[GYMNASIUM NAME]

INFORMATIKA

DVYLIKTA KLASĖ

(*your full name here)
***CITE AS:** Arija A. (*Ari Archer*) | ari@ari.lt | https://ari.lt/

SAUGUMO IR KODO KOKYBĖS BALANSAVIMAS PROGRAMAVIMO PRAKTIKOSE

(VIEŠOJO NAUDOJIMO VERSIJA)

Mokinio brandos darbas

(*) Brandos darbo vadovas (arba -ė)
informatikos mokytojas (arba -ė)
[Full legal name]
[Institution]

(*) Brandos darbo konsultantas (arba -a)
vyriausysis (-ioji) mokslo darbuotojas (arba -a)
[Full legal name]
[Institution]

[CITY NAME]
2026

TURINYS

REPUBLICATION STATEMENT (*)

This document is a redacted version of my *brandos darbas* (graduation/maturity project), which received a perfect score of 10/10. To balance academic accessibility with personal privacy, I have removed specific identifiers, including names of the author (me), consultants, and institutions. Due to ongoing legal constraints regarding my identity as a transgender woman, and to maintain a degree of personal privacy, this version utilises [brackets] and asterisks (*) to indicate redacted information. My goal in sharing this work is to provide a high-quality reference for future students choosing their own graduation topics. For further inquiries, please visit my website at https://ari.lt/.

This statement is NOT a part of my graduation project.

PERPUBLIKAVIMO PAREIŠKIMAS (*)

Šis dokumentas yra redaguota mano brandos darbo, įvertinto aukščiausiu 10 balų įvertinimu, versija. Siekiant suderinti akademinį prieinamumą ir asmens privatumą, iš darbo pašalinau konkrečius identifikatorius, įskaitant autoriaus (mano) bei konsultantų vardus ir pavardes, bei institucijų pavadinimus. Dėl teisinių aplinkybių, susijusių su mano kaip translytės moters tapatybe, bei siekiant apsaugoti privatumą, šioje versijoje [laužtiniais skliaustais] ir žvaigždutėmis (*) pažymėta redaguota informacija. Šio darbo publikavimo tikslas - suteikti kokybišką pavyzdį būsimiems abiturientams. Siekiant susiekti su manimi, apsilankykite mano svetainėje: https://ari.lt/.

Šis pareiškimas NĖRA mano brandos darbo dalis.

SANTRAUKA

Darbas nagrinėja kodo kokybės ir saugumo sąveiką programavimo praktikose (*gairių rinkinio, siekiant tam tikrų tikslų programiniame kode ar jo kūrimo procese*) per praktinius tyrimus su *Python* programavimo kalba ir dirbtinio intelekto (DI) įrankiais. Buvo sukurta ir išbandyta trylika internetinės sistemos versijų, ieškant optimalaus programavimo praktikų derinio. Sistemų analizė parodė, kad žmogaus vadovaujama praktika, kartu su statinės kodo analizės įrankiais, pasiekia geriausius rezultatus. Tyrimu taip pat nustatyta, kad saugumo kontroliniai sąrašai ir tinkamas išskaidymas į modulius yra esminiai aspektai siekiant optimalaus kodo saugumo ir kokybės balanso, o DI įrankiai tik pablogino esminius kodo kokybės ir saugumo aspektus.

Tyrimo tikslas. Nustatyti ir įgyvendinti programavimo praktiką kuri veiksmingai suderina saugumo ir kodo kokybės aspektus, taip užtikrinant patvaraus, patikimo ir ilgaamžio kodo kūrimą.

Tyrimo uždaviniai:

1. Išanalizuoti literatūrą, siekiant geriau suprasti kibernetinės saugos ir kodo kokybės pusiausvyros problemas.
2. Sukurti įrankį, leidžiantį pateiktam *Python* projektui įvertinti saugos ir kokybės rodiklius.
3. Parengti tyrime kuriamai internetiniai sistemai keliamus funkcinius reikalavimus.
4. Sudaryti eksperimento planą, siekiant įvertinti žmogaus darbo, DI ir kodo įvertinimo įrankių taikymo įtaką sistemai.
5. Atlikti eksperimentus, suplanuotų situacijų metu gautų sistemų kokybės ir saugumo rodiklių išgavimui.
6. Atlikti gautų rezultatų analizę, įvertinant, kaip skirtingos praktikos paveikia sistemos kodo saugumą ir kokybę.
7. Parengti rekomendacijas, siekiant gerinti kodo saugumą ir kokybę.

Raktiniai žodžiai: kibernetinė sauga, kodo kokybė, programavimo praktikos, DevOps, DevSecOps, Python Flask, vibe coding, statinė kodo analizė, dinaminė kodo analizė.

SUMMARY

The work examines the interaction between code quality and security in programming practices (*a set of guidelines for achieving certain goals in code or its production*) through practical research using the Python programming language and artificial intelligence (AI) tools. Thirteen versions of an internet system were created and tested in search of the optimal combination of programming practices. Analysis of the systems showed that human-guided practice, combined with static code analysis tools, achieves the best results. The study also found that security checklists and proper modularisation are essential aspects of achieving an optimal balance between code security and quality, while AI tools only worsened key aspects of code quality and security.

Aim of the study. Establish and implement programming practices that effectively balance security and code quality, ensuring the creation of robust, reliable, and long-lasting code.

Objectives of the study:

1. Analyse literature to better understand the problems of balancing cyber security and code quality.
2. Develop a tool that allows the Python project to be evaluated in terms of security and quality indicators.
3. Prepare functional requirements for the online system being developed in the study.
4. Develop an experiment plan to evaluate the impact of human work, AI, and code assessment tools on the system.
5. Conduct experiments to obtain system quality and security indicators in planned situations.
6. Analyse the results obtained, assessing how different practices affect the quality and security of the system code.
7. Prepare recommendations to improve code security and quality.

Keywords: cybersecurity, code quality, programming practices, DevOps, DevSecOps, Python Flask, vibe coding, static code analysis, dynamic code analysis.

ĮVADAS

Pagrindinės sąvokos. Pateikiame pagrindines sąvokas, kurios yra svarbos aprašant temos aktualumą, formuluojant problemą, tikslą ir uždavinius:

- **Kodo kokybė** – tai rodiklis, kuris parodo, kaip efektyviai programinis kodas atlieka savo funkciją, išlieka skaitomas, prižiūrimas, patvarus, saugus ir patikimas laikui bėgant.
- **Kodo saugumas** – tai kodo apsauga nuo neteisėtos prieigos ir pažeidžiamumų, kuriais galėtų pasinaudoti įsilaužėliai.

Temos aktualumas. Šiuolaikinėse programavimo praktikose (*gairių rinkinio, siekiant optimizuoti tam tikrus teigiamus aspektus programiniame kode ar jo kūrimo procese*) nuolat siekiama išlaikyti tinkamą balansą tarp saugumo ir kodo kokybės. Tačiau šį balansą pasiekti nėra visada lengva, nes saugumo priemonės neretai apsunkina kodo struktūrą ir veikimą, o tai gali neigiamai paveikti bendrą kodo kokybę. Be to, programavimo srityje vis dažniau naudojamos dirbtinio intelekto (DI) technologijos, kurioms pateikiamos saugumo ir kodo kokybės užduotys, o dėl to kyla papildomi klausimai, susiję su DI naudojimo kaip programavimo praktikos (taip pat žinomos kaip *Vibe Coding* (Cloudflare, 2025)) įtaka kodo saugumui ir kokybei.

Problema. Programuotojams nuolat tenka spręsti, kaip suderinti saugumo reikalavimus su kodo kokybės standartais. Neretai papildomos saugumo priemonės gali pabloginti kodo įskaitomumą, apsunkinti priežiūrą. Siekiant to išvengti, vis dažniau bandoma automatizuoti šiuos procesus naudojant specialius įrankius, kurie padeda taikyti saugumo praktikas be žmogaus klaidų ar kitų aspektų pablogėjimo. Vis dėlto programavimo praktikų automatizavimui ir taikymui reikia išsamios analizės, kad būtų išsiaiškinta, kaip šie įrankiai ir praktikos gali veikti kartu bei kokių kompromisų gali prireikti norint pasiekti optimalų rezultatą.

Darbo tikslas. Nustatyti ir įgyvendinti programavimo praktiką kuri veiksmingai suderina saugumo ir kodo kokybės aspektus, taip užtikrinant patvaraus, patikimo ir ilgaamžio kodo kūrimą.

Darbo uždaviniai:

1. Išanalizuoti literatūrą, siekiant geriau suprasti kibernetinės saugos ir kodo kokybės pusiausvyros problemas.
2. Sukurti įrankį, leidžiantį pateiktam *Python* projektui įvertinti saugos ir kokybės rodiklius.
3. Parengti tyrime kuriamai internetiniai sistemai keliamus funkcinius reikalavimus.
4. Sudaryti eksperimento planą, siekiant įvertinti žmogaus darbo, DI ir kodo įvertinimo įrankių taikymo įtaką sistemai.

5. Atlikti eksperimentus, suplanuotų situacijų metu gautų sistemų kokybės ir saugumo rodiklių išgavimui.
6. Atlikti gautų rezultatų analizę, įvertinant, kaip skirtingos praktikos paveikia sistemos kodo saugumą ir kokybę.
7. Parengti rekomendacijas, siekiant gerinti kodo saugumą ir kokybę.

1. LITERATŪROS APŽVALGA

Būtinybė rūpintis sistemos kokybe jos kūrimo metu buvo pastebėta jau seniai. 1982 metais pradėtos taikyti žmogaus vadovaujamos kodo peržiūros (Brykczynski ir Wheeler, 1993), kurios leido aptikti klaidas ir laikytis kodo kokybės normų, nors tuo metu reikalavimai sistemų kokybei nebuvo pakankamai apibrėžti.

