Проектна задача по предметот Етичко Хакирање на ФИНКИ со тема:

**OWASP Топ 10 ранливости демонстрирани во Laravel**

Изработено од:

Бисера Галеска 211023

Бојана Аризанковска 211018

СОДРЖИНА

[ВОВЕД 3](#_Toc201691221)

[ПОТРЕБЕН ХАРДВЕР И СОФТВЕР 3](#_Toc201691222)

[Хардвер: 3](#_Toc201691223)

[Софтвер 3](#_Toc201691224)

[УПАТСТВО 4](#_Toc201691225)

[Чекор 1: Подготовка на развојната околина 4](#_Toc201691226)

[OWASP Топ 10 ранливости 5](#_Toc201691227)

[A01: Broken Access Control 5](#_Toc201691228)

[Чести недостатоци во контролата на пристап (Access Control) : 5](#_Toc201691229)

[Како да се справиме? 6](#_Toc201691230)

[Пример 6](#_Toc201691231)

# ВОВЕД

OWASP Топ 10 ги идентификува најкритичните безбедносни ризици со кои се соочуваат веб-апликациите (owasp.org). Овој проект има за цел да создаде Laravel веб-апликација која намерно ги демонстрира овие ранливости, нудејќи им на програмерите практично искуство за учење.

Преку овој проект, ќе научите:

* Да препознавате чести безбедносни стапици во веб-развојот,
* Како Laravel се справува со заканите, и
* Да ги примените најдобрите практики за ефикасна заштита на Вашите апликации.

Занемарувањето на практиките за безбедно кодирање може да доведе до кражба на податоци, оштетување на угледот, правни последици и прекин на услугите. Па затоа, со едукација на програмерите за овие ранливости, можеме значително да го намалиме бројот на небезбедни веб-апликации, правејќи го дигиталниот свет побезбеден за сите.

# ПОТРЕБЕН ХАРДВЕР И СОФТВЕР

## Хардвер:

|  |  |  |
| --- | --- | --- |
| Компонента | Минимум | Препорачано |
| CPU | 2 јадра со VT‑x/AMD‑V | 4 јадра |
| RAM | 8 GB | 16 GB |
| Меморија | 20 GB слободни | 40 GB SSD |
| OS | Windows 10 / macOS 12 / Linux 5.x | Последната стабилна верзија |

Софтвер:

|  |  |  |
| --- | --- | --- |
| ЦЕЛ | Алатка | Верзија |
| Container runtime | **Docker**/ Podman | 24+ |
| Оркестрација | Docker Compose | v2 |
| PHP | PHP | >= 8.1 |
| PHP Framework | Laravel | >= 10.x |
| База на Податоци | MySQL | Latest |
| Package manager | Composer | Latest |
| Веб прелистувач | Chrome/Firefox | Latest |
| IDE | VS Code, PhpStorm, … | Latest |

Опционално: VirtualBox / VMware ако работите со целосни виртуелни машини

***Безбедносно предупредување*:** *Овој проект намерно воведува ранливости. Апликацијата треба да се извршува само во изолирани средини. Не ја изложувајте на јавни мрежи.*

# УПАТСТВО

## Чекор 1: Подготовка на развојната околина

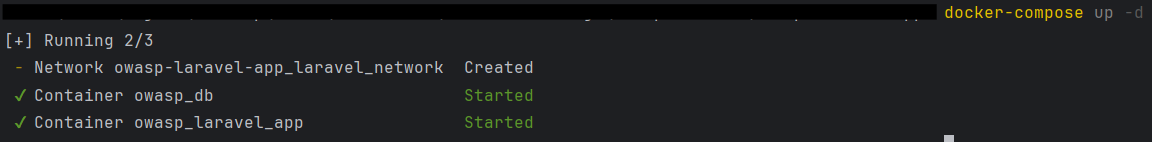
1. Инсталирајте Docker Desktop
2. Отворете терминал и клонирајте го гит репозиториумот:

git clone https://github.com/your‑org/laravel‑owasp‑demo.git

cd laravel‑owasp‑demo

1. Launch the containers:

docker compose up -d –build



1. Seed the database:

docker compose exec app php artisan migrate --seed

# OWASP Топ 10 ранливости

## A01: Broken Access Control

Access control – политика која изјаснува кои акции/дејства корисиникот може да ги извршува во рамки на системот, врз основа на неговите доделени дозволи. Кога овие контроли се погрешно конфигурирани, може да доведе до сериозни нарушувања на безбедноста, вклучувајќи откривање на информации, модификација и уништување на податоци или извршување на функции надвор од дефинираните граници на корисникот.

