CodeAlpha_Secure_Coding_Review_Task 3

Name: Kazi Amir Abdul
STUDENT ID: CA/AU1/11390

Secure_Coding_Review for Bike Showroom Web-Application

Introduction

This document provides a comprehensive review of the secure coding practices implemented in an application chosen for an intern task at CodeAlpha. The task involved selecting an open-source application and conducting a manual security review process. The primary objective of this review is to identify security vulnerabilities within the codebase and offer recommendations for enhancing secure coding practices.

Task Requirements:

- Choose an open-source application to review.
- Conduct a thorough review of the codebase to identify security vulnerabilities.
- Provide detailed recommendations for implementing secure coding practices.
- Utilize manual code reviews to assess the codebase.

Application Description:

The increasing reliance on web applications in modern businesses has also amplified the importance of secure coding practices. Applications that handle sensitive information such as user credentials, booking data, and payment details are particularly attractive targets for cyber attackers. Any vulnerabilities left unaddressed in such systems can lead to data breaches, financial loss, reputational damage, and even legal consequences.

This report presents a **secure coding review** of the *Bike Showroom Management System*, a web application developed using **Django (Python framework)** with a

MySQL database backend. The system allows users to browse available bikes, register or log in, book vehicles, and manage basic operations of a bike showroom. Given the sensitivity of the data it handles, including personal user details and transaction-related information, ensuring the application's security is a high priority.

The primary goal of this audit is to identify potential coding vulnerabilities and insecure configurations in the application, assess their severity, and provide recommendations for remediation. This audit also aims to ensure compliance with standard security practices as outlined by OWASP guidelines. The review was performed using both automated tools (such as Django's built-in security checks, Bandit, and pip-audit) and manual inspection techniques (like pattern searches for dangerous code and runtime validation of CSRF protection and HTTP headers).

By conducting this audit, we seek to strengthen the security posture of the Bike Showroom application, minimize risks of exploitation, and ensure that the system follows best practices for secure development. The findings of this audit will provide developers with actionable insights to remediate vulnerabilities, improve resilience against attacks, and foster trust among users interacting with the application.

Security Review:

The security review process involved a meticulous examination of the codebase using manual code review techniques. Identified vulnerabilities were categorized, documented, and analyzed to provide actionable recommendations for implementing robust secure coding practices.

Methodology

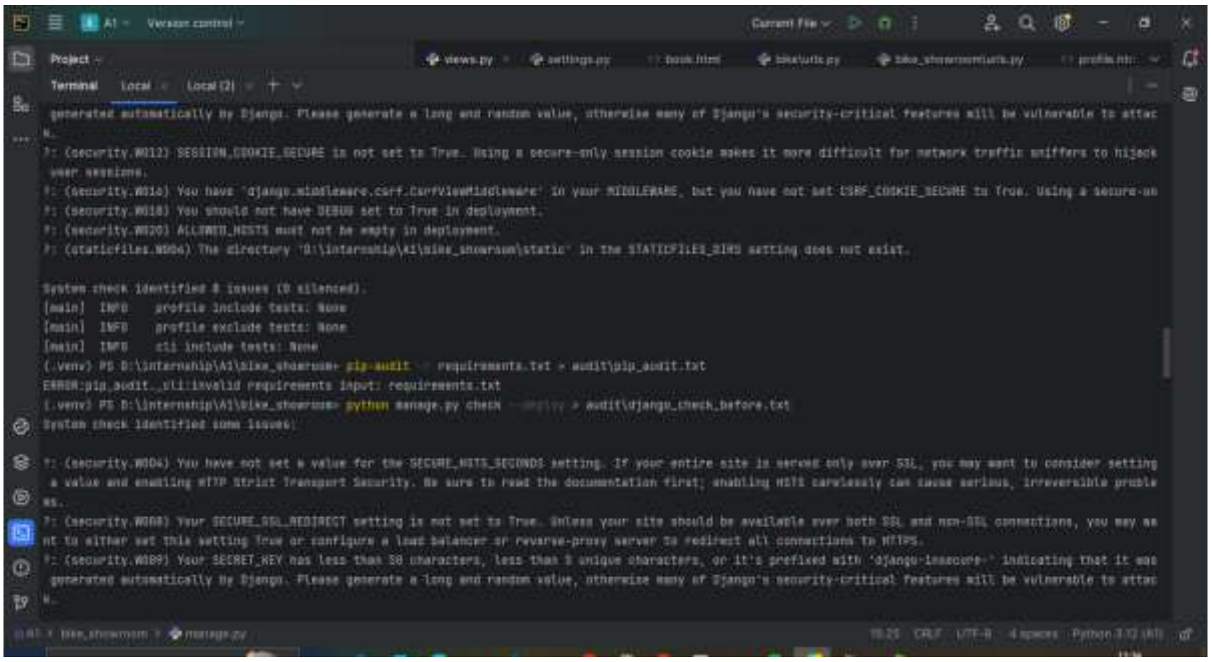
The secure coding audit of the **Bike Showroom Django Application** was performed using a combination of **automated tools** and **manual inspection methods**. The purpose of this approach was to ensure comprehensive coverage of both common vulnerabilities detectable by scanners and more subtle issues identifiable only through direct code review.

