Tampere University

COMP.SEC.300 – Secure Programming

To-do App

Exercise work

Group members

Haiqa Noor

Sang Nguyen

Project link: https://github.com/ndsang001/Secure_To_Do_App/

General Description

Web-based applications serve as fundamental components of current software systems because they provide remote and collaborative data access for users. Online applications have grown more complex, yet security remains as the central issue. The Secure To-Do App provides a full-stack web-based task management system which embeds secure programming practices as its foundation from initial development stages.

The application allows users to register, authenticate, and manage personal tasks within a protected environment. The system uses Django for its backend framework and combines it with frontend functions of React and TypeScript. Secure RESTful APIs from the backend function using JSON Web Tokens (JWT) together with Django's authentication components allow the frontend to consume them through a modern interface with its responsive design.

The main goal of this project includes reliable functionality together with comprehensive security features that implement current industry procedures. To ensure online safety the system implements password governance requirements along with validation checks for input data and authorization permissions throughout front and backend areas and regulates session management protocols and maintains secure user data procedures. The system implements additional security measures that consist of security header enforcement via middleware and rate limiting functionality for thwarting brute force attacks and structural logging systems for monitoring suspicious operations.

Users find the interface easy to understand due to its simple design. Users have access to viewing and creating or modifying and removing personal to-do tasks after successful login. The application maintains robust user experience alongside strong security through CSRF protection and prevention of XSS attacks.

The project illustrates that standard web applications can achieve effective security through principled protection measures incorporated from development initiation and before system deployment by reducing vulnerabilities in the system.

Program Structure

The Secure To-Do App consists of two main components: the backend, built with Django and Django REST Framework (DRF), and the frontend, developed using React with TypeScript. Each component is modularized for clarity, maintainability, and secure development practices.

Backend Modules (Django + Django REST Framework)

1. Core project configuration: app/

   This module manages the application's core configuration. It provides:

   - settings.py: Centralized configuration with secure defaults (rate limiting, CORS, JWT, CSRF). Database configuration with PostgreSQL.
   - middleware/: Contains custom logic:
   - jwt_cookie_auth.py: Parses JWTs from cookies, injects into request.user.
   - security_headers.py: Adds headers like CSP, HSTS, Referrer-Policy for defense-in-depth.
   - urls.py: Root routing, should include app-specific URLs (include(auth_app.urls)).
   - wsgi.py and asgi.py: Ready for production deployment with WSGI/ASGI servers.

2. Application logic: auth_app/

   This module handles user authentication and todo operations. It includes:

   - models.py: Defines the Todo model.
   - views.py: Handles registration, login, logout, JWT token refresh, and CRUD operations for todos.
   - serializers.py: Marshals data between model and API response.

- validators.py: Custom password complexity enforcement.
- urls.py: Scoped route definitions (like /auth/login/, /auth/todos/).

Frontend Modules (React + TypeScript)

1. API abstraction layer: api/

   This helps to handle communication with the Django backend.

   - axios.ts: centralized Axios instance with “baseURL” and “withCredentials” set.
   - authentication.ts: Handles login, registration, logout, CSRF, and token refresh logic.
   - todo.ts: CRUD operations for todo items (fetch, add, toggle, delete).

2. Route-Based UI component: pages/

   In this module (folder), the path was divided into authentication/ and routes/.

   In the authentication/ folder, Signin.tsx and Signup.tsx provide login and registration forms. Meanwhile, Logout.tsx clears authentication cookies and updates state.

   In the routes/ folder, Dashboard.tsx is the main application screen displaying todos. ProtectedRoute.tsx implements route guarding, ensuring only authenticated users can access certain pages. ErrorPage.tsx serves as a fallback UI when an error occurs, and GlobalErrorBoundary.tsx wraps the application to catch and display unexpected rendering errors.

3. State management: store/

   The application’s state is managed using Zustand, with separate stores in the store/ folder.

   The useAuthenticationStore.ts file manages authentication state, including user login, registration, logout, token checking, and CSRF token fetching. It also handles loading and error states.

The useTodoStore.ts file manages the todos list, including fetching todos, adding new tasks, toggling completion, clearing completed todos, and filtering by status. Zustand's minimal and efficient architecture allows clean separation of logic and avoids prop drilling across components.

