# بررسی آسیب‌پذیری‌های امنیتی در سرورهای FastAPI

## حملات XSS (Cross-Site Scripting)

در حملات XSS، مهاجم کدهای مخرب (معمولاً اسکریپت‌های جاوااسکریپت) را در محتوای قابل نمایش برای کاربر تزریق می‌کند. برای جلوگیری از XSS باید هرگونه ورودی کاربر را **کدگذاری خروجی** (ان‌تی‌آرکیو کردن حروف <، > و …) کرده و محتوای HTML را درون نشانه‌گذاری‌های مناسب قرار داد. در FastAPI می‌توان از ویژگی‌های قالب‌دهی Jinja2 با **Autoescape** استفاده کرد و همچنین هدر **CSP** مناسب اعمال نمود. همچنین نرمال‌سازی (Sanitize) ورودی‌ها و فیلتر کردن کاراکترهای خطرناک قبل از ذخیره یا بازپخش، کمک بزرگی به جلوگیری از XSS می‌کند[[1]](https://security.snyk.io/vuln/SNYK-PYTHON-FASTAPIADMIN-7833827#:~:text=,HTML%20or%20URL%20encoded%20equivalents)[[2]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,the%20security%20of%20your%20application). برای مثال، یک آسیب‌پذیری XSS شناخته‌شده در بسته‌ی fastapi-admin گزارش شده است که به مهاجم اجازه می‌داد با تزریق در نام محصول، اسکریپت اجرا کند[[3]](https://security.snyk.io/vuln/SNYK-PYTHON-FASTAPIADMIN-7833827#:~:text=Affected%20versions%20of%20this%20package,into%20the%20Product%20Name%20parameter).

* **جلوگیری در FastAPI:** مطمئن شوید که تمام داده‌های متنی خروجی در قالب HTML کدگذاری شده‌اند. از کتابخانه‌هایی مثل **Bleach** یا روش‌های امن Jinja2 برای تصفیه HTML استفاده کنید. هدر **Content-Security-Policy** را تنظیم کرده و اجرا شدن اسکریپت‌های ناامن را محدود کنید. در کد ارائه‌شده نیز ورودی‌ها توسط تابع sanitize\_input پاکسازی می‌شوند، اما باید در لایه خروجی (قالب‌ها و پاسخ‌ها) نیز اعمال نشانه‌گذاری امن انجام شود.

## حملات CSRF (Cross-Site Request Forgery)

حملات CSRF زمانی رخ می‌دهد که مرورگر کاربر بدون اطلاع او و در قالب درخواست‌های معتبر (مثلاً کوکی واردشده) عملی ناخواسته را انجام می‌دهد. اگر توکن احراز هویت در کوکی ذخیره شده باشد، خطر CSRF وجود دارد. راهکار رایج تولید و بررسی توکن‌های مخفی (CSRF Token) برای فرم‌ها و درخواست‌های حساس است. در FastAPI می‌توان از کتابخانه‌هایی مانند **fastapi-csrf-protect** استفاده کرد[[4]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=FastAPI%20CSRF%20Protect) که توکن‌های امضاشده را در کوکی (با flag‌های ایمن) یا هدر ارسال می‌کند. تنظیمات کوکی نیز مهم است: کوکی CSRF و احراز هویت را با **HttpOnly** و **Secure** و **SameSite** مناسب برچسب‌گذاری کنید تا حملات CSRF کاهش یابد[[5]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=%2C%20the%20library%20uses%20an,as%20well%20as%20templated%20static).

* **جلوگیری در FastAPI:** در اپلیکیشن‌های SPA ممکن است خطر CSRF کم باشد، اما اگر از کوکی‌های اعتبارسنجی استفاده می‌کنید باید حتماً مکانیسم CSRF پیاده شود. کتابخانه‌ی fastapi-csrf-protect راهکاری استاندارد است[[4]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=FastAPI%20CSRF%20Protect). در کد ارائه‌شده توکنی برای CSRF در نظر گرفته شده اما بررسی آن به طور کامل پیاده نشده است. بهتر است پس از ورود کاربر، توکن CSRF ایجاد و در کوکی (همراه HttpOnly/Secure/SameSite) یا در یک هدر مخصوص قرار گیرد و روی درخواست‌های تغییر دهنده (POST/PUT/DELETE) بررسی شود.