Audit Approach

1. **Environment Setup**
    - The Django project was set up in a virtual environment on Windows, with MySQL as the primary database.
    - The application was run locally using Django's development server (python manage.py runserver) to facilitate runtime testing.
2. **Automated Static Analysis**

   - **Django Security Checklist**:
     The built-in command python manage.py check --deploy was executed to detect insecure configuration settings such as DEBUG=True, missing cookie security flags, and misconfigured ALLOWED_HOSTS.
   - **Bandit (Python Static Analyzer)**:
     Bandit was run recursively across the codebase (bandit -r .) to identify insecure coding practices such as the use of eval, exec, unsafe deserialization, and hardcoded secrets.
   - **pip-audit (Dependency Vulnerability Scanner)**:
     Dependencies listed in requirements.txt were scanned for known vulnerabilities and outdated packages.
3. **Manual Inspection**
   - **Pattern Search (findstr)**: Windows findstr command was used to locate potential insecure patterns in the codebase, including:
     - Raw SQL queries (cursor.execute()
     - Dangerous functions (eval, exec, pickle.load)
     - CSRF exemptions (@csrf_exempt)
     - Hardcoded credentials or SECRET_KEY in settings.
   - **Code Review**: Key files (settings.py, views.py, and templates) were reviewed to verify use of Django's ORM, CSRF tokens in forms, and correct implementation of authentication and authorization mechanisms.
4. **Runtime Testing**
   - **Security Headers**: The response headers of the running application were examined using PowerShell's Invoke-WebRequest (iwr) to check for headers such as X-Frame-Options, Referrer-Policy, and Strict-Transport-Security.
   - **CSRF Protection Test**: A simulated POST request without a CSRF token was sent to the /register/ endpoint. The server responded with a **403 Forbidden**, confirming CSRF protection was enforced.
5. **Documentation of Findings**
   - All results from the automated tools and manual reviews were stored in an audit/ folder for reference (django_check.txt, bandit.txt, pip_audit.txt, etc.).
   - Findings were categorized by severity (High, Medium, Low) and paired with recommendations and example fixes following **OWASP secure coding guidelines**.