Security implementation following OWASP TOP 10 2021

1. A01: Broken Access Control
    1.1. Middleware to prevent clickjacking in backend
    We utilized the Django's built-in middleware as "django.middleware.clickjacking.XFrameOptionsMiddleware" to handle UI redress/clickjacking. This middleware was added and could be found in "/backend/app/settings.py".
    1.2. Middleware to restrict access in backend
    "Permissions-Policy" in "SecurityHeadersMiddleware" helps to prevent abuse of device features like camera/mic.
    1.3. Middleware to enforce CSRF in backend
    "django.middleware.csrf.CsrfViewMiddleware" was added and its decorator "@csrf_protect" was applied to register_user, login_user, and logout_view. This CSRF protection ensures that only trusted and intentional POST, PUT, PATCH, or DELETE requests are accepted by the server, especially when they are modifying sensitive data.
    1.4. @permission_classes([IsAuthenticated])
    The @permission_classes([IsAuthenticated]) from Django REST Framework (DRF) was utilized to restrict access to routes like /todos/, /todos/toggle/, and /todos/clear_completed/. These routes are wrapped with this decorator to ensure the logic that checks the request's user authentication status.
    1.5. ProtectedRoute component in frontend
    Our React frontend uses a ProtectedRoute component that checks auth state (authenticated) before rendering protected pages like the Dashboard.
2. A02:2021 – Cryptographic Failures
    2.1. Password validation in backend

The application is using Django's password validation method to ensures password complexity and secure hashing using PBKDF2 + salt.

2.2. Enforce HTTPS in backend

The cookie secure setting current is set to be "False" in views' responses. It must be changed into "True" in production to ensure the cookie information are transmitted securely over HTTPS. Additionally, the "SECURE_SSL_REDIRECT" setting will automatically be "True" in production to enforce HTTPS.

2.3. Using ".env" in backend

The ".env" is used to store secret key and other sensitive settings.

2.4. cookie-based JWTs in frontend

Our frontend uses "withCredentials: true" with axios to rely on cookie-based JWTs and avoid localStorage/sessionStorage risks.

2.5. Source map setting in frontend

The source map setting was disabled in vite.config.ts, which prevents exposing source code as well as improves security and performance.

3. A03: Injection

3.1. Custom security header middleware to prevent code injection or XSS in backend

By adding a custom SecurityHeadersMiddleware, "Content-Security-Policy" was configured here to block inline JS and unauthorized sources. This helps to mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.

3.2. Using Django Object-Relational Mapper (ORM) in backend

This feature from Django framework was utilized to allow application to interact with database using Python code instead of writing raw SQL queries. For example, using "User.objects.get(...)" and "Todo.objects.filter(...)" would allow to query the database for specific data using Python, which protects against SQL injection.

3.3. User input validation in frontend

User inputs (like text, email, password) are validated using Yup schemas and "react-hook-form".

4. A04:2021 – Insecure Design

   The current design of application enforces secure defaults like rate-limiting and CSRF protection in the backend. Furthermore, the “access” and “refresh” tokens are distinguished with “httponly” cookies. In frontend, the interface was applied with “Snackbar” for feedback and does not rely on alert() or global errors. The error page is also wrapped in GlobalErrorBoundary component to ensure proper error handling.

5. A05:2021 – Security Misconfiguration

   5.1. Throttling configuration and rate limit in backend

   The Django REST Framework throttling and Django_ratelimit were implemented to prevent abuse and add a protective layer by default to all views. With throttling configuration, all DRF-protected views would be applied with rate limit, which only allows a limited api call for user per day. Furthermore, the “@ratelimit” decorator in non-DRF views like “login_user” and “register_user” helps stop brute-force attacks, spam, and automated abuse. In this application, the throttling limit for user is 500/day, while it is maximum 5 attempts per minute for login action and 10 attempts per hour for register action. The detail implementation of these configuration can be found in “/backend/app/settings.py” and “/backend/auth_app/views.py”