## تزریق SQL (SQL Injection)

وقتی کوئری‌های پایگاه‌داده با رشته‌های ورودی کاربر ساخته شوند، مهاجم می‌تواند با قرار دادن قطعات دستوری SQL در ورودی، کوئری‌ها را تغییر دهد. راه اصلی جلوگیری از SQLi این است که **هرگز از ادغام دستی رشته برای ساخت کوئری استفاده نکنیم** و به جای آن از پارامترهای SQL ایمن بهره ببریم. ابزارهای ارتباط با پایگاه‌داده (مانند SQLite، PostgreSQL و…) به طور معمول از پارامترگذاری پشتیبانی می‌کنند. مثال:

await conn.execute("SELECT \* FROM users WHERE email = ?", (user\_email,))

به جای قرار دادن مستقیم user\_email در رشته‌ی کوئری. استفاده از پارامتر پرس‌وجو (Query Parameters) جلوی تزریق را می‌گیرد[[2]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,the%20security%20of%20your%20application)[[6]](https://realpython.com/prevent-python-sql-injection/#:~:text=match%20at%20L631%20think%20about,compose%20a%20query%20with%20parameters).

* **جلوگیری در FastAPI:** حتماً از پارامترهای جایگزین (? یا :param) در اجرای کوئری‌ها استفاده کنید. در کد ارائه‌شده اغلب کوئری‌ها به شکل پارامتری نوشته شده‌اند که بسیار خوب است. علاوه بر آن، اعتبارسنجی ورودی‌ها (از طریق Pydantic و تابع sanitize\_input) انجام شده است[[2]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,the%20security%20of%20your%20application). اما نباید تنها به شناسایی الگوهای خاص (مثل حذف کلمات DROP یا SELECT) اتکا کرد؛ بهتر است کوئری‌سازای امن (ORM یا پارامترساز) به کار رود. برای مثال، استفاده از **SQLAlchemy** یا **Tortoise-ORM** به طور خودکار جنبه‌های امنیتی را بهبود می‌دهد.

## تزریق فرمان (Command Injection)

در این آسیب‌پذیری، اگر ورودی کاربر مستقیماً در فرمان‌های سیستم (مثلاً os.system یا subprocess.run) قرار گیرد، مهاجم می‌تواند فرمان‌های دلخواه را اجرا کند. جلوگیری از آن ساده است: **هرگز** ورودی کاربر را مستقیماً در shell=True به کار نبرید و همیشه از فرم لیستی استفاده کنید. به عنوان مثال:

# نادرست: shell=True و ورودی کاربر مستقیماً در دستور  
subprocess.run(f"ls {user\_input}", shell=True)  
  
# صحیح: استفاده از لیست آرگومان‌ها بدون shell  
subprocess.run(["ls", user\_input])

با این کار جلوی اجرای شل توسط کاربر گرفته می‌شود[[7]](https://snyk.io/blog/command-injection-python-prevention-examples/#:~:text=subprocess.run%28%22ls%20).

* **جلوگیری در FastAPI:** اگر ضرورتی برای اجرای فرمان سیستم دارید (مثلاً مدیریت فایل‌های محلی)، حتماً از subprocess.run(args\_list, shell=False) استفاده کرده و ورودی کاربر را اعتبارسنجی کنید. هیچ کجای کد ارائه‌شده به اجرای دستورات سیستم نمی‌پردازد، اما اگر اضافه شود باید مراقب این نکات بود. همچنین برای برنامه‌های تحت وب ترجیحاً از کتابخانه‌های سطح بالا (مثلاً برای پردازش تصویر یا فایل) استفاده کنید که نیازی به فرمان شل ندارند.