Vėliau vystėsi ir kitos metodikos, tokios kaip kiekybiniai (skaitiniai) rodikliai, kurie papildė kodo peržiūrą su pamatuojamu matu. Jie leidžia nuolat automatizuotai stebėti kokybės ir saugumo rodiklius naudojant SPST (*Statinis Programų Saugumo Testavimas, angl. Static Application Security Testing*), DPST (*Dinaminis Programų Saugumo Testavimas, angl. Dynamic Application Security Testing*) ir kitus kodo statinės bei dinaminės analizės įrankius (Coleman ir kt., 1994).

SPST tyrimai atskleidžia automatinės analizės naudą pažeidžiamumų aptikimui, tačiau taip pat parodo ir kliūtis, kurios gali paskatinti programuotojus laikyti SPST rezultatus nereikšmingus ir taikyti neoptimalius sprendimus (Li ir kt., 2025*)*.

Lyginant su SPST, DPST įrankiai turi atvirkštinę problemą: jų efektyvumas yra santykinai žemas dėl prieigos prie sistemos programinio kodo ribotumo, todėl gali praleisti gilesnius pažeidimus, bei negali pamatuoti kodo kokybės (Dencheva, 2022).

Mažesnio masto eksperimentai taip pat patvirtina esamą kokybės ir saugumo kompromisą: tyrimas su *Python* programomis parodė, kad išsamūs saugumo kontroliniai sąrašai – gairių ir instrukcijų sąrašais, siekiančiais optimizuoti kodo saugumą per rekomendacijas, standartus ir reikalavimus – žymiai sumažina įprastus pažeidžiamumus, tačiau kartu ir padidina kodo kokybės klaidų skaičių bei kodo apimtį (***6**).

Kodo kokybės pedagogikos tyrimai rodo, kad ankstyvas praktikų mokymas sumažina riziką, jog vėlesnis saugumo stiprinimas sukurs žemos kokybės kodą; CS1 (įvadinio informatikos kurso, kurio metu mokoma pagrindinių programavimo sąvokų) lygmens mokymo priemonės suteikia žinių, padedančias formuoti patikimo ir patvaraus programinio kodo kūrimo mąstyseną ir saugių programavimo praktikų sampratą (Izu ir kt., 2025).

Programų kiekybinių rodiklių tyrimų rezultatai rodo, jog vieno matmens nepakanka pilnam sistemos kokybės ir saugumo įvertinimui (Coleman ir kt., 1994), pavyzdžiui: *Halstead* apimties ir darbo tipų įvertinimai matuoja programos dydį ir atliekamą darbą (Abdulkareem, 2021); ciklominis sudėtingumas matuoja programos srautų ir testavimo sudėtingumo įtaka (Filho, 2025); kognityvinis (pažintinis) sudėtingumas bando įvertinti žmogaus supratimo apkrovą (Filho, 2025); SPST įrankių išvestis matuoja aptiktų saugumo problemų kiekį programiniame kode (Dencheva, 2022); DPST įrankiai testuoja sistemą apsimesdami naudotojais aptikdami paviršutiniškus pažeidimus naudotojo pusėje. Kelių rodiklių tipų ir įrankių derinys leidžia pilniau atskleisti, kaip saugumo pakeitimai veikia kodo ilgaamžiškumą ir kūrėjų darbo sąnaudas (Dencheva, 2022).

Su šiuolaikiniu DI technologijų proveržiu išryškėja ir didėjanti pažangių DKM (*Didysis Kalbos Modelis, angl. Large Language Model – dirbtinio intelekto modelis, imituojantis žmogaus kalbą*) reikšmė kodo kūrimo procese. Palyginamieji DKM tyrimai rodo ženklius automatizacijos pranašumus sprendžiant problemas, tačiau šios sistemos reikalauja kruopščios žmogiškos priežiūros ir griežto vertinimo, kad būtų išvengta žemo kodo saugumo bei kokybės lygio (Kawchak, 2025).

Išnagrinėjus šaltinius galima išskirti penkis praktinius principus, kurios galima laikyti esminiais:

A. Derinti kalibruotą automatizuotą analizę su koncentruota žmogaus peržiūra;

B. Stebėti kelis rodiklius, o ne vieną;

C. Įtraukti pavyzdžiais pagrįstą instruktažą ir lengvai suprantamus kontrolinius sąrašus;

D. Skatinti praktikų mokinimą;

E. Atsakingai naudotis dirbinio intelekto priemonėmis.

Jei sistemos kūrėjai nesilaiko aukščiau pateiktų rekomendacijų ir neužtikrinę sistemos kokybės ją pateikia naudotojams, vėliau dažnai nebegrįžta prie to pačio kodo tobulinimo. Tai yra vadinama *techninе skola*. Nors tai leidžia pagreitinti sistemos išleidimą, tai didina riziką, kad nesužiūrėti sistemos trūkumai ir toliau kaupsis netaisomi, kol bus sunkiai ištaisomi (Cunningham, 1992).

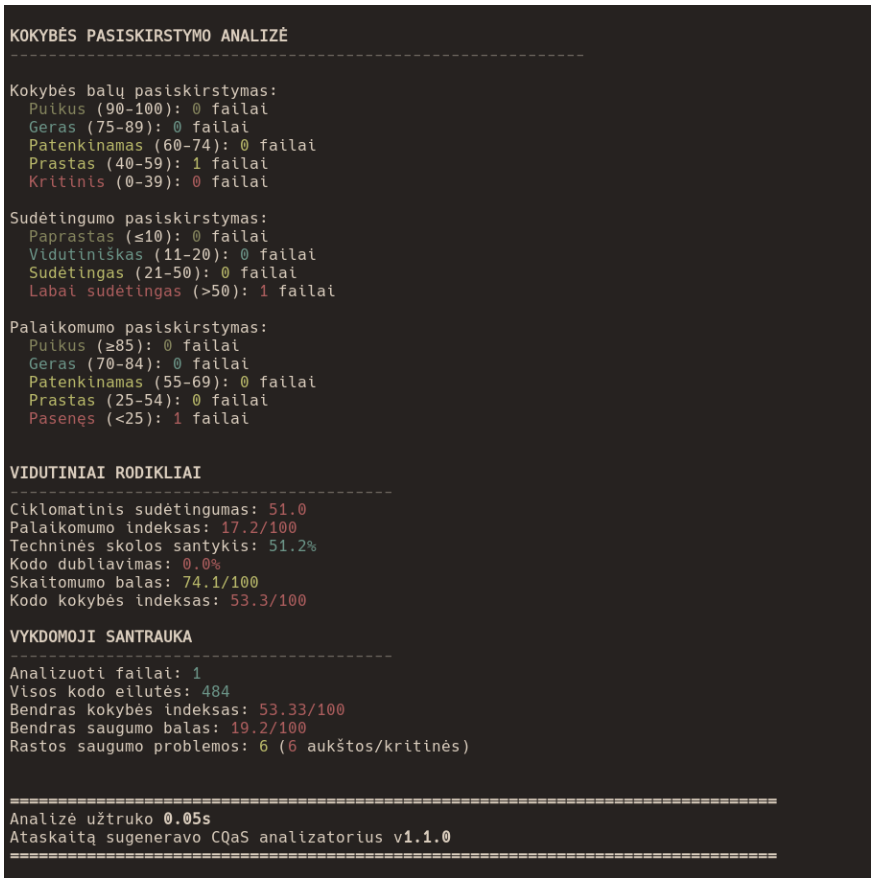
2. ĮRANKIO KŪRIMAS (*PIRMOJI VERSIJA*)

Po teorijos analizės šio darbo rengėjas sukūrė atviro kodo įrankį *QCaS* (*angl. Code Quality and Security*) pritaikant teorines žinias. Šis įrankis integruoja skirtingus kodo kokybės ir saugumo rodiklius:

1. Ciklominį ir kognityvinį (pažintinį) sudėtingumą.
2. PEP8 (van Rossum, 2001) stiliaus standarto atitikimą.
3. Kodo struktūrą, išskaidymą į modulius ir koncentraciją.
4. Saugumo klaidų aptikimą.

5. Patikimumo rodiklius.
6. Nenaudojamo kodo atradimą.
7. Apytikslę *techninę skolą*.

CQaS buvo sukurtas *Python* kalba, realizuojant visus numatytus programinio kodo kokybės ir saugumo vertinimo rodiklius. Nurodžius norimą projektą analizei, naudotojui pateikiama jo vertinimo ataskaita. 1 pav. vaizduojamas *CQaS* ataskaitos ištraukos pavyzdys.



1 pav. *CQaS projekto ataskaitos ištrauka*

Įrankis palaiko tiek lietuvių, tiek anglų kalbas, bei turi daug nustatymų, kurie padeda optimizuoti įrankio naudojimą.

Pagrindiniai *CQaS* rodikliai yra:

- **CS** – ciklominis sudėtingumas (kodo kokybės rodiklis, kuris matuoja nepriklausomų vykdymo kelių skaičių per programos šaltinio kodą).
- **PI** – patikimumo indeksas pagal Don Coleman (1994).
- **IL** – įskaitomumo lygis, apskaičiuotas pagal kitų rodiklių svertinę sumą.
- **KP** – kodo pasikartojimas.
- **SK** – aptiktos saugumo klaidos.

- **sK** – stiliaus klaidos pagal PEP8 standartą.
- **TS** – *techninės skolos* santykis.