   5.2. Custom security header middleware to prevent misconfigurations in backend

   By adding a custom SecurityHeadersMiddleware, “Strict-Transport-Security”, “Permissions-Policy”, “X-Content-Type-Options”, Spectre headers were configured here. In that, “Strict-Transport-Security” would enforce HTTPS connection in production. “Permissions-Policy” helps to restrict access to browser features (such as camera, mic, etc.), while “X-Content-Type-Options” mitigates MIME-type confusion attacks. Additionally, Spectre headers including “Cross-Origin-Opener-Policy”, “Cross-Origin-Embedder-Policy”, and “Cross-Origin-Resource-Policy” supports to mitigate cross-origin attacks, enhance isolation for shared resources, and prevent cross-origin resource sharing when unintended. These settings can be found in “/backend/app/middleware/ security_headers.py”.

5.3. Using Axios in frontend

The app uses axios.ts with centralized config (withCredentials, baseURL from .env). This ensures secure cookie-based authentication, code readability and debugging.

6. A06:2021 – Vulnerable and Outdated Components

For error disclosure handling in frontend, the GlobalErrorBoundary component was added to wrap “App.tsx” component to avoid crashing pages showing raw errors. This implementation can be found in “/frontend/src/pages/routes/GlobalErrorBoundary.tsx” and ““‘/frontend/src/main.tsx”.

7. A07:2021 – Identification and Authentication Failures

7.1. JWT authentication,session handling, and refresh token method in backend

To improve the security for user authentication, we decided to use a custom token refresh view “CustomTokenRefreshView” for handling “/auth/refresh/” endpoint instead of relying on default “TokenRefreshView” from simplejwt framework. The custom view reads the refresh token from cookies instead of relying on the default Authorization header, which can catch and handle InvalidToken exceptions. Additionally, the token rotation setting was enabled to update the new refresh token as the used token will be added into token “blacklist”. The credentials and tokens now are properly handled, preventing user enumeration risk. The detail implementation of CustomTokenRefreshView can be found in /backend/auth_app/views.py

7.2. Throttling configuration and rate limit in backend

This implementation (metioned in A05:2021 – Security Misconfiguration section above) helps defend against brute force attacks (like guessing credentials) and credential stuffing (like leaked credentials) .

7.3. HTTPS connection in backend

The header middleware was added including “Strict-Transport-Security” to enforce HTTPS, which protects authentication tokens and credentials from being exposed over unencrypted channels.

7.4. Secure password handling

The project's implementation ensures the password being stored using strong hashing algorithms as PBKDF2 with salt. Besides, the custom password validator (CustomComplexityValidator) enforces four password complexity rules. The implementation can be checked in ""/backend/auth_app/validators.py" and "/backend/auth_app/views.py"

7.5. JWT authentication and session handling

8. A08:2021 – Software and Data Integrity Failures

In frontend, the React app uses TypeScript with yup for runtime validation.

Manual testing

Testing Type	Description	Result / Action Taken
SQL Injection (SQLi)	Attempted SQL injection via task input fields and URL params.	No injection possible due to Django ORM and input validation.
Cross-Site Request Forgery (CSRF)	Submitted unauthorized requests from third-party origins.	Django's CSRF middleware blocked requests lacking CSRF token.
Sensitive Data Exposure	Triggered intentional errors and reviewed server/client responses.	Replaced stack traces with generic error messages;
Dependency Audit	Ran npm audit and checked Python dependencies.	Minor warnings resolved; no critical vulnerabilities found.

Jenkins pipeline testing

The application was tested through Jenkins pipeline following the same structure and technique used in Secure programming course's exercises. Overall, the Jenkins pipeline automates multiple stages of security testing and deployment, which is illustrated in the Figure 1 below. Key stages include:

- Code checkout from GitHub
- Static Application Security Testing (SAST) using Semgrep
- Software Composition Analysis (SCA) using Snyk and CycloneDX
- File system and container image scanning using Trivy
- Docker image build and deployment