## مدیریت نشست‌ها و احراز هویت

توسعه برنامه‌های وب باید طوری باشد که نشست‌های کاربری را ایمن مدیریت کند. در مدل FastAPI معمولاً از **توکن JWT** در کوکی یا هدر استفاده می‌شود. نکات کلیدی عبارتند از: - تعیین **زمان انقضای مناسب** برای توکن‌ها. در کد ارائه‌شده، زمان اعتبار توکن بسیار کوتاه (۵ دقیقه) در نظر گرفته شده که امن است اما ممکن نیاز به مکانیزم تجدید (Refresh Token) داشته باشد. - قابلیت **ابطال نشست** (مثلاً هنگام خروج کاربر) با حذف یا منسوخ کردن توکن‌ها. - محدود کردن **تعداد نشست همزمان** برای هر کاربر (جلسه همزمان): در تنظیمات مقدار ۳ تعیین شده اما در عمل بررسی مربوطه غیرفعال است (همیشه True برمی‌گردد). باید در دیتابیس یا کش تعداد نشست‌ها را ذخیره و کنترل کرد. - ذخیره‌ی امن کوکی (HttpOnly/Secure/SameSite) که در کد رعایت شده است[[5]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=%2C%20the%20library%20uses%20an,as%20well%20as%20templated%20static).

همچنین مدیریت تلاش‌های ورود ناموفق (Brute-force) با محدود کردن تعداد تلاش‌ها (مثلاً ۵ بار) و قفل موقت حساب (یا IP) در کد پیاده شده است. در کل احراز هویت با JWT و رمز عبور هش‌شده قابل قبولی انجام شده اما می‌توان با افزودن **احراز هویت دومرحله‌ای (2FA)** و ذخیره نشست‌های فعال در پایگاه داده، سطح امنیت را بالاتر برد.

## حملات انکار سرویس (DoS و DDos)

محدودیت نرخ درخواست (**Rate Limiting**) برای جلوگیری از پر شدن سرور با درخواست‌های زیاد حیاتی است. با تعیین سقف درخواست برای هر IP یا کاربر در یک بازه زمانی مشخص، می‌توان حملات DoS ساده را خنثی کرد[[8]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=Implementing%20rate%20limiting%20is%20an,security%20of%20your%20FastAPI%20application). در FastAPI می‌توان از کتابخانه‌هایی مانند **fastapi-limiter** استفاده کرد[[9]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,any%20other%20criteria%20you%20choose) که به سرعت محدودیت درخواست را اعمال می‌کنند.

* **اقدامات پیشگیرانه:** نصب و راه‌اندازی middleware مربوط به محدودیت نرخ؛ بررسی حجم درخواست‌ها (مثلاً حجم بدنه‌ی فایل‌های آپلود نباید بیش از حد باشد)؛ استفاده از سرور HTTP ایمن (مانند Nginx) برای مقابله با حملات سطح پایین؛ ابزارهای مانیتورینگ ترافیک. در کد ارائه‌شده، middleware محدودیت نرخ وجود دارد و برای مسیرهای API سقف متفاوتی تنظیم شده است که بسیار مناسب است[[8]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=Implementing%20rate%20limiting%20is%20an,security%20of%20your%20FastAPI%20application)[[9]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,any%20other%20criteria%20you%20choose).

## نشت اطلاعات حساس

* **خطاها و لاگ‌ها:** اطلاعات حساس (مانند جزئیات خطا، رشته‌های اتصال به پایگاه‌داده و غیره) نباید به کاربر عادی نمایش داده شود یا در لاگ‌ها به طور آشکار ثبت شود. پاسخ‌های خطا در کد معمولاً عدد استاتوس و پیغام کلی (“Login error” یا پیام فارسی) دارند و اطلاعات داخلی فاش نمی‌شود که خوب است.
* **نام‌های پیش‌فرض و محیط توسعه:** مطمئن شوید تغییرات ابتدایی را انجام داده‌اید (مانند تغییر SECRET\_KEY از مقدار پیش‌فرض) و محیط توسعه (debug) در پروداکشن غیرفعال است. در کد، مقدار SECRET\_KEY از فایل .env بارگذاری می‌شود که مناسب است.