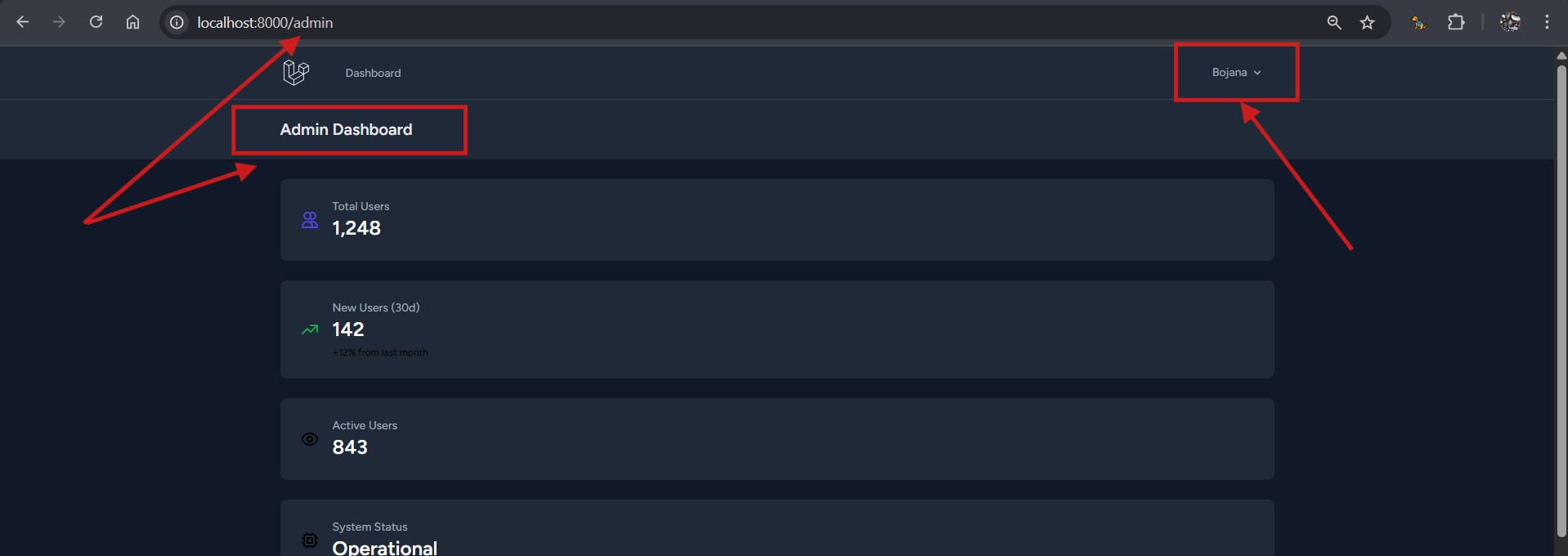
### Чести недостатоци во контролата на пристап (Access Control) :

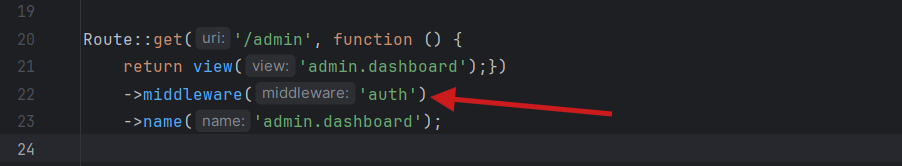
* Прекршување на најмалата привилегија / deny by default – се случува каде што пристапот треба да се дозволи само за одредени можности, улоги или корисници, но е достапен за сите. Во суштина, ако нешто не е експлицитно дозволено, треба да се одбие.
* Заобиколување на проверките (Bypassing Checks) - напаѓачите можат да ги заобиколат контролите за пристап со директно манипулирање со компонентите на апликацијата. Ова вклучува неовластено менување на параметрите (модифицирање на URL параметрите), принудно прелистување (директен пристап до ограничени URL адреси), менување на внатрешната состојба на апликацијата, модифицирање на HTML или користење на алатки за напад за манипулирање со API барањата..
* Небезбедни директни референци на објекти (IDOR) - напаѓачот може да ја прегледа или уреди сметката или податоците на друг корисник едноставно со промена на уникатен идентификатор во URL-то или API-барањето.
* Недостаток на API controls: Недостаток на автентикација за дејства како POST, PUT и DELETE.
* Ескалација на привилегии -Дејствување како помоќен корисник без соодветно овластување.
* Неовластено менување на метаподатоци: Менување на JWT, колачиња или скриени полиња за добивање зголемен пристап.
* Погрешни конфигурации на CORS - Овозможување неовластен пристап до API-ја од недоверливи извори.
* Присилно прелистување - Пристап до ограничени страници без автентикација или соодветни привилегии.

