-13 15:49:43.788586 57 NULL
1891 1891 player2 selected piece at (1, 4) 2020-05-13 15:49:45.357369 2020-05-13 15:49:45.357395 57 NULL
1892 1892 player2 moved a piece to (2, 3) 2020-05-13 15:49:46.013811 2020-05-13 15:49:46.013842 57 NULL
1893 1893 player1 selected piece at (0, 3) 2020-05-13 15:49:47.158753 2020-05-13 15:49:47.158778 57 NULL
1894 1894 player1 moved a piece to (1, 4) 2020-05-13 15:49:47.588951 2020-05-13 15:49:47.588977 57 NULL
1895 1895 player2 selected piece at (2, 3) 2020-05-13 15:49:49.568706 2020-05-13 15:49:49.568733 57 NULL
1896 1896 player2 moved a piece to (3, 2) 2020-05-13 15:49:50.001162 2020-05-13 15:49:50.001191 57 NULL
1897 1897 player1 selected piece at (1, 4) 2020-05-13 15:49:51.174830 2020-05-13 15:49:51.174856 57 NULL
1898 1898 player1 moved a piece to (2, 5) 2020-05-13 15:49:51.638720 2020-05-13 15:49:51.638751 57 NULL
1899 1899 player2 selected piece at (3, 2) 2020-05-13 15:49:53.024146 2020-05-13 15:49:53.024171 57 NULL
1900 1900 player2 moved a piece to (2, 1) 2020-05-13 15:49:53.560765 2020-05-13 15:49:53.560788 57 NULL
1901 1901 player1 selected piece at (2, 5) 2020-05-13 15:49:54.503643 2020-05-13 15:49:54.503670 57 NULL
1902 1902 player1 moved a piece to (3, 6) 2020-05-13 15:49:54.936345 2020-05-13 15:49:54.936375 57 NULL
1903 1903 player2 selected piece at (2, 1) 2020-05-13 15:49:56.313414 2020-05-13 15:49:56.313439 57 NULL
Type of data curr
1 char(s)
Show SQL submi
60 SELECT
61 PRAGMA
62 SELECT t
63 SELECT
64 SELECT
65 PRAGMA
66 SELECT t
67 SELECT
68 SELECT
69 PRAGMA
70 SELECT t
71 SELECT
72 SELECT
73 PRAGMA
74 SELECT t


https://localhost:8080/foxmin/login/?next=/foxmin/
Gmail
Amazon
myMason
Office 365
Slickdeals
Health
Security Resources
Comp-Sci R
Django administration
You are authenticated as admin, but are not authorized to access this page. Would you like to login to a different account?
Username:
admin
Password:
OTP Token:
Log in

4.8.1. Error Monitoring

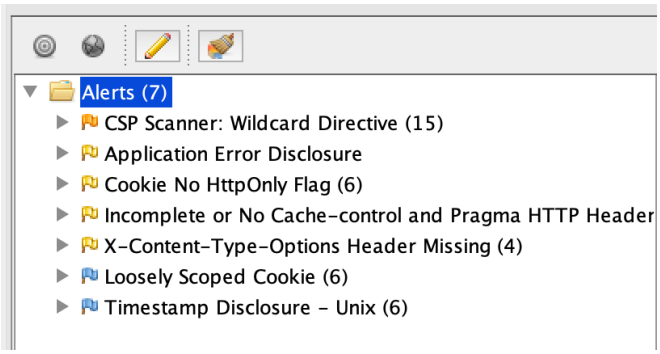
We also use sentry.io application monitoring and error tracking software to monitor both our Django backend and ReactJS frontend. If there happens to be runtime errors, we can see them in the console and appropriately handle them.



5. Vulnerability Management

5.1. OWASP ZAP

We used OWASP ZAP Web Application vulnerability scanner to find potential vulnerabilities in our web application. We performed a scan on our application using the attack mode and it found seven vulnerabilities (1 medium, 4 low, and 2 informational).



1. **Found:** CSP Scanner: Wildcard Directive – The following directive will either allow wildcard sources, are not defined, or overly defined.
   Handled: Added a CSP Ancestor setting to the Django application.