## امنیت کوکی‌ها

کوکی‌هایی که برای نگهداری توکن یا اطلاعات نشست استفاده می‌شوند باید با گزینه‌های امنیتی مناسب تنظیم شوند: - **HttpOnly:** (دسترسی به کوکی از طریق JavaScript غیرممکن باشد) که در کد استفاده شده است[[5]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=%2C%20the%20library%20uses%20an,as%20well%20as%20templated%20static)[[10]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=response.set_cookie%28%20key%3D,). - **Secure:** (کوکی فقط از طریق HTTPS ارسال شود) که بسته به محیط (production) تنظیم شده است[[5]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=%2C%20the%20library%20uses%20an,as%20well%20as%20templated%20static)[[11]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=httponly%3DTrue%2C%20secure%3Dos.getenv%28,). - **SameSite:** (محدودیت ارسال کوکی در درخواست‌های بین‌مبدا) که در کد مقدار Lax دارد، مناسب است تا CSRF کاهش یابد.

در کد نمونه، کوکی access\_token با همان گزینه‌های توصیه‌شده (HttpOnly، Secure، SameSite) تنظیم شده است[[5]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=%2C%20the%20library%20uses%20an,as%20well%20as%20templated%20static)[[10]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=response.set_cookie%28%20key%3D,) که وضعیت خوبی است. تنها نکته آن است که کوکی language که تنظیم محلی زبان است، HttpOnly نیست که عادی است و برای کاربردش ایراد ندارد.

## امنیت هدرهای HTTP

استفاده از هدرهای امنیتی استاندارد می‌تواند نفوذ را کاهش دهد: - X-Content-Type-Options: nosniff، X-Frame-Options: DENY (جلوگیری از Clickjacking)، Strict-Transport-Security (اجبار HTTPS) و Content-Security-Policy (محدودیت منابع) در تنظیمات دیده می‌شوند. در کد، تمام این هدرها (همراه با تنظیمات Cache-Control و COOP/COEP) به صورت مرکزی اعمال می‌گردد[[12]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20apply_security_headers%28response%29%3A%20,header%5D%20%3D%20value). مهم‌ترین نکته این است که خط تولید (production) باید HSTS و CSP مناسب را داشته باشد و اطلاعات حساس در هدرها (مانند Server/Powered-By) فاش نشود، که در کد Server به نام دلخواه تغییر یافته و X-Powered-By حذف شده است[[13]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=,By).

## کنترل دسترسی (Authorization)

کاهش آسیب‌پذیری‌های دسترسی به معنی اطمینان از این است که هر کاربر فقط مجوزهای تعریف‌شده را دارد. اقدامات لازم: - تعیین نقش‌ها (Role) و بررسی آنها در هر درخواست. در کد role در جدول کاربران ذخیره شده ولی کنترل نقش‌ها در روترها مشخص نیست (برای مثال، برخی مسیرهای مدیریت فقط باید توسط نقش admin اجرا شوند). باید در endpointهای حساس (مانند حذف کاربر، تغییر تنظیمات سطح بالا) بررسی شود که نقش کاربر، admin است. - محدود کردن دسترسی به فایل‌ها و مسیرهای خصوصی روی سرور. در کد، مسیرهای ایستا (/static, /gallery و …) به صورت جداگانه محدود شده و تزریق مسیر جلوگیری شده است. - استفاده از الگوی *least privilege* (کمترین دسترسی) در طراحی API. FastAPI این موارد را برعهده توسعه‌دهنده می‌گذارد؛ در کد ارائه‌شده عمده مسیرها به جز مسیرهای عمومی (/login, /register و …) نیازمند کاربر توکن‌شده هستند که مناسب است.

## احراز هویت (Authentication)

در احراز هویت از روش‌های استاندارد مانند JWT استفاده شده است. نکاتی که باید رعایت شوند: - **رمزنگاری ارتباطات:** همیشه از HTTPS استفاده کنید تا تبادل توکن/کلمه‌عبور در شبکه رمزنگاری شود. در کد، Strict-Transport-Security اعمال می‌شود. - **توکن امن:** کلید مخفی JWT باید قوی و محرمانه باشد. استفاده از env برای بارگذاری آن مناسب است. در صورت حمله احتمالی، نیاز به تعویض دوره‌ای کلیدهاست. - **مدیریت پاسخ خطا:** هنگام ورود نادرست نباید جزئیات سیستم فاش شود (در کد پیام خطا کلی است که مناسب است).