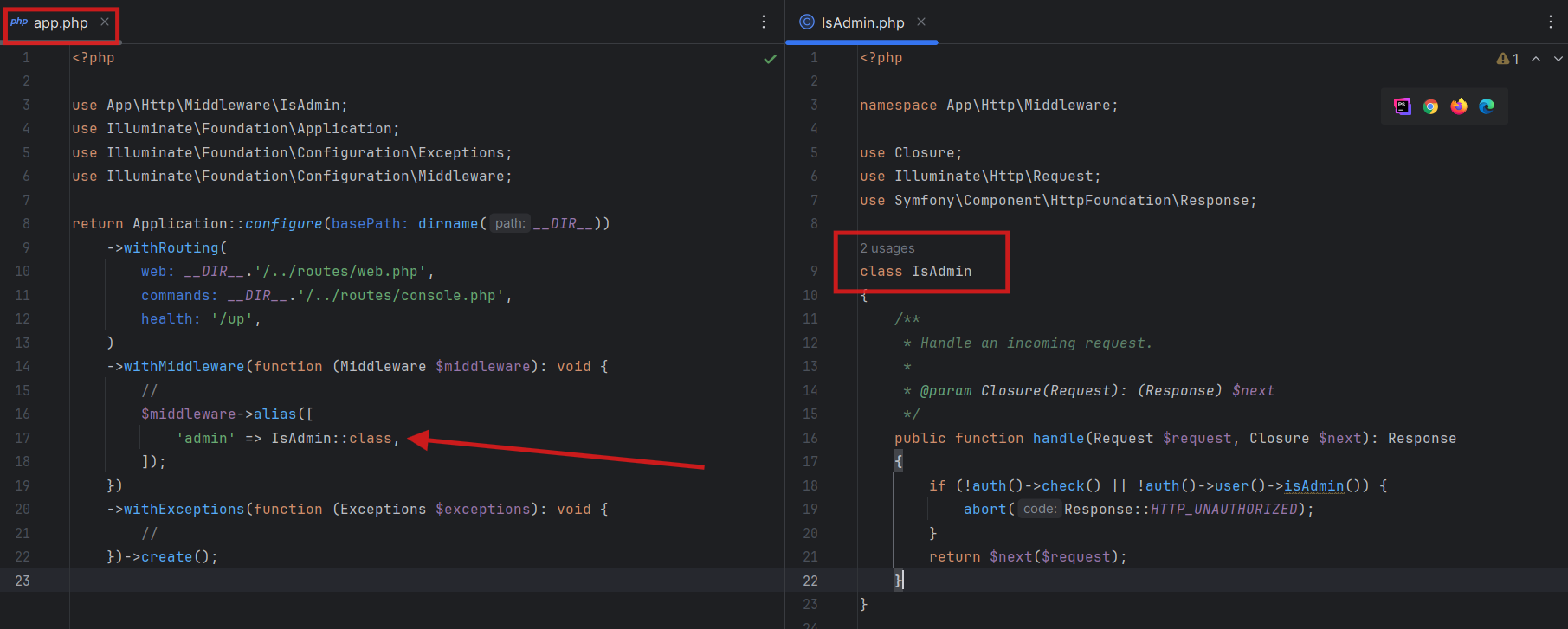
### Како да се справиме?