- Dynamic Application Security Testing (DAST) using OWASP ZAP
- Saving Artifact

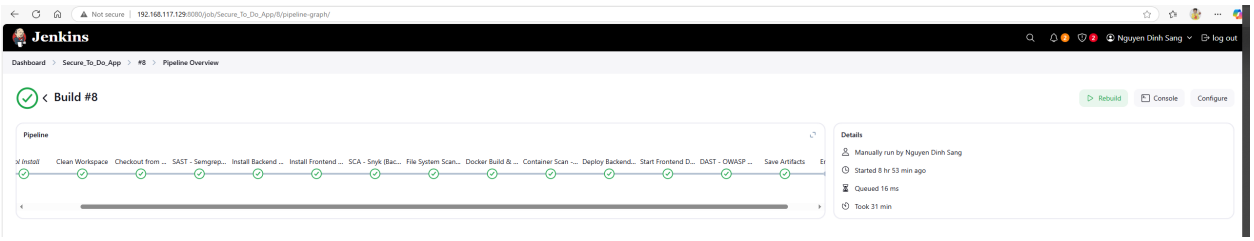


Figure 1: Jenkins pipeline overview

1. Static Application Security Testing (SAST)

   Semgrep was used to scan both backend (Django) and frontend (React). Results:

   - Frontend: no issues found
   - Backend: warning regarding missing DRF throttling (fixed by adding DEFAULT_THROTTLE_CLASSES and RATES in settings.py)

   Mitigation and possible solution for improving security: the found issue related to missing throttling configuration are vulnerable to Denial of Service (DoS) attacks through abuse of endpoints. Therefore, we applied solutions as adding Django REST Framework throttling settings (DEFAULT_THROTTLE_CLASSES and DEFAULT_THROTTLE_RATES). Furthermore, “django_ratelimit” was added to rate-limit authentication endpoints. For detail implementation, please check section “Security implementation – 5.1” above for detail information.

2. Software Composition Analysis (SCA)

   In this stage, CycloneDX was used to generate SBOMs for both frontend and backend. Meanwhile, Snyk scanned both backend and frontend dependencies:

   - Frontend: snyk tested 110 dependencies for known issues, and no vulnerable paths found.
   - Backend: snyk tested 12 dependencies for known issues, which found 1 vulnerability related to djangorestframework-simplejwt@5.5.0. This issue is vulnerable to

Information Exposure, which allows attacker to exploit and access application's resources.

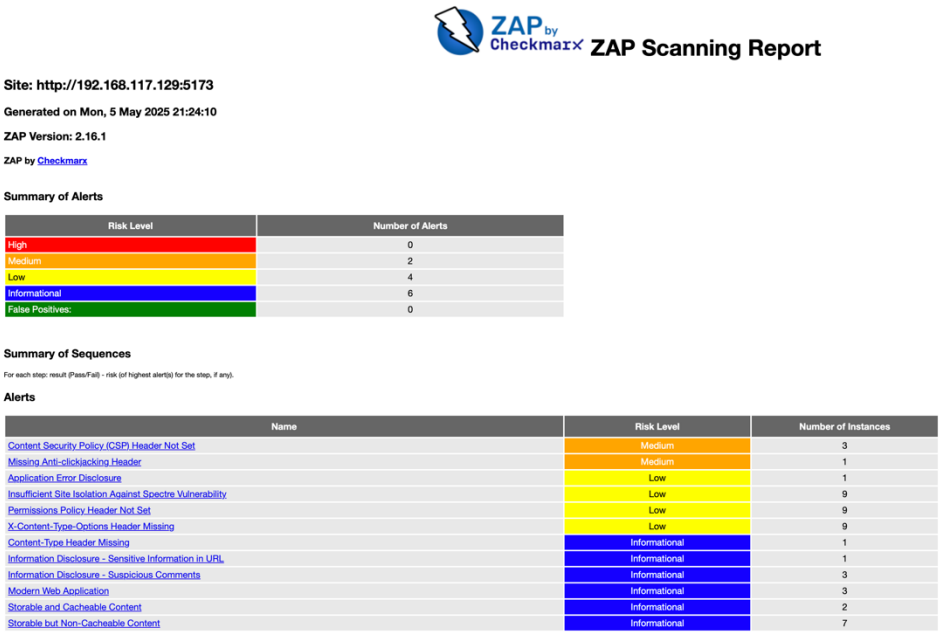
Mitigation and possible solution for improving security: as the current package version is the latest one, there is no available fixed version for this problem. However, this vulnerability is mitigated by our security implementation for refresh token, which prevents leaking user existence info. Please check section "Security implementation – 7.1" above for detail information about this implementation.