3. REIKALAVIMAI TYRIMUI KURIAMOMS SISTEMOMS

Siekiant įgyvendinti tyrimo planą, buvo sukurta užduotis sukurti sistemą leidžiančią kurti mikrotinklaraščius. Sistema turėjo būti realizuota naudojant *Python* programavimo kalbą su *Flask* mikrobiblioteka ir privalėjo turėti šias funkcijas:

1. **Naudotojų autentifikacija:** prisijungimas naudojant el. pašto adresą ir slaptažodį.
2. **Įrašų valdymas:** naudotojai gali kurti įrašus *Markdown* žymėjimo kalba (Cone, 2020) formatu, o autoriai ir administratoriai – juos šalinti.
3. **Sekimo sistema:** naudotojai gali sekti vieni kitus.
4. **Naujienų puslapis:** naudotojai turi individualizuotą naujienų puslapį, sudarytą iš sekamų naudotojų įrašų.
5. **Paieška:** autentifikuotiems naudotojams leidžiama vykdyti paiešką visoje platformoje esančiuose įrašuose.
6. **Duomenų eksportas/importas:** naudotojams leidžiama eksportuoti ir importuoti ZIP archyvus su YAML failais, kuriuose pateikiami naudotojo duomenys ir visi jo įrašai.

4. TYRIMO ATLIKIMAS

Pagal sukurtos užduoties gaires iš viso buvo sukurtos 6 tipų sistemos versijos. 1 lentelėje pateikti šių sistemos tipų apibūdinimai, daliai jų nurodant kelias versijas. Iš viso eksperimentinei daliai buvo sukurta 12 sistemos versijų.

Eil. Nr.	Eksperimento tipas	Eksperimento versija	Versijos nr.	Vykdytojas
1.	Žemo saugumo ir žemos kokybės kodas	Pirminė projekto versija, sukurta negalvojant apie sistemos saugumą ar kokybę	1	žmogus
2.	Kodo tobulinimas	Žmogaus vykdomas kodo tobulinimas, savarankiškai tobulinant sistemą	2.1	žmogus
3.		DKM vykdomas kodo tobulinimas, nurodant pagerinti kodo kokybę ir saugumą	2.2	DKM
4.	Saugumo kontrolinių sąrašų taikymas	Žmogaus vykdomas kodo tobulinimas, remiantis pilnu kontroliniu sąrašu	3.1	žmogus
5.		DKM vykdomas kodo tobulinimas, nurodant remtis pilnu kontroliniu sąrašu	3.2	DKM
6.		Žmogaus vykdomas kodo tobulinimas, remiantis susiaurintu kontroliniu sąrašu	3.3	žmogus
7.		DKM vykdomas kodo tobulinimas, nurodant remiantis susiaurintu kontroliniu sąrašu	3.4	DKM

8.	Kodo saugos testavimas	Taikomas SPST (*Bandit, Pyright, Pycodestyle, Mypy, CQaS*) sistemos saugumui vertinti	4.1	žmogus
9.		Taikomas DPST (*OWASP Zed Attack Proxy (ZAP)*) sistemos saugumui vertinti	4.2	žmogus
10.	Kodo peržiūra	Žmogaus vykdoma programinio kodo peržiūra ir tobulinimas	5.1	žmogus
11.		DKM vykdoma programinio kodo peržiūra ir tobulinimas	5.2	DKM
12.	Sistemos dokumentacija	DKM vykdoma sistemos dokumentacija	6	DKM

1 lentelė. *Tyrime naudojamų situacijų apibendrinimas*

DKM vadovaujamoms užduotims buvo pasirinktas *ChatGPT-5.2-Thinking* modelis atsižvelgiant į jo samprotavimo gebėjimus ir siekiant suteikti DKM kuo didesnes galimybes pasiekti priimtinus rezultatus (OpenAI ir kt., 2025).

Galiausiai buvo atlikta kodo analizė pasitelkiant *Bandit, Pyright, Pycodestyle, Mypy* ir *CqaS* įrankius. Šie rezultatai buvo surinkti, sudėti į skaičiuoklės dokumentą ir išnagrinėjami pasitelkiant lyginamosios bei empirinės analizės metodus.

4.1. REZULTATAI

2 lentelėje pateikti apibendrinti visų 12 sistemos versijų kokybės ir saugos rodiklių rezultatai.

#	Versijos nr.	Pagr. Rodiklis	Bandit (ž/v/a)	Pyright (k/į)	Pycodestyle (k/į)	Mypy (k)	CQaS (CS/PI/IL/KP/SK/sK/TS)
1	1	visi	7/3/2	264/1	0/0	6	51.0/17.2/74.1/0.0/6/19/51.2
2	2.1	kokybė	7/2/2	237/1	0/0	1	5.1/70.0/84.4/0.0/5/3/4.4
3	2.2	kokybė	7/3/2	288/1	2/6	10	9.3/48.9/70.9/0.0/6/20/9.3
4	3.1	visi	6/4/0	262/0	0/0	0	58.0/12.9/74.8/2.3/4/19/46.5
5	3.2	visi	2/1/0	348/2	28/1	13	80.0/17.8/70.5/1.2/0/32/31.4
6	3.3	visi	6/4/0	262/0	0/0	0	58.0/12.9/74.8/2.3/4/19/46.5
7	3.4	visi	3/0/0	311/1	4/1	7	72.0/17.8/68.3/4.6/1/34/38.3
8	4.1	saugumas	6/0/0	2/0	0/0	1	52.0/16.8/74.1/0.0/3/18/52.6
9	4.2	saugumas	7/3/2	264/1	0/0	6	51.0/16.9/74.1/1.4/6/19/40.7
10	5.1	visi	6/2/0	239/1	0/0	0	5.5/69.2/84.2/0.0/5/3/4.5
11	5.2	visi	6/1/0	290/1	0/1	8	69.0/15.2/71.7/6.5/1/26/40.2
12	6	kokybė	7/3/2	264/1	0/1	6	51.0/13.2/85.6/0.9/6/14/51.9

2 lentelė. *Tyrimui naudotų sistemos versijų saugos ir kokybės testavimo rezultatai*

2 lentelėje duomenys yra pateikti šiuo formatu:

- **Bandit (ž/v/a)** – *Bandit* **ž**emo/**v**idutinio/**a**ukšto lygio klaidų kiekis.
- **Pyright (k/į) ir Pycodestyle (k/į) –** *Pyright* ir *Pycodestyle* **k**laidų/**į**spėjimų kiekis.
- **Mypy (k) –** *Mypy* **k**laidų kiekis.
- **CQaS (CS/PI/IL/KP/SK/sK/TS)** – *CQaS* **c**iklominis **s**udėtingumas/**p**atikimumo**i** ndeksas/**į**skaitomumo **l**ygis/**k**odo **p**asikartojimo santykis (%)/**s**augumo **k**laidos/**s**tiliaus

klaidos/**t**echninės **s**kolos santykis (%). (***Pastebėkite: S****K*** = ***saugumo*** *klaidos,* ***s****K* = ***stiliaus*** *klaidos*)

4.2. REZULTATŲ ANALIZĖ

Buvo pradėta nuo rezultatų normalizavimo; pasitelkta *Python* programavimo kalba (su *Pandas*, *Matplotlib* ir *Seaborn* bibliotekomis) ir *min-max normalizavimo* formulė:

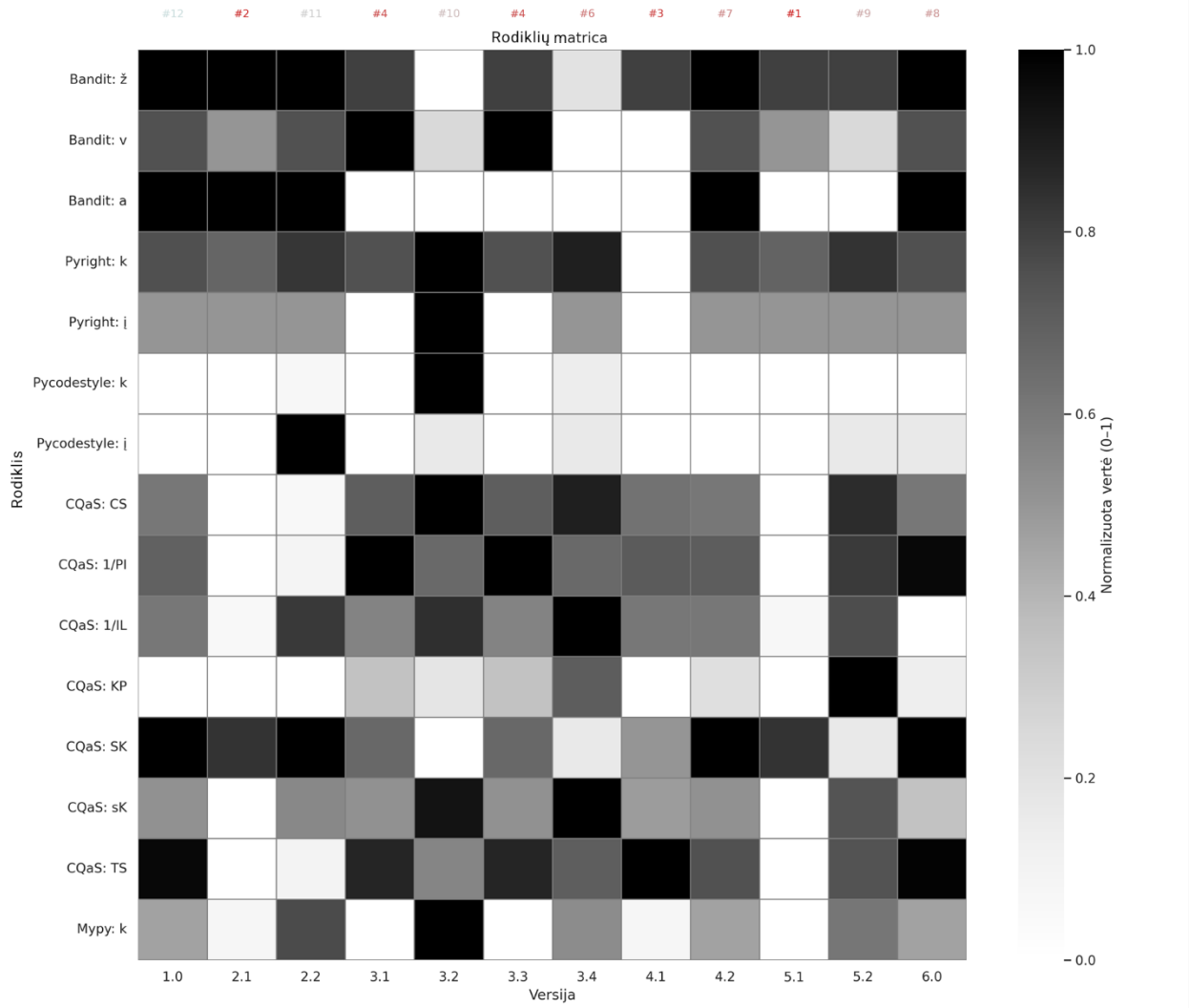
$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

Sunormalizavus duomenis, visi stulpeliai buvo reitinguoti pagal triskaičio vidurkio (*angl. Trimean, pagal: Čekanavičius, 2001*) formulę:

$$\frac{Q_1 + 2Q_2 + Q_3}{4}$$

Šio vidurkio tipas buvo pasirinktas todėl, kad jis subalansuoja skirtingus kodo kokybės ir saugumo rodiklius bei yra atsparus ekstremalioms reikšmėms duomenų imtyje (Tukey, 1977).