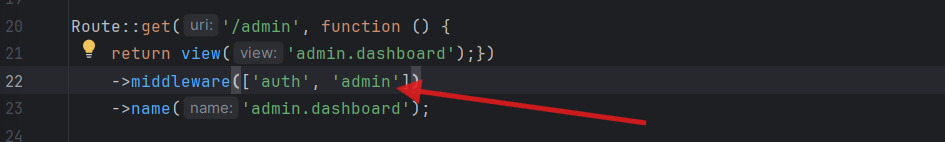
* Стандард да биде одбиј (Deny) - Ако ресурсот не е експлицитно јавен, секој пристап треба да биде одбиен по стандард.
* Централизирани и повеќекратно употребливи механизми - Имплементирајте механизми за контрола на пристап еднаш и применувајте ги конзистентно низ целата апликација. Минимизирајте ја употребата на споделување ресурси со вкрстено потекло (CORS) за да ги намалите потенцијалните вектори на напад.
* Спроведување на сопственост на записи - дизајнирај контроли за пристап такви што ќе ја налагаат сопственоста на записот, наместо само да прифаќаат дека корисникот може да креира, чита, ажурира или брише билокој запис.
* Имплементирајте ограничувања на деловната логика - Уникатните бизнис ограничувања специфични за апликацијата (на пр., максимална количина на нарачка) треба да се спроведени во доменските модели.
* Безбеден датотечен систем - Оневозможете листање на директориуми на веб-серверот и осигурете се дека чувствителните датотеки како што се .git или резервните датотеки не се достапни во рамките на веб-коренот.
* Сеопфатно евидентирање и известување (Log access control failures) - Имплементирајте соодветни известувања за администраторите кога е потребно
* Ограничување на брзината - Применете ограничувања на брзината на пристапот до API и контролерот.
* Робустно управување со сесии - Веднаш по одјавувањето, идентификаторите на сесии со состојба на серверот треба да станат невалидни. За JWT токени без состојба, користете краткотрајни токени за да го минимизирате прозорецот на можности за напаѓачите. За JWT со подолг век на траење, почитувајте ги стандардите на OAuth за поништување на токени.

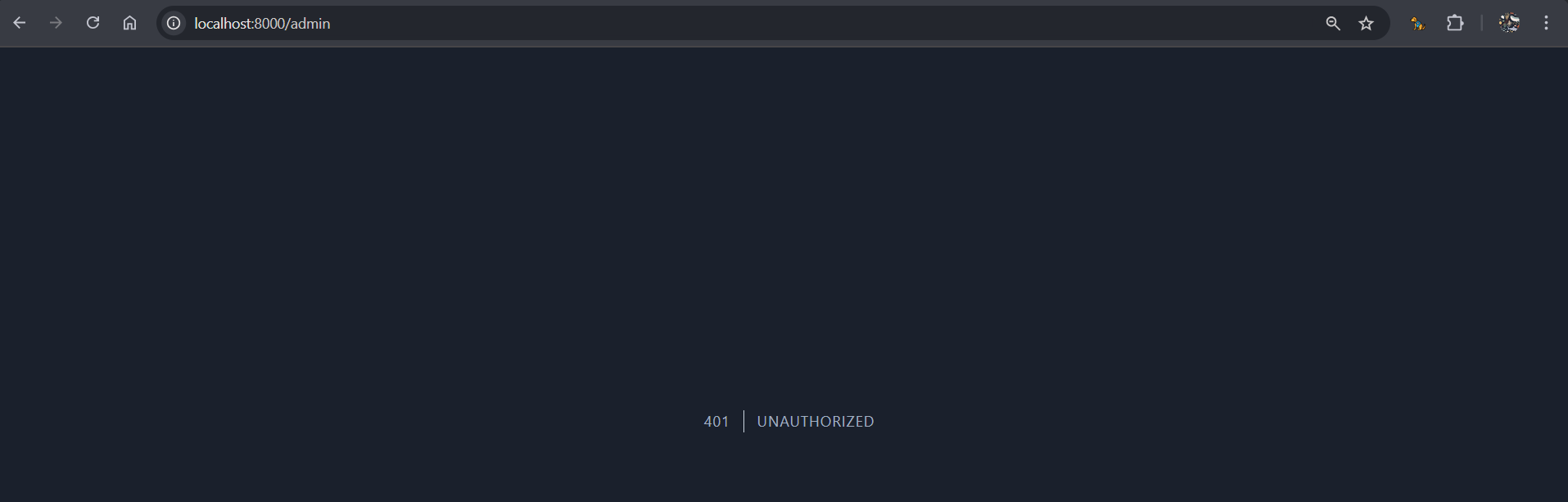
### Пример



****

****

****

****

## A02:2021 – Cryptographic Failures