3. DAST – Dynamic Application Security Testing

For this security testing, OWASP ZAP was used to identify potential security issues in our web application for both frontend and backend.

3.1. DAST frontend

There were some issues found based on the frontend report, which is shown by figure 2 below:



ZAP by Checkmarx ZAP Scanning Report

Site: http://192.168.117.129:5173

Generated on Mon, 5 May 2025 21:24:10

ZAP Version: 2.16.1

ZAP by Checkmarx

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	4
Informational	6
False Positives:	0

Summary of Sequences

For each step: result (Pass/Fail) - risk (of highest alert(s) for the step, if any).

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	3
Missing Anti-clickjacking Header	Medium	1
Application Error Disclosure	Low	1
Insufficient Site Isolation Against Spectre Vulnerability	Low	9
Permissions Policy Header Not Set	Low	9
X-Content-Type-Options Header Missing	Low	9
Content-Type Header Missing	Informational	1
Information Disclosure - Sensitive Information in URL	Informational	1
Information Disclosure - Suspicious Comments	Informational	3
Modern Web Application	Informational	3
Storable and Cacheable Content	Informational	2
Storable but Non-Cacheable Content	Informational	7

Figure 2: ZAP scanning report for frontend

There are several DAST issues are technically served by the backend as the application architecture is React frontend + Django backend. For example, these issues should be fixed in the backend:

- Content Security Policy (CSP) Header Not Set
- Missing Anti-Clickjacking Header (X-Frame-Options)
- Permissions-Policy Header Not Set
- X-Content-Type-Options Missing
- Cross-Origin Headers for Spectre Mitigation (CORP, COOP, COEP)

On other side, some issues should be fixed in the frontend:

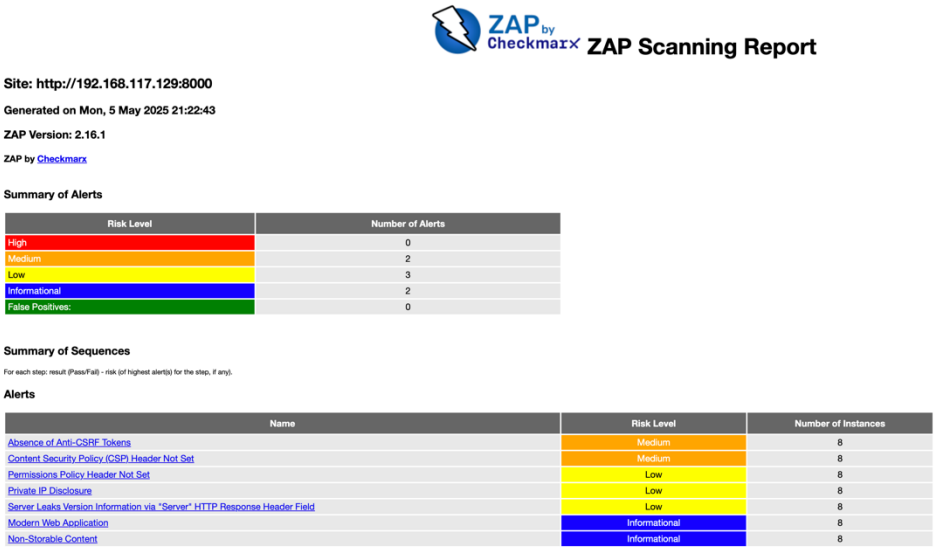
- Application Error Disclosure
- Content-Type Header Missing
- Information Disclosure - Sensitive Information in URL
- Information Disclosure - Suspicious Comments

Mitigation and possible solution for improving security:

- In the backend, a custom SecurityHeadersMiddleware was implemented to mitigate most key issues and ensure that security headers are properly handled. Detailed information about this implementation can be found in the section “Security implementation – 5.2” above.
- On the frontend, error disclosure is addressed using a global error boundary, which prevents source files and stack traces from being exposed in production. Additionally, since the application uses cookie-based authentication, access and refresh tokens are not included in the URL. This implementation ensures that sensitive information is not passed via URL parameters.
- It is worth noting that, because the frontend is currently running in a development environment (using Vite), some issues may continue to appear during scans. However, these issues will be properly addressed in the production environment.