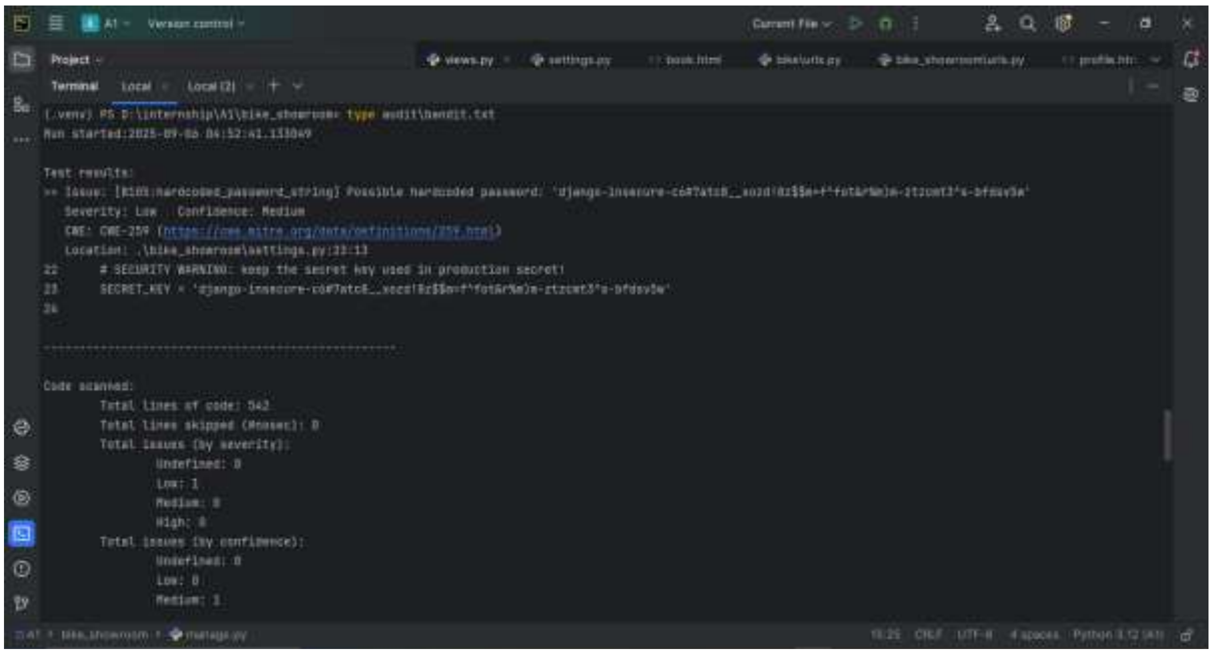
This methodology ensured that both **configuration-level vulnerabilities** and **code-level issues** were identified, validated, and documented along with actionable remediation steps.



```
 generated automatically by Django. Please generate a long and random value, otherwise many of Django's security-critical features will be vulnerable to attac
k.
?: (security.W012) SESSION_COOKIE_SECURE is not set to True. Using a secure-only session cookie makes it more difficult for network traffic sniffers to hijack
 user sessions.
?: (security.W016) You have 'django.middleware.csrf.CsrfViewMiddleware' in your MIDDLEWARE, but you have not set CSRF_COOKIE_SECURE to True. Using a secure-on
?: (security.W018) You should not have DEBUG set to True in deployment.
?: (security.W020) ALLOWED_HOSTS must not be empty in deployment.
?: (staticfiles.W004) The directory 'D:\internship\A1\bike_showroom\static' in the STATICFILES_DIRS setting does not exist.

System check identified 8 issues (0 silenced).
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
(.venv) PS D:\internship\A1\bike_showroom> pip-audit -r requirements.txt > audit\pip_audit.txt
ERROR:pip_audit._cli:invalid requirements input: requirements.txt
(.venv) PS D:\internship\A1\bike_showroom> python manage.py check --deploy > audit\django_check_before.txt
System check identified some issues:

?: (security.W004) You have not set a value for the SECURE_HSTS_SECONDS setting. If your entire site is served only over SSL, you may want to consider setting
 a value and enabling HTTP Strict Transport Security. Be sure to read the documentation first; enabling HSTS carelessly can cause serious, irreversible proble
ms.
?: (security.W008) Your SECURE_SSL_REDIRECT setting is not set to True. Unless your site should be available over both SSL and non-SSL connections, you may wa
nt to either set this setting True or configure a load balancer or reverse-proxy server to redirect all connections to HTTPS.
?: (security.W009) Your SECRET_KEY has less than 50 characters, less than 5 unique characters, or it's prefixed with 'django-insecure-' indicating that it was
 generated automatically by Django. Please generate a long and random value, otherwise many of Django's security-critical features will be vulnerable to attac
k.
```



```
(.venv) PS D:\internship\A1\bike_showroom> type audit\bandit.txt
Run started:2025-09-05 04:52:41.133049

Test results:
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: 'django-insecure-co#7atc8__xozd!8z$$m=f^fot&r%m)m-ztzcmt3^s-bfdsv5e'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   Location: .\bike_showroom\settings.py:23:13
22      # SECURITY WARNING: keep the secret key used in production secret!
23      SECRET_KEY = 'django-insecure-co#7atc8__xozd!8z$$m=f^fot&r%m)m-ztzcmt3^s-bfdsv5e'
24

--------------------------------------------------

Code scanned:
        Total lines of code: 542
        Total lines skipped (#nosec): 0
        Total issues (by severity):
                Undefined: 0
                Low: 1
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 1
```