Visa rezultatų informacija buvo išreikšta matricos grafiku, kuris pateiktas 2 pav.



2 pav. *Normalizuota reitinguota rodiklių matrica*

Grafike matome matricą, kurioje eilutės atitinka skirtingus rodiklius, o stulpeliai – sistemos versijas. Kiekvieno langelio spalva rodo normalizuotą rodiklio reikšmę. Virš kiekvieno stulpelio pateiktas versijos reitingas, apskaičiuotas pagal tos versijos visų rodiklių normalizuotų reikšmių triskaitį vidurkį, kur mažesnė vertė reiškia „optimalesnę“ praktiką. *CQaS PI* ir *IL* rodiklių eilutėms vaizduojama atvirkštinė rodiklio reikšmė, nes norime optimizuoti jų didėjimą, o ne mažėjimą.

2 pav. pateiktoje matricoje matome, jog **efektyviausios** praktikos buvo:

I. **5.1 versija (žmogaus vadovaujama kodo peržiūra) – 1 vieta.**
II. **2.1 versija (žmogaus vadovaujamas išskaidymas) – 2 vieta.**
III. **4.1 versija (SPST) – 3 vieta.**
IV. 3.1 versija (žmogaus vadovaujamas plataus saugumo kontrolinio sąrašo taikymas) – 4 vieta.

V. 3.3 versija (žmogaus vadovaujamas siauro saugumo kontrolinio sąrašo taikymas) – 4 vieta (panašus efektas į 3.1).
VI. 3.4 versija (DKM vadovaujamas siauro saugumo kontrolinio sąrašo taikymas) – 6 vieta.

Neefektyviausios praktikos buvo:

I. 1 versija (žemo saugumo ir žemos kokybės kodas) – 12 vieta.
II. 2.2 versija (DKM vadovaujama išskaidymas) – 11 vieta.
III. 3.2 versija (DKM vadovaujamas plataus saugumo kontrolinio sąrašo taikymas) – 10 vieta.
IV. 5.2 versija (DKM vadovaujama kodo peržiūra) – 9 vieta.
V. 6 versija (DKM vadovaujama dokumentacija) – 8 vieta.
VI. 4.2 versija (DPST) – 7 vieta.

Remiantis rezultatais galime daryti šias prielaidas:

A. Žmogaus vadovaujamos praktikos yra efektyviausios.
B. DKM taikymas neretai pablogina kodo kokybę ir saugumą.
C. Žmogaus vadovaujama kodo peržiūra ir išskaidymas yra tarp geriausių praktikų.
D. Saugumo kontroliniai sąrašai turi didelį poveikį, tačiau jų efektyvumas priklauso nuo apimties ir pritaikymo metodikos.
E. DPST metodai buvo neefektyvūs; SPST – tarp efektyviausių.
F. Dokumentacijos pridėjimas neturi didelio poveikio kodo kokybei ir saugumui trumpalaikiu.

Po kiekybinių rodiklių analizės buvo atlikta žmogaus vadovaujama kodo peržiūra. Peržiūros rezultatai parodė įdomius individualių praktikų aspektus, kurie buvo praleisti automatizuotoje analizėje:

Pirmosios (1) versijos analizė atskleidė rimtas problemas, kurios kėlė grėsmę sistemos saugumui, jos ilgaamžiškumui ir patikimumui. Joje buvo naudojama nesaugi *MD5* slaptažodžių maišos funkcija (*kriptografinė maišos funkcija (tai – matematinis algoritmas, kuris atvaizduoja bet kokio dydžio įvesties duomenis į fiksuoto ilgio baitų seką), kuri yra laikoma nepatikima saugumo požiūriu; Black ir kt., 2006*), neįgyvendinta apsauga nuo tarpinių užklausų klastojimo atakų (*CSRF, angl. Cross-Site Request Forgery*), trūko dokumentacijos. Taip pat aptikti *SQL (angl. Structured Query Language, duomenų bazių užklausų kalba)* injekcijos (*leidžiantys įvykdyti neautorizuotas duomenų bazės užklausas*) bei XSS (*angl. Cross-Site Scripting, leidžiantys vykdyti kenksmingą kodą naudotojo naršyklėje*)

pažeidžiamumai. Rasta daug kitų rimtų klaidų, kurios tapo pagrindu tolesniam saugumo ir kodo kokybės praktikų tyrimui. 3 pav. pateikti keli pirmosios versijos pažeidžiamumų pavyzdžiai.

3 pav. *Pirmosios versijos nesaugumų pavyzdžiai*

2.1 versijoje buvo pritaikyti išskaidymo į modulius patobulinimai. Kodo išskaidymas pagerino sistemos tvarkingumą bei žymiai sumažino *techninės skolos* santykį, bet saugumo spragos vis dar išliko. Tačiau 2.2 versijoje įdiegtas DKM vadovaujamas išskaidymas į modulius buvo antroji blogiausia praktika pagal rodiklių analizę, nes DKM metodai nesugeba susitvarkyti su pilnu projekto kontekstu.

Trečioje (3) versijoje pastebėta, kad saugumo kontroliniai sąrašai žymiai pagerino kodo saugumą ir kokybę, tačiau ir ši versija vis tiek turėjo trūkumų, išliko nežymių saugumo ir ryškių kokybės spagų.

Ketvirtoje (4) versijoje buvo taikomos SPST ir DPST praktikos. Rezultatai tapo geresni. SPST buvo pastebimai patobulinta slaptažodžių sauga, įdiegta klaidų valdymo sistema, išsamiau tikrinama duomenų įvestis ir pasiekti geresni rodikliai. DPST įrankiai turėjo minimalų poveikį kodo kokybei ir saugumui.

Penktoje (5) versijoje žmogaus vadovaujama kodo peržiūra (5.1 versija) ženkliai pagerino kodo saugumą ir kokybę; DKM vadovaujama kodo peržiūra (5.2 versija) ir vėlesnė dokumentacija (šeštoji (6) versija) nepasiekė reikšmingų patobulinimų.

Remiantis peržiūros atradimais galime daryti šias prielaidas:

A. DKM vadovaujami metodai, nors ir padeda pasiekti tam tikrą efektyvumo lygį, negali visiškai užtikrinti kodo saugumo ir kokybės, nes nesugeba tinkamai įvertinti projekto konteksto ir detalių.
B. Žmogaus vadovaujama kodo peržiūra yra viena iš efektyviausių praktikų tiek kodo kokybės, tiek saugumo užtikrinimo atžvilgiu.
C. Išskaidymas į modulius gerina kodo struktūrą ir mažina *techninę skolą*, tačiau nežymiai gerina kodo saugumą.
D. Saugumo kontroliniai sąrašai gali labai pagerinti kodo kokybę ir saugumą, tačiau jų taikymas turi būti nuoseklus ir išsamus.
E. Kodo kokybė ir saugumas yra suderinami tik tada, kai abu aspektai yra tolygiai integruoti į kūrimo procesą ir sunkiai gali būti laikomi atskirais tikslais.
F. SPST padeda aptikti pažeidžiamumus bei klaidas, bet jų praktinis efektas kodui ne visada atitinka teoriją; DPST metodika yra nežymiai efektyvi.
G. Per didelis dėmesys automatizuotoms sistemoms gali pabloginti kodo kokybę.

5. REZULTATAS

5.1. PRAKTIKOS SUDARYMAS

Remiantis atlikto tyrimo rezultatais, geriausios praktikos, užtikrinančios optimalią kodo kokybę ir saugumą, yra šios:

A. **Žmogaus vadovaujama kodo peržiūra:** šis metodas pasirodė esąs efektyviausias, nes žmogiškas įsikišimas leidžia geriau atpažinti subtilius kodo trūkumus.

B. **Saugumo kontrolinių sąrašų taikymas:** viena iš veiksmingiausių priemonių užtikrinti kodo saugumą ir kokybę, nes jie sistemingai tikrina kritinius saugumo aspektus.

C. **Išskaidymas į modulius:** gerina kodo struktūrą ir sumažina techninę skolą, kas padeda ilgainiui lengviau vystyti ir palaikyti sistemą.

D. **SPST:** pasirodė kaip efektyviausia automatizuota programavimo praktika saugumo ir kodo kokybės gerinimui.

E. **Įvesties tvarkymas:** svarbu užtikrinti, kad visi įvesti duomenys būtų patikrinti ir teisingi prieš juos apdorojant.

F. **Kriptografinės priemonės:** nesaugios kriptografinės priemonės pražlugdo mūsų sistemos saugumą, todėl būtina užtikrinti, kad visi naudojami kriptografiniai resursai yra saugūs.