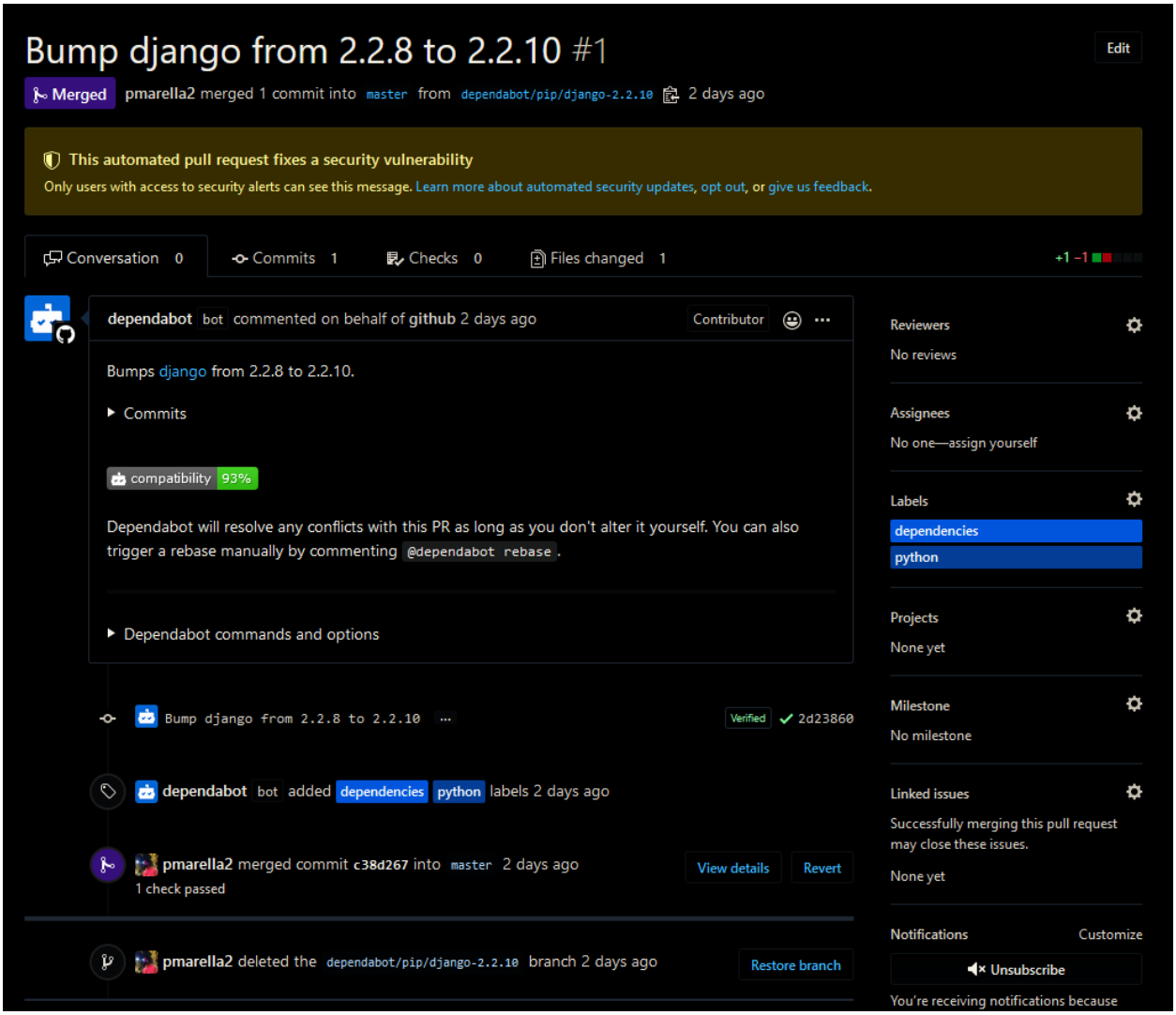
```
)
CSP_IMG_SRC = ("'self'",)
CSP_FONT_SRC = ("'self'",)
CSP_FRAME_ANCESTORS = ("'self'",)      You, a fe

LOGGING = {
```

2. **Found:** Application Error Disclosure – This page contains an error/warning message that may disclose sensitive information like the location of a file.
   Handled: This is a false positive because the error page being served is a custom error page that we created that contains no sensitive information.
3. **Found:** Cookie No HttpOnly Flag – A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by javascript.
   Handled: Accepting this as an acceptable risk. "Designating the CSRF cookie as `HttpOnly` doesn't offer any practical protection because CSRF is only to protect against cross-domain attacks. If an attacker can read the cookie via JavaScript, they're already on the same domain as far as the browser knows, so they can do anything they like anyway. (XSS is a much bigger hole than CSRF.)" [Djnago2020]
4. **Found:** Incomplete or No Cache-control – Cache-control has not been set properly to allow browsers to cache content.
   Handled: Accepting this as an acceptable risk.
5. **Found:** X-Content-Type-Option Header Missing
   Handled: Accepting this as an acceptable risk.
6. **Found:** Loosely Scoped Cookie
   Handled: This is a false positive. Due to the cookie coming from localhost, this was triggered, and it is the CSRF cookie.
7. **Found:** Timestamp Disclosure – Unix: A timestamp was disclosed by the application/web server
   Handled: This is a false positive because the timestamp resolves to 1970.

5.2. Dependabot

Our application depends on quite several external libraries and package. It is hard to manually keep up to date with all the vulnerabilities that these might have. So, we enabled dependabot on our GitHub project. Dependabot will scope our dependencies and if they are vulnerable, it will create a pull request to upgrade that dependency to a version that has the vulnerability fixed – this also depends on the compatibility of the new version with our current project.



References

[TutorialP2020] Django Tutorial - Tutorialspoint. (2020). Retrieved 13 May 2020, from https://www.tutorialspoint.com/django/index.htm

[Channels2017] In Short — Channels 1.1.8 documentation. (2017). Retrieved 13 May 2020, from https://channels.readthedocs.io/en/1.x/inshort.html#what-is-channels

[Moss2016] Finally, Real-Time Django Is Here: Get Started with Django Channels. (2017). Retrieved 13 May 2020, from https://blog.heroku.com/in_deep_with_django_channels_the_future_of_real_time_apps_in_django

[Parker2017] Tutorial: Create a real-time web game with Django Channels and React - CodyParker.com. (2017). Retrieved 13 May 2020, from https://codyparker.com/django-channels-with-react/

[OWASP2017] OWASP Top Ten Web Application Security Risks | OWASP. (2017). Retrieved 13 May 2020, from https://owasp.org/www-project-top-ten/

[OWASP2020] Transport Layer Protection · OWASP Cheat Sheet Series. (2020). Retrieved 13 May 2020, from https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

[Django2017] Middleware | Django documentation | Django. (2020). Retrieved 13 May 2020, from https://docs.djangoproject.com/en/2.2/ref/middleware/#http-strict-transport-security