## ذخیره‌سازی رمزهای عبور

هیچ‌گاه رمز عبور کاربران را به شکل متن ساده ذخیره نکنید. به جای آن از الگوریتم‌های **هش قوی** استفاده کنید. OWASP توصیه می‌کند از الگوریتم‌هایی مثل BCrypt یا Argon2id با هزینه‌ی مناسب استفاده شود. برای BCrypt، حداقل work factor=10 پیشنهاد شده است[[14]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=,256). در کد ارائه‌شده از bcrypt با 12 دور تکرار استفاده شده که مطابق توصیه‌ی OWASP (حداقل ۱۰) است[[14]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=,256). علاوه بر این، این الگوریتم‌ها به صورت خودکار نمک (salt) تصادفی بر روی پسورد اعمال می‌کنند[[15]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=Modern%20hashing%20algorithms%20such%20as,are%20required%20when%20using%20them)، بنابراین لازم نیست کاربر خودتان رمز یا نمکی اضافه تولید کنید.

* **راهبردهای تکمیلی:** می‌توان برای لایه‌های بالاتر امنیت، از “pepper” (یک مقدار محرمانه و مشترک جدا از DB) استفاده نمود و نیز هش‌شدن مکرر را در کلاس‌های متفاوت به کار برد. اما نکته اصلی این است که قدرت هش تنظیم شده باشد؛ در این کد bcrypt نصب شده و در توابع hash\_password و verify\_password استفاده شده است که مناسب است.

## مدیریت فایل‌ها و مسیرها

اگر برنامه اجازه‌ی آپلود یا نمایش فایل می‌دهد، باید: - **اعتبارسنجی پسوند و نوع فایل:** فقط پسوندهای بی‌خطر (عکس، ویدئو و…) را مجاز کنید و اندازه‌ی فایل را محدود کنید. در کد، validate\_file\_upload پسوندها و MIME Type را بررسی می‌کند و فهرست بلندی از پسوندهای خطرناک را مسدود کرده است. به عنوان مثال، برای تصاویر امکان صرفاً jpg/png و برای ویدئو mp4/webm داده شده و همچنین اندازه باید زیر حد تعریف‌شده باشد[[16]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20validate_file_upload,return%20False)[[17]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=,Invalid%20content%20type%3A%20%7Bcontent_type). - **اجتناب از traversal (دورزدن مسیر):** نام فایل دریافتی را پاکسازی کنید (فیلتر کردن ../ و غیره). در کد، توابع sanitize\_filename و validate\_path\_traversal پیچیده‌ای برای حذف الگوهای خطرناک و جلوگیری از مسیرگذاری اضافی دارند[[18]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20sanitize_filename%28filename%3A%20str%29%20,replace%28char%2C)[[19]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20validate_path_traversal%28filename%3A%20str%29%20,media%27%2C%20%27proc%27%2C%20%27sys%27%2C%20%27dev%27%2C%20%27tmp). - **سطوح دسترسی فایل‌ها:** اطمینان حاصل کنید که آپلودها خارج از دایرکتوری‌های امن ذخیره شوند و در صورت امکان با نام‌های تصادفی/مجاور ذخیره شوند.

کد نمونه به‌خوبی این موارد را در نظر گرفته و مسیرهای /gallery و /security\_videos را تحت اعتبارسنجی قرار داده است. تنها نکته آن است که پس از ذخیره، اطمینان از اجرای هیچ فایل اجرایی در شاخه‌ی فایل‌های استاتیک ضروری است.