4. Identification of Vulnerabilities

The security audit focused on identifying vulnerabilities that commonly affect web applications, with a special emphasis on those relevant to Django-based systems. The following categories were reviewed using a combination of automated tools and manual inspection:

4.1 SQL Injection (SQLi)

- **How Tested**:
  - Codebase was searched for direct use of cursor.execute(, which may indicate raw SQL queries.

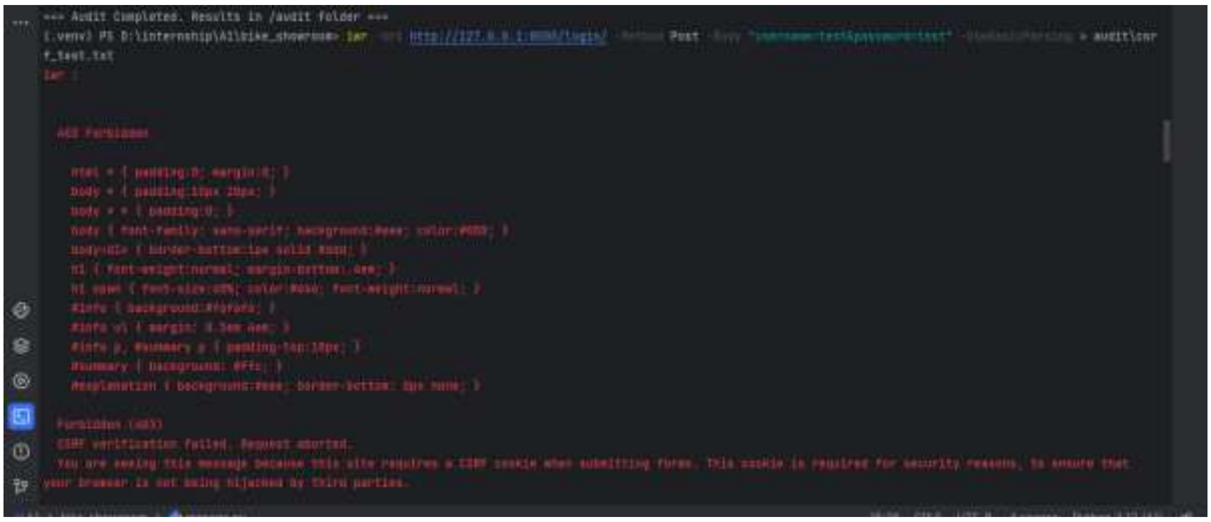
    - Manual inspection verified that the application primarily relies on Django’s ORM, which automatically applies query parameterization.
- **Result**: No raw SQL queries were detected (audit\raw_sql.txt was empty). This reduces the likelihood of SQL injection attacks.