G. **Kodo stiliaus standartai:** kad užtikrintume kodo kokybę, būtina įrėminti visą kodą į tą patį stiliaus standartą.

Šios praktikos buvo įgyvendintos taikant interaktyvios statinės kodo analizės principus. CQaS įrankis buvo patobulintas pridedant du veikimo režimus:

- **Testavimo režimas** atlieka visas automatizuotas statines kodo analizės užduotis. Šiame režime atkreipiamas dėmesys į standartus, išskaidymą į moduliu, įvairius kiekybinius matavimus ir statinį saugos tikrinimą.
- **Peržiūros režimas** atlieka tas pačias testavimo užduotis, tačiau papildomai identifikuoja kodo dalis, kurios vertos peržiūros ir užduoda peržiūrėtojui (*CqaS* naudotojui) klausimą: „*Ar tai atrodo teisingai?*".

Šis derinys leidžia efektyviai taikyti geriausias praktikas, subalansuojant kodo kokybę, saugumą bei viso kodo visuotinės kokybės užtikrinimo lengvumą.

5.2. ĮRANKIO TOBULINIMAS

Pirmoji *CQaS* versija tyrė išskaidymą, pritaikė SPST metodus, testavo kodo saugumą bei kriptografiją, dalinai užtikrino PEP8 stilių. Todėl *CQaS* 2.0.0 vystymui buvo:

1. Pridėtas *-r (--review) CQaS* nustatymas, kuris įjungia peržiūros režimą.
2. Šiek tiek praplėstos *CQaS* saugumo taisyklės pagal saugumo kontrolinį sąrašą pagal „*Assessing Vulnerability of Students' Programming Projects: Application of Testing Tools and Estimation of Checklist Effect on Code Quality“ (2025)*.
3. Pridėtos taisyklės, įskaičiuojančias duomenų tipų apsaugą į kodo kokybes rodiklius.

Toliau (4 pav.) pateikiamas peržiūros režimo pavyzdys.

4 pav. *CqaS 2.0.0 peržiūros režimo pavyzdys*

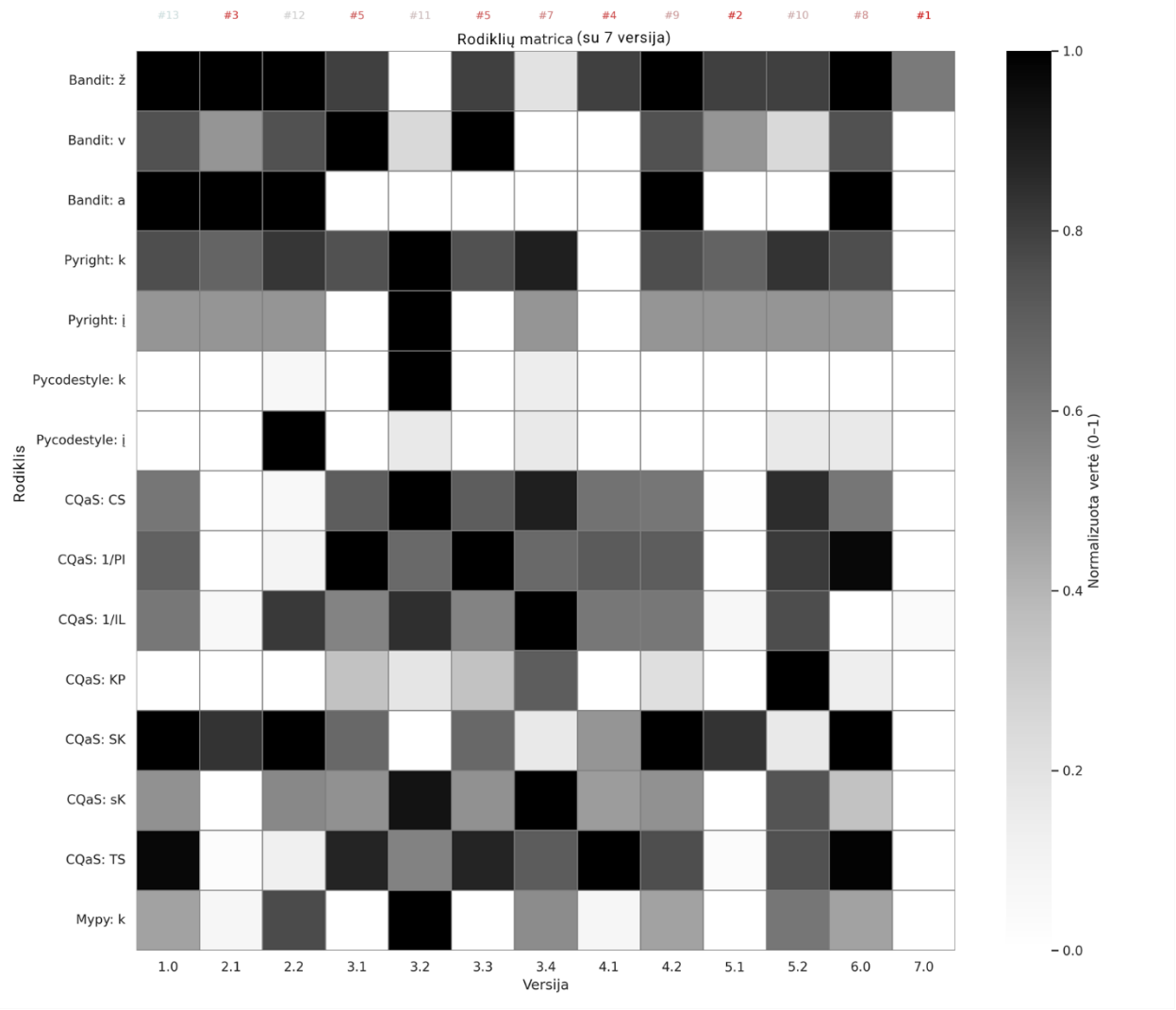
5.3. PRAKTIKOS PRITAIKYMAS

Po *CQaS* tobulinimo buvo pritaikyta galutinė praktika ir įrankis: sukurta 7 sistemos versija. Šios versijos analizės rezultatai yra pateikti 3 lentelėje.

Versija	Vykdytojas	Pagr. rodiklis	**Bandit (ž/v/a)**	**Pyright (k/į)**	**Pycodestyle (k/į)**	**Mypy (k)**	**CQaS (CS/PI/IL/KP/SK/sK/TS)**
7	**Žmogus**	visi	5/0/0	0/0	0/0	0	4.8/67.7/84.7/0.0/0/3/2.9

3 lentelė. *Septintos (7) versijos rezultatai*

Šie rezultatai buvo pridėti prie rezultatų matricos tolesnei analizei. 5 pav. pateikta galutinė rodiklių matrica su septintąją (7) versija palyginimui su kitomis praktikomis.



5 pav. *Galutinė normalizuota rodiklių matrica su 7 versija*

Rodikliai rodo, jog 7 versija su nustatyta praktika veiksmingai pasiekia optimalius kodo kokybės ir saugumo rodiklius, juos efektyviai subalansuodama.

Parašius *Python* programą duomenų pokyčio analizei matome, kad visi rodikliai žymiai pagerėjo. Toliau sąraše pateikiama rodiklių pokyčio analizė – žemesnis (labiau neigiamas) procentas reiškia optimalesnius rodiklius.

- *Bandit*: -73,33% – saugumo klaidos.
  - Žemo lygio: -20%
  - Vidutinio lygio: -100%
  - Aukšto lygio: -100%
- *Pyright*: -100% – kodo kokybė.
  - Klaidos: -100%
  - Įspėjimai: -100%

- *Pycodestyle*: -100% – kodo stilius.
  - Klaidos: -100%
  - Įspėjimai: -100%
- *CQaS*: -79,22% – teoriniai rodikliai (saugumas ir kokybė).
  - *CS*: -90,6%
  - 1/*PI*: -74,62%
  - 1/*IL*: -12,50%
  - *KP*: -100,00%
  - *SK*: -100,00%
  - *sK*: -84,31%
  - *TS*: -92,49%
- *Mypy*: -100% – duomenų tipų užtikrinimas.
  - Klaidos: -100%

APIBENDRINIMAS

Atliktas tyrimas ir analizė parodė, kad efektyviausios praktikos kodo kokybei ir saugumui užtikrinti yra kodo peržiūra, išskaidymas į modulius, saugumo kontroliniai sąrašai ir SPST. Sukurtas įrankis *CQaS*, integruojantis šias praktikas, pasiekė optimalius rezultatus pritaikant sudarytą praktiką. Dirbtinio intelekto poveikis kodui buvo gana ribotas, nes DI įrankiai nesugebėjo įvertinti pilno projekto ir reikalavimų konteksto. Darbo tikslas buvo įgyvendintas, sudarant veiksmingą praktiką, pasiekiant optimalų kodo kokybės ir saugumo derinį bei sukuriant įrankį jos automatiniam taikymui.

REKOMENDACIJOS

Remiantis tyrimo rezultatais bei teorinėmis žiniomis, norint kurti patikimą, kokybišką ir saugų kodą, reikia vadovautis šiais principais:

1. **Kodo peržiūra:** reikia taikyti sistemiškas kodo peržiūras, siekiant užtikrinti kodo saugumą ir kokybę visame projekto kontekste.
2. **Vengti dirbtinio intelekto priemonių:** duomenys rodo, jog dirbtinio intelekto priemonės tik pablogina kodo saugumą ir kokybė su mažai naudos. Rekomenduojama vengti DI įrankių.
3. **Saugumo kontrolinių sąrašų taikymas:** saugumo kontroliniai sąrašai žymiai pagerino kodo kokybė, nes jie skatina sistemišką kodo peržiūrą. Tai leidžia užtikrinti patikimą bei tvarų programinį kodą.
4. **Išskaidymas į modulius:** siekiant aukštos kodo kokybės yra rekomenduojama išskaidyti programinį kodą į nepriklausomus vienetus, siekiant sumažinti *techninės skolos* santykį bei išvengiant pasikartojančio kodo ir klaidų.
5. **SPST priemonės:** taikant SPST, galima optimizuoti kodo kokybę ir saugumą prižiūrint juos automatizuotu būdu ir išvengiant kokybės ar saugumo lygio nukritimo.
6. **Kodo stiliaus standartai:** taikant vieną kodo standartą galima pasiekti vientisos, tvarkingos ir kokybiškos kodo bazės.
7. **Gynybinis programavimas:** reikia tvarkyti ir tikrinti duomenis bei klaidas, siekiant išvengti potencialių pažeidžiamumų dėl įvesties tvarkymo nepakankamumo.
8. **Pedagogika:** kiti tyrimai rodo, jog saugumo ir kodo kokybės principų mokinimasis leidžia išvengti klaidų, kurios gali paveikti kodo kokybę ar saugumą. Be to, švietimas leidžia visada turėti naujausia informaciją, kas leidžia išvengti įvairių klaidų, tokių kaip nesaugių kriptografinių priemonių naudojimo.