3.2.DAST backend

There were some issues found based on the backend report, which is shown by figure 3 below:



ZAP by Checkmarx ZAP Scanning Report

Site: http://192.168.117.129:8000

Generated on Mon, 5 May 2025 21:22:43

ZAP Version: 2.16.1

ZAP by Checkmarx

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	3
Informational	2
False Positives:	0

Summary of Sequences

For each step: result (Pass/Fail) - risk (of highest alert(s) for the step, if any).

Alerts

Name	Risk Level	Number of Instances
Absence of Anti-CSRF Tokens	Medium	8
Content Security Policy (CSP) Header Not Set	Medium	8
Permissions Policy Header Not Set	Low	8
Private IP Disclosure	Low	8
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	8
Modern Web Application	Informational	8
Non-Storable Content	Informational	8

Figure 3: ZAP scanning report for backend

As reported by ZAP scanning, there are some issues related to:

- Absence of Anti-CSRF Tokens
- Content-Security-Policy header not set
- Permissions-Policy header not set
- Server leaks version info (Server: header)
- Private IP Disclosure

Mitigation and possible solution for improving security:

- The @csrf_protect is applied to directly mitigates CSRF risks, which enforce CSRF protection.

- “SecurityHeadersMiddleware” includes “Content-Security-Policy” to prevent XSS and code injection.

- “SecurityHeadersMiddleware” includes “Permissions-Policy” to prevent abuse of device features like camera/microphone.

- As the application is run in development mode (DEBUG = true), some of these issues maybe still being reported.

- and aligns with [OWASP A01:2021 - Broken Access Control] and [A05: Security Misconfiguration].

4. Trivy File System & Container Scanning

In this stage, trivy tool was used to perform file system scan and container scan. This resulted in reporting multiple issues in our application. The Figure 4 below illustrates a part of the image scan report:

util-linux-extra	CVE-2022-0563	LOW	2.38.1-5+deb12u3		https://access.redhat.com/security/cve/CVE-2022-0563 https://blog.trailofbits.com/2023/02/16/suid-logic-bug-linux-readline/ https://lore.kernel.org/util-linux/20220214110609.msiwlm457ngoic6w%40ws.net.home/T/#u Toggle more links
uuid-dev	CVE-2022-0563	LOW	2.38.1-5+deb12u3		https://access.redhat.com/security/cve/CVE-2022-0563 https://blog.trailofbits.com/2023/02/16/suid-logic-bug-linux-readline/ https://lore.kernel.org/util-linux/20220214110609.msiwlm457ngoic6w%40ws.net.home/T/#u Toggle more links
wget	CVE-2021-31879	MEDIUM	1.21.3-1+deb12u1		https://access.redhat.com/security/cve/CVE-2021-31879 https://mail.gnu.org/archive/html/bug-wget/2021-02/msg00002.html https://nvd.nist.gov/vuln/detail/CVE-2021-31879 Toggle more links
wget	CVE-2024-10524	MEDIUM	1.21.3-1+deb12u1		http://www.openwall.com/lists/oss-security/2024/11/18/6 https://access.redhat.com/security/cve/CVE-2024-10524 https://git.savannah.gnu.org/cgit/wget.git/commit/?id=c419542d956a2607bbce5df64b9d378a8588d778 Toggle more links
zlib1g	CVE-2023-45853	CRITICAL	1:1.2.13.dfsg-1		http://www.openwall.com/lists/oss-security/2023/10/20/9 http://www.openwall.com/lists/oss-security/2024/01/24/10 https://access.redhat.com/security/cve/CVE-2023-45853 Toggle more links
zlib1g-dev	CVE-2023-45853	CRITICAL	1:1.2.13.dfsg-1		http://www.openwall.com/lists/oss-security/2023/10/20/9 http://www.openwall.com/lists/oss-security/2024/01/24/10 https://access.redhat.com/security/cve/CVE-2023-45853 Toggle more links
No Misconfigurations found					
python-pkg					
Package	**Vulnerability ID**	**Severity**	**Installed Version**	**Fixed Version**	**Links**
Django	CVE-2025-27556	MEDIUM	5.1.7	5.0.14, 5.1.8	http://www.openwall.com/lists/oss-security/2025/04/02/2 https://access.redhat.com/security/cve/CVE-2025-27556 https://docs.djangoproject.com/en/dev/releases/security Toggle more links
Django	CVE-2025-27556	MEDIUM	5.1.7	5.0.14, 5.1.8	http://www.openwall.com/lists/oss-security/2025/04/02/2 https://access.redhat.com/security/cve/CVE-2025-27556 https://docs.djangoproject.com/en/dev/releases/security Toggle more links
setuptools	CVE-2024-6345	HIGH	65.5.1	70.0.0	https://access.redhat.com/errata/RHSA-2024:6726 https://access.redhat.com/security/cve/CVE-2024-6345 https://bugzilla.redhat.com/2297771 Toggle more links