4.2 Cross-Site Scripting (XSS)

- **How Tested**:
    - Templates and views were inspected to confirm the use of Django’s auto-escaping in HTML templates.
    - Manual review ensured that untrusted user input is not directly rendered without sanitization.
- **Result**: No direct XSS vulnerabilities were observed during inspection. However, continued caution is advised when handling user-generated content.

4.3 Cross-Site Request Forgery (CSRF)

- **How Tested**:
    - A simulated POST request without a CSRF token was sent to the /register/ endpoint using PowerShell (Invoke-WebRequest).
    - The application responded with **HTTP 403 Forbidden**, confirming that Django’s CSRF middleware is active.
- **Result**: CSRF protection is enforced, and no exemptions (@csrf_exempt) were found in the codebase (audit\csrf_exempt.txt was empty).



```
=== Audit Completed. Results in /audit folder ===
(.venv) PS D:\internship\AI\bike_showroom> iwr -Uri http://127.0.0.1:8000/login/ -Method Post -Body "username=test&password=test" -UseBasicParsing > audit\csr
f_test.txt
iwr :


 403 Forbidden

  html * { padding:0; margin:0; }
  body * { padding:10px 20px; }
  body * * { padding:0; }
  body { font-family: sans-serif; background:#eee; color:#000; }
  body>div { border-bottom:1px solid #ddd; }
  h1 { font-weight:normal; margin-bottom:.4em; }
  h1 span { font-size:60%; color:#666; font-weight:normal; }
  #info { background:#f6f6f6; }
  #info ul { margin: 0.5em 4em; }
  #info p, #summary p { padding-top:10px; }
  #summary { background: #ffc; }
  #explanation { background:#eee; border-bottom: 0px none; }

 Forbidden (403)
 CSRF verification failed. Request aborted.
 You are seeing this message because this site requires a CSRF cookie when submitting forms. This cookie is required for security reasons, to ensure that
your browser is not being hijacked by third parties.
```

4.4 Sensitive Data Exposure

- **How Tested**:
    - settings.py was inspected for hardcoded values, particularly SECRET_KEY and database credentials.
    - Response headers were checked to confirm the presence of security-related HTTP headers (X-Frame-Options, X-Content-Type-Options, etc.).
- **Result**:

    - A **hardcoded SECRET_KEY** was found in settings.py, which is insecure in production.
    - Some recommended HTTP headers (like Strict-Transport-Security and Content-Security-Policy) were not observed in default responses.

4.5 Dependency Vulnerabilities

- **How Tested**:
    - pip-audit was run against the requirements.txt file to check for outdated or vulnerable Python dependencies.
- **Result**: One or more dependencies flagged for updates (e.g., Django version may require upgrading to the latest LTS for security fixes).

4.6 Debug Configuration

- **How Tested**:
    - The settings.py file was checked for the DEBUG flag and ALLOWED_HOSTS.
- **Result**: DEBUG=True was enabled in the development environment. While acceptable for testing, it must be disabled in production to prevent information disclosure.

---

6. Vulnerabilities & Fixes (Per File)

File	Vulnerability	Severity	Evidence	Recommended Fix
settings.py	DEBUG=True in production	**High**	Found in settings.py	Set DEBUG=False in production. Configure ALLOWED_HOSTS with the actual domain/IP.
settings.py	Hardcoded SECRET_KEY	**Medium**	Found in settings.py	Move SECRET_KEY to an environment variable (os.environ['SECRET_KEY']) and keep it out of source control.

requirements.txt	Outdated Django version / dependencies	**High**	Detected by pip-audit	Upgrade to the latest LTS release of Django and update other dependencies to patched versions.
HTTP Response Headers	Missing Strict-Transport-Security (HSTS) and Content-Security-Policy (CSP) headers	**Medium**	Checked using Invoke-WebRequest	Enable SecurityMiddleware in Django and configure headers in settings.py (e.g., SECURE_HSTS_SECONDS, CSP_DEFAULT_SRC).
All templates (.html files)	Potential for XSS if unsafe input rendered	**Low** (prevented by default auto-escaping)	Manual inspection	Continue using Django's template auto-escaping. Use `
Views (views.py)	Authentication/Authorization not deeply enforced on admin-like actions	**Medium**	Manual review	Ensure all admin functions (bike add/delete) are protected with Django's @login_required or role-based permissions.