## تحلیل وضعیت امنیتی کد ارائه‌شده

| آسیب‌پذیری | وضعیت فعلی کد | توضیحات |
| --- | --- | --- |
| XSS | تا حدی رعایت شده | ورودی‌ها با sanitize\_input پاکسازی می‌شوند؛ در قالب‌ها نیز خروجی‌بندی امن باید باشد. |
| CSRF | بخشاً پیاده شده (ناسازگار با فرم‌ها) | ساخت توکن CSRF در نظر گرفته شده اما استفاده‌ی کامل نشده است. توصیه به استفاده از کتابخانه fastapi-csrf-protect. |
| تزریق SQL | رعایت شده | کوئری‌ها به شکل پارامتری نوشته شده‌اند (علامت ?) که امنیت بالایی دارد[[2]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,the%20security%20of%20your%20application)[[6]](https://realpython.com/prevent-python-sql-injection/#:~:text=match%20at%20L631%20think%20about,compose%20a%20query%20with%20parameters). |
| تزریق فرمان | رعایت شده (در اجرای فرمانی وجود ندارد) | در کد نمونه هیچ subprocess وجود ندارد؛ اگر اضافه شود باید ورودی را لیستی ارسال کرد[[7]](https://snyk.io/blog/command-injection-python-prevention-examples/#:~:text=subprocess.run%28%22ls%20). |
| مدیریت نشست‌ها | ناقص | توکن JWT با زمان کوتاه استفاده شده؛ قابلیت‌های ابطال یا ریفرش توکن کامل نیست. کنترل تعداد نشست همزمان هنوز پیاده نشده (همیشه اجازه). |
| DoS (محدودیت نرخ) | رعایت شده | محدودیت نرخ درخواست بر اساس IP و مسیر وجود دارد[[8]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=Implementing%20rate%20limiting%20is%20an,security%20of%20your%20FastAPI%20application)[[9]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,any%20other%20criteria%20you%20choose). همچنین تلاش‌های ورود محدود شده است. |
| نشت اطلاعات | رعایت شده | پیام‌های خطا عمومی است و اطلاعات حساس لو نرفته است. |
| امنیت کوکی‌ها | رعایت شده | کوکی access\_token با HttpOnly, Secure و SameSite تنظیم شده[[5]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=%2C%20the%20library%20uses%20an,as%20well%20as%20templated%20static)[[10]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=response.set_cookie%28%20key%3D,). |
| هدرهای HTTP | رعایت شده | هدرهای امنیتی استاندارد (CSP، HSTS، X-Frame-Options و …) اعمال می‌شوند[[12]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20apply_security_headers%28response%29%3A%20,header%5D%20%3D%20value). |
| کنترل دسترسی | تا حدی رعایت شده | وجود وابستگی get\_current\_user برای بیشتر مسیرها؛ اما نقش‌بندی (role) برای مسیرهای خاص باید تکمیل شود. |
| احراز هویت | رعایت شده | استفاده از JWT و رمزنگاری HTTPS و کلید مخفی مناسب. |
| ذخیره‌سازی رمزها | رعایت شده | از bcrypt با ۱۲ دور استفاده شده که مطابق توصیه‌های OWASP است[[14]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=,256). |
| مدیریت فایل/مسیر | رعایت شده | نام فایل‌ها و مسیرها با توابع کمکی پاکسازی شده و از الگوهای traversal جلوگیری می‌شود. |