INFORMACIJOS ŠALTINIAI

1. Čekanavičius, V. ir Murauskas, G. (2001). *Statistika ir jos taikymai.* Vilnius: Leidykla TEV, 39 p. ISBN 9986-546-93-1.
2. Tukey, J. W. (1977). Exploratory Data Analysis. Addison-Wesley. ISBN 0-201-07616-0.
3. Abdulkareem, S. A. ir Abboud, A. J. (2021). *Evaluating Python, C++, JavaScript and Java Programming Languages Based on Software Complexity Calculator (Halstead Metrics)*. IOP Conference Series: Materials Science and Engineering. https://doi.org/10.1088/1757-899X/1076/1/012046 [žiūrėta 2026-01-15]
4. Anderson, P. ir Ragab, M. (2021). *Bandit: A Python AST-Visitor for Finding Common Security Issues. PyCQA.* https://github.com/PyCQA/bandit [žiūrėta 2026-01-15]
5. (*) https://git.ari.lt/ari/cqas [žiūrėta 2026-01-15]
6. (*not exactly 1:1 but it'll do) https://git.ari.lt/ari/research-school-2024/src/branch/main/writeups/article.md [žiūrėta 2026-01-15]
7. Black, J., Cochran, M. ir Highland, T. (2006). *A Study of the MD5 Attacks: Insights and Improvements.* https://doi.org/10.1007/11799313_17 [žiūrėta 2026-01-15]
8. Brykczynski, B. ir Wheeler, D. A. (1993). *An annotated bibliography on software inspections. SIGSOFT Software Engineering Notes*, 18(1), 81-88. https://doi.org/10.1145/157397.157411 [žiūrėta 2026-01-15]
9. Cloudflare (2025). *Cloudflare learning: What is vibe coding?*. https://www.cloudflare.com/en-gb/learning/ai/ai-vibe-coding/ [žiūrėta 2026-01-15]
10. Coleman, D., Ash, D., Lowther, B. ir Oman, P. (1994). *Using Metrics to Evaluate Software System Maintainability.* Computer, 27, 44-49. https://doi.org/10.1109/2.303623 [žiūrėta 2026-01-15]
11. Cone, M. (2020). *The Markdown Guide.* https://www.markdownguide.org/book / ISBN-13. 979-8656504492. [žiūrėta 2026-01-15]
12. Cunningham, W. (1992). *The WyCash portfolio management system.* SIGPLAN OOPS Messenger, 4(2), 29-30. https://doi.org/10.1145/157710.157715 [žiūrėta 2026-01-15]
13. Dencheva, L. (2022*). Comparative analysis of Static application security testing (SAST) and Dynamic application security testing (DAST) by using open-source web application penetration testing tools.* Magistro darbas, National College of Ireland, Dublin. https://norma.ncirl.ie/5956/1/lyubkadencheva.pdf [žiūrėta 2026-01-15]
14. *Ekonomikos ir kalbos žinių enciklopedija.* Prieiga per: http://www.ims.mii.lt/EK%C5%BD/enciklo.html [žiūrėta 2026-01-15]
15. Grinberg, M. ir kt. (2010). *Flask: A Web Microframework for Python. Pallets Projects.* https://flask.palletsprojects.com/ [žiūrėta 2026-01-15]

16. Izu, C., Mirolo, C., Börstler, J., Connamacher, H., Crosby, R., Glassey, R., Haldeman, G., Kiljunen, O., Kumar, A. N., Liu, D., Luxton-Reilly, A., Matsumoto, S., de Oliveira, E. C., Russell, S. ir Shah, A. (2025). *Introducing Code Quality at CS1 Level: Examples and Activities. 2024 Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE 2024).* ACM, 339-377. https://doi.org/10.1145/3689187.3709615 [žiūrėta 2026-01-15]
17. Kawchak, K. (2025). *Code Generation Competition: 16 Proprietary vs. Open-Source LLMs & Iterative Learning Based on FDA Adverse Event Reporting System.* Zenodo. https://doi.org/10.5281/zenodo.18029100 [žiūrėta 2026-01-15]
18. Li, Y., Yao, P., Yu, K., Wang, C., Ye, Y., Li, S., Luo, M., Liu, Y. ir Ren, K. (2025). *Understanding Industry Perspectives of Static Application Security Testing (SAST) Evaluation. Proceedings of the ACM on Software Engineering*, 2(FSE), straipsnis FSE134, 24 p. https://doi.org/10.1145/3729404 [žiūrėta 2026-01-15]
19. Luciani, J. ir kt. (2017*). Pycodestyle: Python Style Guide Checker (formerly PEP8).* PyCQA. https://github.com/PyCQA/pycodestyle [žiūrėta 2026-01-15]
20. MagicStack ir kt. (2012). *Mypy: Optional Static Typing for Python. Python Software Foundation.* https://github.com/python/mypy [žiūrėta 2026-01-15]
21. Microsoft (2020). *Pyright: Fast, Featured Static Type Checker for Python.* Microsoft Corporation. https://github.com/microsoft/pyright [žiūrėta 2026-01-15]
22. OWASP Foundation ir kt. (2010). *OWASP Zed Attack Proxy (ZAP): Open-Source Web Application Security Scanner.* OWASP Foundation. https://www.zaproxy.org/ [žiūrėta 2026-01-15]
23. OpenAI ir kt. (2025). *GPT-5.2: Technical Report on Advanced Reasoning Capabilities with Thinking Mode. OpenAI Corporation.* https://openai.com/index/introducing-gpt-5-2/ [žiūrėta 2026-01-15]
24. Sabbag Filho, N. (2025). *Comparative Analysis between Cognitive Complexity and Cyclomatic Complexity in Software Development.* Leaders Tec Br, 2(7), 1-4. Zenodo. https://doi.org/10.5281/zenodo.14879076 [žiūrėta 2026-01-15]
25. van Rossum, G., Warsaw, B. ir Coghlan, A. (2001*). PEP 8 - Style Guide for Python Code. Python Enhancement Proposals.* https://peps.python.org/pep-0008/ [žiūrėta 2026-01-15]

PRIEDAI

1 PRIEDAS. BRANDOS DARBO IR JO APRAŠO AUTENTIŠKUMO PATVIRTINIMAS (*)

BRANDOS DARBO IR JO APRAŠO AUTENTIŠKUMO PATVIRTINIMAS

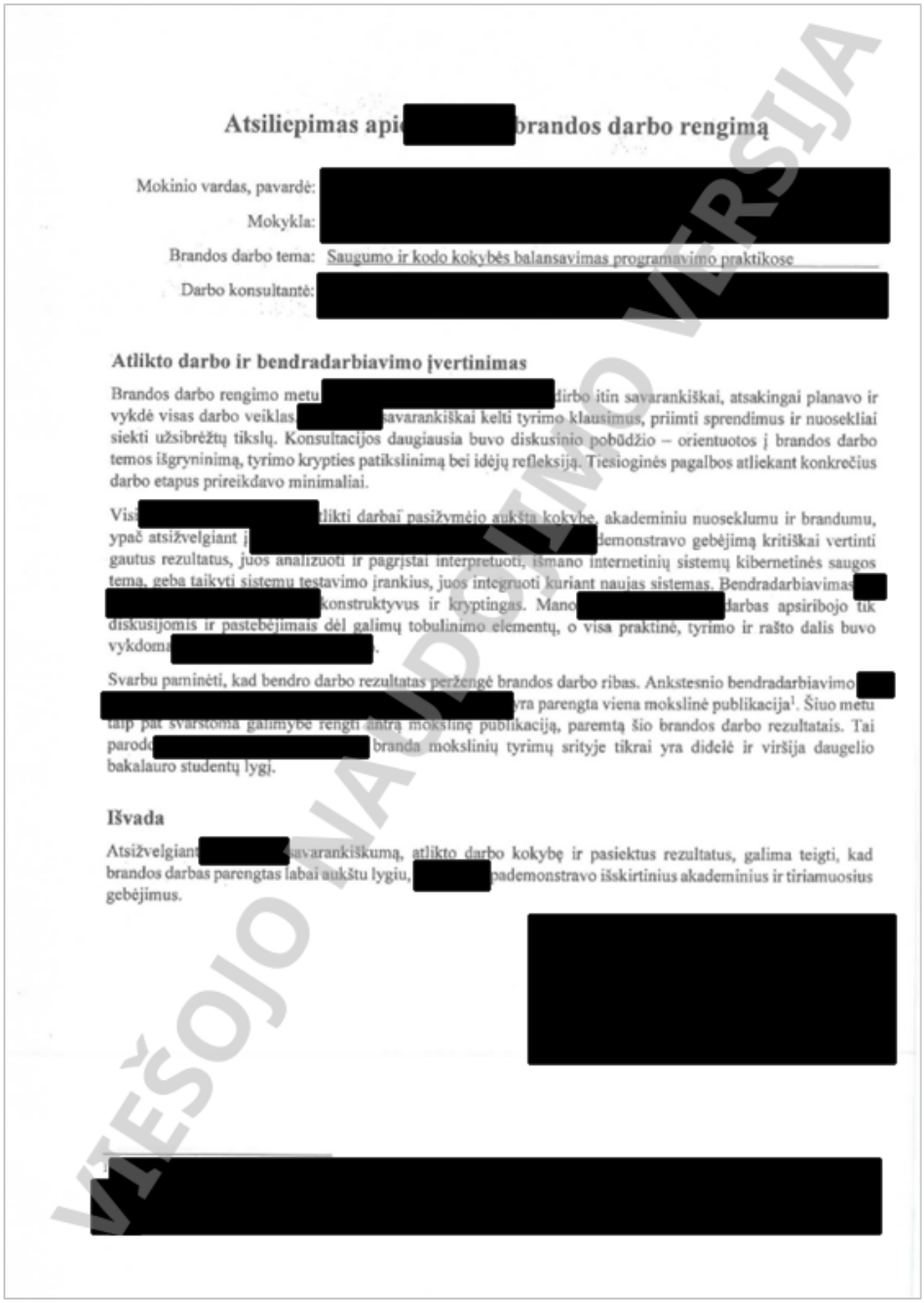
2026 m. vasario 3 d.