Figure 4: Trivy report for image scan

In file system report, there were two issues related to Django version (potential Denial-of-Service (DoS) attack) and hardcoded JWT secret.

In image system report, there were several issues with medium, high, and critical severity levels. These vulnerabilities are related to different security aspects from buffer overflow to potential DoS. Please check detail information from this link folder "https://github.com/ndsang001/Secure_To_Do_App/tree/main/documents/Jenkins_artifacts/Trivy/"

Mitigation and possible solution for improving security:

- For vulnerabilities found by trivy file system scan, new package of Django and setuptools have been updated to fixed version to ensure the application handling of user inputs properly as well as preventing malicious code execution during package installation or build processes.

- For issues found in the Trivy image scan, we have conducted research on many of them. However, due to the high number of reported vulnerabilities and the constraint of the time limit, we can only offer some possible solutions to mitigate these problems.

  + Remove unused (unnecessary) packages: Most of the reported vulnerable packages are never used, so we can safely remove these packages from the Docker image to improve security. Tools like docker-slim or manual analysis with strace can help detect unused libraries. For example, the libbluetooth3 and libbluetooth-dev packages are implemented for handling Bluetooth communication, which is not required for our current application (project). Therefore, we can safely remove these packages.

  + Monitor for patched updates for essential packages: These packages must be monitored and updated frequently to implement the latest fixed versions.

  + Use a minimal base image: Instead of using full Debian, developers can consider using python:3.12-slim. This implementation helps reduce the attack surface of our image by including only essential runtime libraries, drastically decreasing the number of CVEs and exposure to attack vectors.

5. Artifacts

   Full artifacts folder from Jenkins pipeline could be found via this link: “https://github.com/ndsang001/Secure_To_Do_App/tree/main/documents/Jenkins_artifacts/”

Future work and plan

The application currently works well in development mode, so the next step should be testing it in a production environment. Additionally, remaining security risks (related to the image scan) should be properly addressed to ensure the application's security. At the current stage, there is no role-based access control implementation, as it would be redundant for the application's current use by a private user. However, this implementation should be considered for scalable security in the future. Furthermore, there is no transport encryption for data at rest or in transit implemented, so field level encryption for database should be highly considered in the next development phase.

On other hand, CI/CD testing for security should be implemented in the next development phase, aligning with A08:2021 – Software and Data Integrity Failures. To address security logging and monitoring failures (A09:2021), Django’s logging configuration should be added to capture login_user failures or suspicious rate-limit blocks.

In frontend, the application interface works fine at present though designers should enhance both its visual appeal and make it easier to use. The program's front-end development would be my primary focus through improvement of both interface responsiveness and device compatibility if I continued to build on the current system.

To enhance application usage, new features could be implemented, such as a group to-do feature. This feature could allow the application to function like a Kanban board, supporting development teams with a shared to-do tracklist.

AI usage

To complete the document work and project, we have used different AI such as ChatGPT and Copilot to support for:

- Researching topic and planning the project implementation.
- Code reviewing and debugging.
- Documentation checks for grammar and word usage

Current issues notice

With the latest version of the application, most of the functions and features works well. However, there would be error logged when logged user tries to access the application again after restarting the backend, which is related to “refresh” token handling method. For temporary solution if the page is not shown up, user can try to refresh the page and wait for few seconds to get into login page.