## توصیه‌های بهبود امنیت (مستوى حرفه‌ای)

* **استفاده از کتابخانه‌های استاندارد:** برای بسیاری از وظایف امنیتی، کتابخانه‌های آماده و تست‌شده وجود دارد. مثلاً از **fastapi-csrf-protect** برای CSRF، **fastapi-limiter** برای Rate Limiting و **fastapi-users** برای مدیریت کاربران (ثبت‌نام، ورود، ایمیل تأیید، 2FA و…) بهره بگیرید. این کتابخانه‌ها بهترین شیوه‌ها را پیاده می‌کنند و آزمون‌های امنیتی بر روی آن‌ها انجام شده است.
* **مدیریت نشست و توکن:** پیاده‌سازی Refresh Token و امکان ابطال توکن (ری‌ولک) را در نظر بگیرید. برای نشست‌های حساس از کتابخانه‌هایی مانند **Authlib** (برای OAuth2) استفاده کنید تا استانداردهای بین‌المللی رعایت شود. همچنین اطلاعات نشست (Session) را در بانک اطلاعات نگه دارید تا امکان بررسی تعداد نشست و ابطال آن‌ها وجود داشته باشد.
* **پایش و لاگ مناسب:** یک سیستم مانیتورینگ و لاگینگ (مانند Sentry یا ELK) راه‌اندازی کنید تا حملات احتمالی را به سرعت تشخیص دهید. لاگ‌ها را طوری ذخیره کنید که در صورت رخنه، حمله‌کننده نتواند به آن‌ها دسترسی پیدا کند.
* **SSL/TLS قوی:** همیشه سرور را با گواهی معتبر HTTPS تنظیم کنید و HSTS‌ را فعال نگه دارید. علاوه بر این، هدرهای امنیتی مانند CSP را به‌صورت دقیق بررسی و در صورت نیاز سختگیرانه‌تر کنید.
* **اعتبارسنجی و دسترسی لایه‌بندی:** از مدل‌های **Pydantic** برای تعریف اسکیماهای ورودی استفاده کنید (در کد نیز رعایت شده). اما برای اجرای کوئری‌ها یا دستورات حساس، علاوه بر صرفاً اعتبارسنجی رشته، از منطق کسب‌وکار و نقش‌ها مطمئن شوید. کنترل دسترسی را به‌صورت جزئی (row-level) نیز پیاده کنید تا فقط منابع متعلق به کاربر نمایش داده شود.
* **موازی‌سازی ایمن:** برای جلوگیری از حملات همزمان (race condition)، دستورات حساس مانند بازیابی رمز یا تغییرات مالی را اتمیک کنید (مثلاً از تراکنش یا نشانگر اشتراک‌گذاری استفاده کنید).
* **بررسی وابستگی‌ها:** به‌طور منظم از ابزارهایی مانند **bandit** یا **pip-audit** برای اسکن وابستگی‌های پایتون استفاده کنید تا آسیب‌پذیری‌های شناخته‌شده در کتابخانه‌ها را بیابید. همچنین در مستندات و انتشار‌های FastAPI‌ به‌روزرسانی‌ها را دنبال کنید.

## کتابخانه‌ها و ابزارهای امنیتی پیشنهادی برای FastAPI

* **fastapi-csrf-protect**: کتابخانه‌ای برای تولید و بررسی توکن‌های CSRF در اپلیکیشن‌های FastAPI. این کتابخانه با استفاده از کوکی امن و گزینه‌های HttpOnly/SameSite به سادگی CSRF را مدیریت می‌کند[[4]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=FastAPI%20CSRF%20Protect)[[5]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=%2C%20the%20library%20uses%20an,as%20well%20as%20templated%20static).
* **fastapi-limiter**: یک محدودکننده نرخ درخواست (Rate Limiter) سریع و آسان برای FastAPI که می‌تواند با Redis یا حافظه‌ی داخلی کار کند. با چند خط کد، محدودیت‌های IP یا کاربر را تنظیم می‌کند[[8]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=Implementing%20rate%20limiting%20is%20an,security%20of%20your%20FastAPI%20application)[[9]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,any%20other%20criteria%20you%20choose).
* **fastapi-users**: فریم‌ورکی جامع برای مدیریت ثبت‌نام، ورود، تأیید ایمیل، بازیابی رمز، احراز هویت JWT/OAuth و 2FA در پروژه‌های FastAPI. این کتابخانه از روش‌های رمزنگاری امن (مانند BCrypt) و ORMهای استاندارد پشتیبانی می‌کند.
* **Authlib**: برای پیاده‌سازی استاندارد OAuth2 (ورود با گوگل/فیس‌بوک و غیره) می‌توانید از Authlib استفاده کنید که با FastAPI سازگار است و امنیت سطح بالایی دارد.
* **SQLAlchemy** یا **Tortoise-ORM**: استفاده از ORM به‌جای نوشتن کوئری‌های دستی می‌تواند خطاهای امنیتی (مانند SQLi) را کاهش دهد و نگهداری کد را آسان‌تر کند.
* **python-jose** یا **PyJWT**: برای تولید و بررسی توکن‌های JWT توصیه می‌شود از کتابخانه‌های معتبر و به‌روز مانند python-jose استفاده کنید.
* **پکیج‌های رمزنگاری پیشرفته**: برای مدیریت امن داده‌های حساس (مانند رمزنگاری مبتنی بر کلید عمومی)، از کتابخانه‌هایی مانند cryptography یا PyNaCl بهره ببرید.
* **ابزارهای اسکن امنیتی**: در فاز توسعه از ابزارهایی مثل OWASP ZAP یا SonarQube برای اسکن برنامه استفاده کنید تا آسیب‌پذیری‌های شناخته‌شده را بیابید. هرچند این موارد کتابخانه نیستند اما از اهمیت بالایی برخوردارند.