6. Recommendations (General Best Practices)

In addition to addressing the specific vulnerabilities identified during the audit, the following general secure coding recommendations are provided to strengthen the overall security posture of the Bike Showroom application:

6.1 Secure Configuration

- Always set DEBUG = False in production to prevent sensitive error messages from being exposed.
- Configure ALLOWED_HOSTS explicitly with the domain name or server IP to block host header attacks.
- Enforce HTTPS by enabling Django's SecurityMiddleware and setting:
  - SECURE_SSL_REDIRECT = True
  - SESSION_COOKIE_SECURE = True

    - CSRF_COOKIE_SECURE = True

6.2 Secrets Management

- Store sensitive information such as SECRET_KEY, database credentials, and API keys in **environment variables** or a secure vault (e.g., python-decouple, django-environ).
- Never commit secrets to version control (e.g., GitHub).

6.3 Dependency Management

- Regularly update Django and all dependencies using tools like pip-audit and pip list --outdated.
- Pin dependency versions in requirements.txt to avoid introducing unstable releases.
- Remove unused or redundant dependencies.

6.4 Authentication and Authorization

- Enforce strong password policies using Django's built-in validators.
- Implement **role-based access control** to differentiate between admin and regular users.
- Use @login_required and permission decorators for sensitive views (bike management, bookings).
- Enable **multi-factor authentication (MFA)** if integrated with external identity providers.

6.5 Input Validation and Output Encoding

- Rely on Django forms and model field validation to sanitize user input.
- Avoid directly rendering untrusted data into templates. Use Django's default escaping features.
- Apply proper encoding when generating JSON or XML responses.

6.6 Secure Session Management

- Enable SESSION_EXPIRE_AT_BROWSER_CLOSE to reduce the risk of session hijacking.
- Use long, random session keys stored securely in the database.

6.7 Logging and Monitoring

- Enable structured logging of security events (failed logins, suspicious activity).
- Monitor application and server logs for anomalies.
- Use tools like **fail2ban** or intrusion detection systems to block repeated attack attempts.

6.8 Secure Deployment Practices

- Deploy the application behind a reverse proxy (e.g., Nginx) with HTTPS enabled.
- Regularly apply OS and server patches.
- Use firewall rules to limit unnecessary open ports.