[GYMNASIUM NAME]

Aš, [pilnas teisinis vardas], patvirtinu, kad pats (-i) atlikau savo brandos darbą **„Saugumo ir kodo kokybės balansavimas programavimo praktikose“** ir parengiau jo aprašą, nepažeidžiau kitų autorių teisių cituodamas (-a) ar kitaip panaudodamas (-a) jų kūrinius.

[parašas]	[pilnas teisinis vardas]
(parašas)	(vardas ir pavardė)

2 PRIEDAS. KONSULTANTO (-ĖS) [VARDAS] ATSILIEPIMAS APIE MOKINIO BRANDOS DARBO RENGIMĄ (*)



Atsiliepimas api[redacted]brandos darbo rengimą

Mokinio vardas, pavardė: [redacted]

Mokykla: [redacted]

Brandos darbo tema: Saugumo ir kodo kokybės balansavimas programavimo praktikose

Darbo konsultantė: [redacted]

Atlikto darbo ir bendradarbiavimo įvertinimas

Brandos darbo rengimo metu [redacted] dirbo itin savarankiškai, atsakingai planavo ir vykdė visas darbo veiklas [redacted] savarankiškai kelti tyrimo klausimus, priimti sprendimus ir nuosekliai siekti užsibrėžtų tikslų. Konsultacijos daugiausia buvo diskusinio pobūdžio – orientuotos į brandos darbo temos išgryninimą, tyrimo krypties patikslinimą bei idėjų refleksiją. Tiesioginės pagalbos atliekant konkrečius darbo etapus prireikdavo minimaliai.

Visi [redacted] atlikti darbai pasižymėjo aukšta kokybe, akademiniu nuoseklumu ir brandumu, ypač atsižvelgiant į [redacted] demonstravo gebėjimą kritiškai vertinti gautus rezultatus, juos analizuoti ir pagrįstai interpretuoti, išmano internetinių sistemų kibernetinės saugos temą, geba taikyti sistemų testavimo įrankius, juos integruoti kuriant naujas sistemas. Bendradarbiavimas [redacted] konstruktyvus ir kryptingas. Manc [redacted] darbas apsiribojo tik diskusijomis ir pastebėjimais dėl galimų tobulinimo elementų, o visa praktinė, tyrimo ir rašto dalis buvo vykdoma [redacted].

Svarbu paminėti, kad bendro darbo rezultatas peržengė brandos darbo ribas. Ankstesnio bendradarbiavimo [redacted] yra parengta viena mokslinė publikacija[1]. Šiuo metu taip pat svarstoma galimybė rengti antrą mokslinę publikaciją, paremtą šio brandos darbo rezultatais. Tai parod[redacted] branda mokslinių tyrimų srityje tikrai yra didelė ir viršija daugelio bakalauro studentų lygį.

Išvada

Atsižvelgian[redacted] savarankiškumą, atlikto darbo kokybę ir pasiektus rezultatus, galima teigti, kad brandos darbas parengtas labai aukštu lygiu, [redacted] pademonstravo išskirtinius akademinius ir tiriamuosius gebėjimus.

[redacted]

[redacted]

3 PRIEDAS. KODO FRAGMENTŲ ILIUSTRACIJOS

```
@app.post("/import-data")
@login_required
def import_data():
    f = flask.request.files.get("file")
    if not f:
        flask.abort(400)

    try:
        z = zipfile.ZipFile(f.stream)
    except Exception:
        flask.abort(400)

    try:
        udata = yaml.load(z.read("user.yaml"), Loader=yaml.UnsafeLoader)
    except Exception:
        udata = {}

    for k, v in (udata or {}).items():
        if k in ("following", "followers"):
            continue
        try:
            setattr(current_user, k, v)
        except Exception:
            pass

    db.session.commit()

    for fid in (udata or {}).get("following", []):
```

6 pav. *Pirmos (1) versijos duomenų importo funkcija – nekokybiškas kodas*

```
ari@ari-gentoo (main) src % find . -not -name '*.pyc' -type f
./templates/forms.j2
./templates/home/index.j2
./templates/base.j2
./templates/auth/index.j2
./templates/dashboard/feed.j2
./templates/dashboard/search.j2
./templates/dashboard/user.j2
./templates/dashboard/import.j2
./mblog/db.py
./mblog/models/post.py
./mblog/models/followers.py
./mblog/models/__init__.py
./mblog/models/user.py
./mblog/views/auth.py
./mblog/views/__init__.py
./mblog/views/home.py
./mblog/views/dashboard.py
./mblog/forms/auth.py
./mblog/forms/__init__.py
./mblog/forms/dashboard.py
./mblog/__init__.py
./app.py
./instance/db.sqlite
```

7 pav. *Antros (2) versijos išskaidyto projekto pavyzdys*

```
 1 app.py
18 #!/usr/bin/env python3
17 # -*- coding: utf-8 -*-
16 """Microblog App"""
15
14 import flask
13
12 from mblog import create_app
11
10 app: flask.Flask = create_app(__name__)
 9
 8
 7 def main() -> int:
 6     """entry / main function"""
 5
 4     app.run("127.0.0.1", 8080, True)
 3
 2     return 0
 1
19
 1 if __name__ == "__main__":
 2     raise SystemExit(main())
```

8 pav. *Trumpa ir tvarkinga išskaidyto projekto pagrindinė („main“) funkcija*

```python
@auth.post("/")
def authenticate() -> t.Any:
    """Authenticate a user"""

    form: AuthForm = AuthForm()

    if not form.validate_on_submit():  # type: ignore
        flask.flash("Invalid form data", "error")
        return render_auth_home(form), 400

    # Validate that all data is present
    if not (form.email and form.email.data and form.password and form.password.data):
        flask.flash("No/invalid email or password", "error")
        return render_auth_home(form), 400

    email: str = form.email.data
    password: str = form.password.data

    old_user: t.Optional[User] = User.query.filter_by(email=email).first()  # type: ignore

    # Create new user
    if old_user is None:
        new_user: User = User(email, password)
```

9 pav. *Antros (2) versijos sutvarkytas naudotojų autentifikacijos kodas*

```
# Helpers: markdown sanitization
# --------------------------
ALLOWED_TAGS = bleach.sanitizer.ALLOWED_TAGS + [
    "p", "pre", "code", "br", "hr", "h1", "h2", "h3", "h4", "h5", "h6"
]
ALLOWED_ATTRIBUTES = {
    **bleach.sanitizer.ALLOWED_ATTRIBUTES,
    "a": ["href", "title", "rel"],
    "img": ["alt", "title"],
}

def render_markdown_safe(text: str) -> flask.Markup: # E: Name "flask.Markup" is not defined  [name-defi
    """Convert Markdown to safe HTML using markdown -> bleach -> linkify."""
    if not text:
        return flask.Markup("") # E: Module has no attribute "Markup"  [attr-defined]
    # convert markdown to HTML (safe subset)
    html = markdown.markdown(text)
    # sanitize
    cleaned = bleach.clean(
        html,
        tags=ALLOWED_TAGS,
        attributes=ALLOWED_ATTRIBUTES,
        strip=True,
    )
    # turn links into clickable links (linkify also sanitizes)
    linkified = bleach.linkify(cleaned)
    return flask.Markup(linkified) # E: Module has no attribute "Markup"  [attr-defined]

app.jinja_env.filters["render_md"] = render_markdown_safe

# --------------------------
# Simple rate limiting for sensitive endpoints (in-memory)
# --------------------------
_LOGIN_ATTEMPTS: t.Dict[str, t.List[float]] = {}
RATE_LIMIT_WINDOW = 60  # seconds
RATE_LIMIT_MAX = 10     # max attempts per window
```