<hr>

**منابع:** مستندات و مقالات امنیت FastAPI[[2]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,the%20security%20of%20your%20application)[[8]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=Implementing%20rate%20limiting%20is%20an,security%20of%20your%20FastAPI%20application)[[1]](https://security.snyk.io/vuln/SNYK-PYTHON-FASTAPIADMIN-7833827#:~:text=,HTML%20or%20URL%20encoded%20equivalents)[[20]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=FastAPI%20CSRF%20Protect)[[6]](https://realpython.com/prevent-python-sql-injection/#:~:text=match%20at%20L631%20think%20about,compose%20a%20query%20with%20parameters)[[7]](https://snyk.io/blog/command-injection-python-prevention-examples/#:~:text=subprocess.run%28%22ls%20)[[14]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=,256). اطلاعات مربوط به کد سرور از تحلیل کد ارائه‌شده استخراج شده است.

[[1]](https://security.snyk.io/vuln/SNYK-PYTHON-FASTAPIADMIN-7833827#:~:text=,HTML%20or%20URL%20encoded%20equivalents) [[3]](https://security.snyk.io/vuln/SNYK-PYTHON-FASTAPIADMIN-7833827#:~:text=Affected%20versions%20of%20this%20package,into%20the%20Product%20Name%20parameter) Cross-site Scripting (XSS) in fastapi-admin | CVE-2024-42818 | Snyk

<https://security.snyk.io/vuln/SNYK-PYTHON-FASTAPIADMIN-7833827>

[[2]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,the%20security%20of%20your%20application) [[8]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=Implementing%20rate%20limiting%20is%20an,security%20of%20your%20FastAPI%20application) [[9]](https://escape.tech/blog/how-to-secure-fastapi-api/#:~:text=FastAPI%20provides%20built,any%20other%20criteria%20you%20choose) How to secure APIs built with FastAPI: A complete guide

<https://escape.tech/blog/how-to-secure-fastapi-api/>

[[4]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=FastAPI%20CSRF%20Protect) [[5]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=%2C%20the%20library%20uses%20an,as%20well%20as%20templated%20static) [[20]](https://www.stackhawk.com/blog/csrf-protection-in-fastapi/#:~:text=FastAPI%20CSRF%20Protect) FastAPI CSRF Protection: How to Secure Your API

<https://www.stackhawk.com/blog/csrf-protection-in-fastapi/>

[[6]](https://realpython.com/prevent-python-sql-injection/#:~:text=match%20at%20L631%20think%20about,compose%20a%20query%20with%20parameters) Preventing SQL Injection Attacks With Python – Real Python

<https://realpython.com/prevent-python-sql-injection/>

[[7]](https://snyk.io/blog/command-injection-python-prevention-examples/#:~:text=subprocess.run%28%22ls%20) Command injection in Python: examples and prevention | Snyk

<https://snyk.io/blog/command-injection-python-prevention-examples/>

[[10]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=response.set_cookie%28%20key%3D,) [[11]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=httponly%3DTrue%2C%20secure%3Dos.getenv%28,) [[12]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20apply_security_headers%28response%29%3A%20,header%5D%20%3D%20value) [[13]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=,By) [[16]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20validate_file_upload,return%20False) [[17]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=,Invalid%20content%20type%3A%20%7Bcontent_type) [[18]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20sanitize_filename%28filename%3A%20str%29%20,replace%28char%2C) [[19]](file://file-JBGvwhDfPyDcJ9odkV62uU#:~:text=def%20validate_path_traversal%28filename%3A%20str%29%20,media%27%2C%20%27proc%27%2C%20%27sys%27%2C%20%27dev%27%2C%20%27tmp) server\_fastapi.py

<file://file-JBGvwhDfPyDcJ9odkV62uU>

[[14]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=,256) [[15]](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#:~:text=Modern%20hashing%20algorithms%20such%20as,are%20required%20when%20using%20them) Password Storage - OWASP Cheat Sheet Series

<https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html>