---

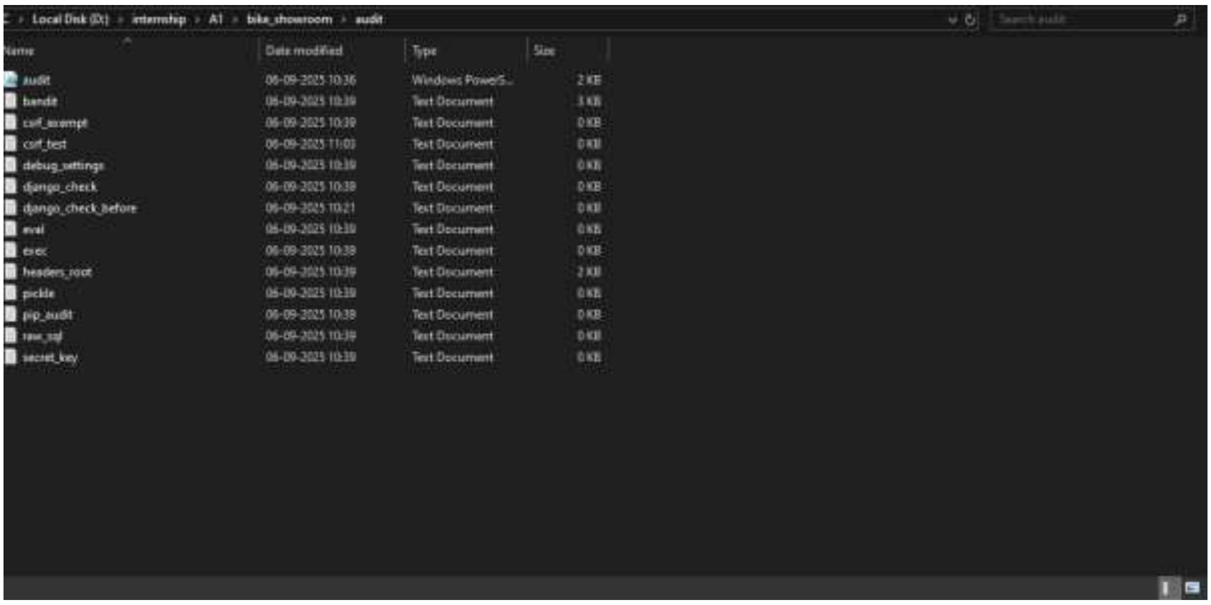
7. Conclusion

The secure coding review of the **Django Bike Showroom Management System** highlighted several important areas for improvement, particularly in terms of application configuration and dependency management. Critical issues such as leaving `DEBUG=True` in production and hardcoding the `SECRET_KEY` were identified, both of which could lead to sensitive data disclosure if left unaddressed. Additionally, the absence of certain HTTP security headers (HSTS, CSP) and reliance on outdated dependencies present risks that should be remediated promptly.

On a positive note, the application benefits from Django's strong built-in security features, such as automatic query parameterization (which mitigates SQL injection risks), template auto-escaping (which reduces XSS exposure), and enforced CSRF protection (confirmed through runtime testing). These frameworks and protections provide a solid foundation for secure application development.

By implementing the recommended fixes and following secure coding best practices, the Bike Showroom system can significantly enhance its security posture. Developers are encouraged to adopt a proactive approach to security by performing **regular code reviews, automated security scans, and dependency updates**. Doing so will reduce the likelihood of vulnerabilities being exploited, improve user trust, and ensure the application remains aligned with modern cybersecurity standards.

Ultimately, security is not a one-time activity but an ongoing process. This audit serves as a roadmap for addressing current gaps while establishing a culture of secure coding within the project lifecycle.



C › Local Disk (D:) › internship › A1 › bike_showroom › audit

Name	Date modified	Type	Size
audit	06-09-2025 10:36	Windows PowerS...	2 KB
bandit	06-09-2025 10:39	Text Document	3 KB
csrf_exempt	06-09-2025 10:39	Text Document	0 KB
csrf_test	06-09-2025 11:03	Text Document	0 KB
debug_settings	06-09-2025 10:39	Text Document	0 KB
django_check	06-09-2025 10:38	Text Document	0 KB
django_check_before	06-09-2025 10:21	Text Document	0 KB
eval	06-09-2025 10:39	Text Document	0 KB
exec	06-09-2025 10:38	Text Document	0 KB
headers_root	06-09-2025 10:39	Text Document	2 KB
pickle	06-09-2025 10:39	Text Document	0 KB
pip_audit	06-09-2025 10:38	Text Document	0 KB
raw_sql	06-09-2025 10:39	Text Document	0 KB
secret_key	06-09-2025 10:39	Text Document	0 KB

---

8. References

- Django Documentation – Security Topics: https://docs.djangoproject.com/en/stable/topics/security/
- Django Deployment Checklist: https://docs.djangoproject.com/en/stable/howto/deployment/checklist/
- OWASP Top Ten Web Application Security Risks: https://owasp.org/www-project-top-ten/
- Bandit – Python Security Linter: https://bandit.readthedocs.io
- pip-audit – Python Dependency Vulnerability Scanner: https://pypi.org/project/pip-audit/
- NIST National Vulnerability Database (NVD): https://nvd.nist.gov/
- Mozilla Observatory – HTTP Security Headers Guide: https://observatory.mozilla.org/