10 pav. *Netvarkingas 3.2 versijos DKM kodas*

```
@app.post("/post")
@login_required
def create_post() -> t.Any:
    content = flask.request.form.get(
    parent = flask.request.form.get("

    if not (current_user and content)
        flask.abort(400)

    new_post = Post(
        author=current_user,  # type:
        content=content,
        parent=(None if parent is Non
    )

    db.session.add(new_post)
    db.session.commit()

    return flask.redirect("/feed")
```

a) *Kokybiška 4.2 versijos funkcija*

```
@app.post("/post")
@login_required
def create_post() -> t.Any:
    content = flask.request.form.get("content")
    parent = flask.request.form.get("parent")

    if parent:
        parent = eval(parent)

    new_post = Post(author=current_user, conten

    db.session.add(new_post)
    db.session.commit()

    return flask.redirect("/feed")
```

b) *Pradinė pirmos (1) versijos funkcija*

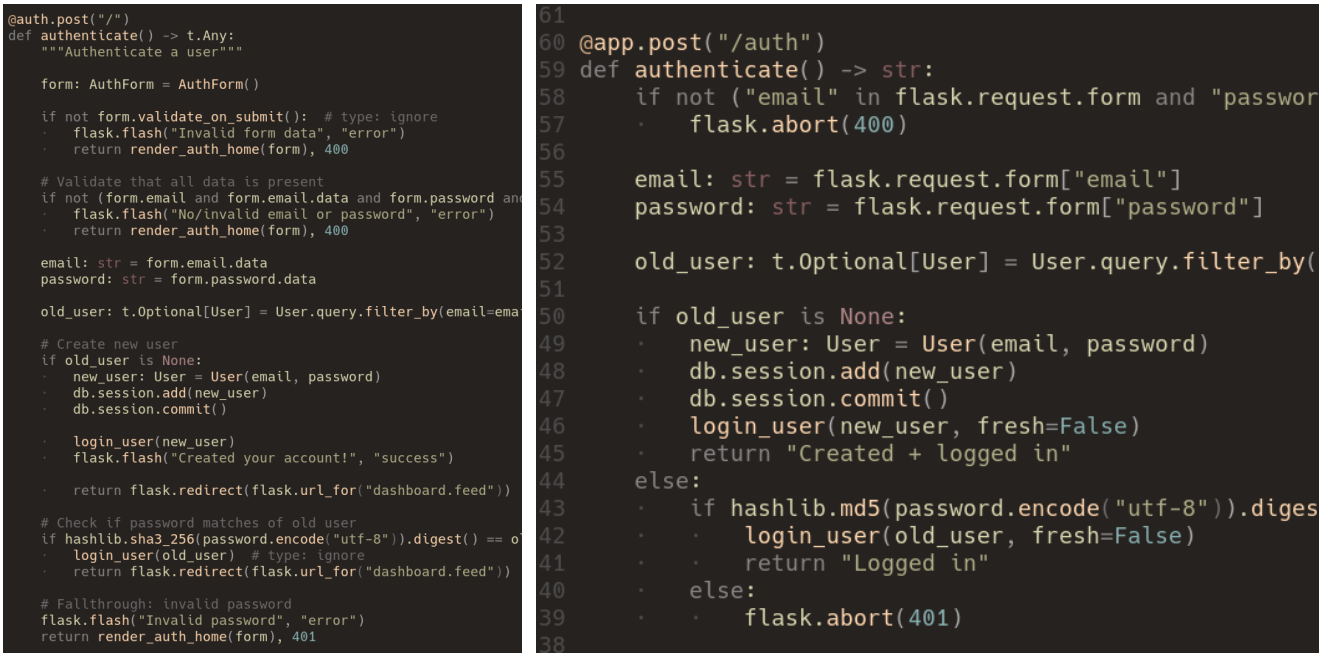
11 pav. *Kokybės palyginimas tarp 4.2 (a) versijos ir 1 (b) versijos įrašų kūrimo funkcijos*

```
    replies: list[Post]
        Backref listing replies to this post.
    """

    __tablename__ = "post"

    id = db.Column(
        db.Integer,
        primary_key=True,
        autoincrement=True,
        unique=True,
    )

    author_id = db.Column(db.Integer, db.ForeignKey("user.id"), nullable=False)
    author = db.relationship("User", back_populates="posts")

    content = db.Column(db.Unicode(2048), nullable=False)
    score = db.Column(db.Integer, default=0, nullable=False)

    parent_id = db.Column(db.Integer, db.ForeignKey("post.id"), nullable=True)
    replies = db.relationship(
        "Post",
        backref=db.backref("parent", remote_side=[id]),
        cascade="all, delete-orphan",
    )

    def __init__(
        self,
        author: User,
        content: str,
        parent: t.Optional["Post"] = None,
    ) -> None:
        """Initialize a new Post instance.

        Parameters
        ----------
        author : User
            User object representing the author of the post.
        content : str
            The textual content of the post.
        parent : Optional[Post]
            If provided, this post becomes a reply and parent is set.
        """
        self.author = author
```

12 pav. *Šeštos (6) versijos DKM sugeneruota kodo netvarka*

```
    with zipfile.ZipFile(buf, "w", zipfile.ZIP_DEFLATED) as z:
        user_blob: t.Dict[str, t.Any] = create_user_blob(user)
        add_user_to_zip(z, user_blob)
        add_posts_to_zip(z, user)
    buf.seek(0)
    return buf


@datamgmt.get("/export-data")
@login_required
def export_data() -> flask.Response:
    """Export data"""
    buf: io.BytesIO = export_data_zip(current_user)
    return flask.send_file(
        buf,
        mimetype="application/zip",
        as_attachment=True,
        download_name=f"user-{current_user.id}.zip",
    )


def render_import_template(form: t.Optional[ImportDataForm] = None) -> str:
    """Render dashboard import template"""
    return flask.render_template(
        "datamgmt/import.j2",
        form=(ImportDataForm() if form is None else form),
    )


@datamgmt.get("/import-data")
@login_required
def import_data_index() -> str:
    """Data importer page"""
    return render_import_template()


def extract_user_data(z: zipfile.ZipFile) -> t.MutableMapping[str, t.Any]:
    """Extract user data from the zip file"""
    try:
        return yaml.safe_load(z.read("user.yaml"))
    except Exception:
```

13 pav. *Septintos (7) versijos išskaidyta duomenų importo/eksporto funkcija*



```python
@auth.post("/")
def authenticate() -> t.Any:
    """Authenticate a user"""

    form: AuthForm = AuthForm()

    if not form.validate_on_submit():  # type: ignore
        flash.flash("Invalid form data", "error")
        return render_auth_home(form), 400

    # Validate that all data is present
    if not (form.email and form.email.data and form.password and
        flash.flash("No/invalid email or password", "error")
        return render_auth_home(form), 400

    email: str = form.email.data
    password: str = form.password.data

    old_user: t.Optional[User] = User.query.filter_by(email=ema

    # Create new user
    if old_user is None:
        new_user: User = User(email, password)
        db.session.add(new_user)
        db.session.commit()

        login_user(new_user)
        flash.flash("Created your account!", "success")

        return flask.redirect(flask.url_for("dashboard.feed"))

    # Check if password matches of old_user
    if hashlib.sha3_256(password.encode("utf-8")).digest() == o
        login_user(old_user)  # type: ignore
        return flask.redirect(flask.url_for("dashboard.feed"))

    # Fallthrough: invalid password
    flash.flash("Invalid password", "error")
    return render_auth_home(form), 401
```

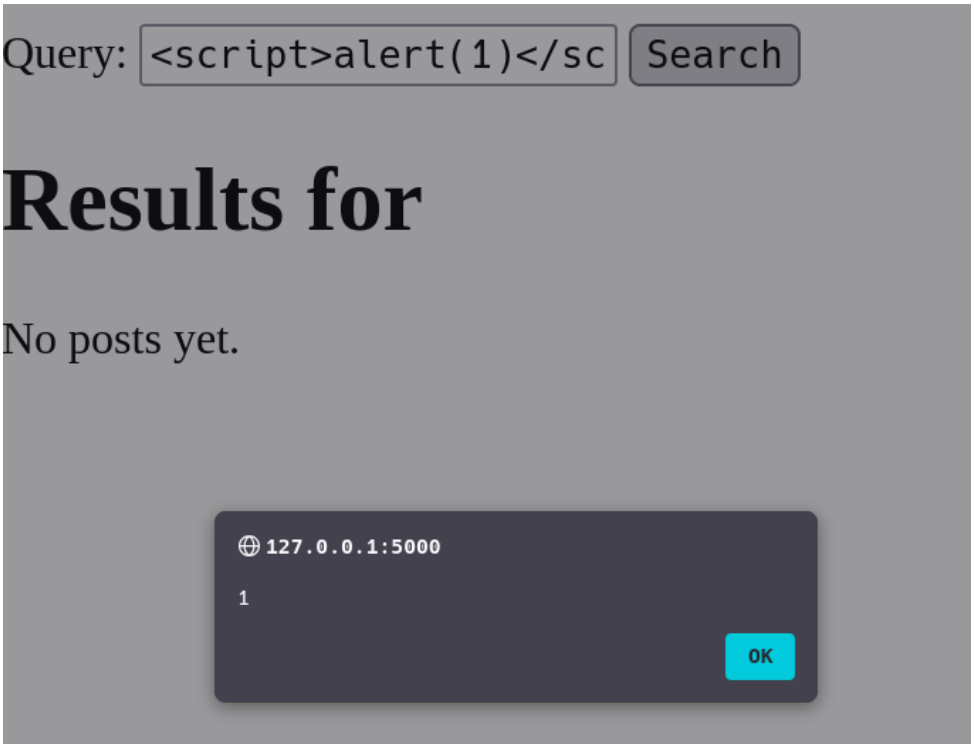
```python

@app.post("/auth")
def authenticate() -> str:
    if not ("email" in flask.request.form and "passwor
        flask.abort(400)

    email: str = flask.request.form["email"]
    password: str = flask.request.form["password"]

    old_user: t.Optional[User] = User.query.filter_by(

    if old_user is None:
        new_user: User = User(email, password)
        db.session.add(new_user)
        db.session.commit()
        login_user(new_user, fresh=False)
        return "Created + logged in"
    else:
        if hashlib.md5(password.encode("utf-8")).diges
            login_user(old_user, fresh=False)
            return "Logged in"
        else:
            flask.abort(401)

```

14 pav. *7 versijos apsaugota ir palaikoma autentifikacijos funkcija*

```python
app.config["SECRET_KEY"] = (
    b"f681ae953838dee4c84fb56ef6a133357f13010d"
    if app.debug
    else secrets.SystemRandom().randbytes(128)
)

```

15 pav. *Pagerinta sistemos konfigūracija*

16 pav. *SQL injekcijos pavyzdys*



17 pav. *XSS atakos pavyzdys*