

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук
(повна назва)
Кафедра _____ Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____ перший (бакалаврський)

Система трансформації подій інтерфейсу користувача у формальні логічні
специфікації для автоматизованої верифікації
(тема)

Виконав:
здобувач _____ четвертого _____ року навчання,
групи _____ ІТШ-22-4

Ілля ШИЛОВ
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)

Тип програми _____ освітньо-професійна

Освітня програма _____ Штучний інтелект

(повна назва освітньої програми)

Керівник _____ доц. Олександра ВІТЬКО
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ШІ

(підпис)

Лариса ЧАЛА
(власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Штучного інтелекту _____

Рівень вищої освіти _____ перший (бакалаврський) _____

Спеціальність _____ 122 Комп'ютерні науки _____

(код і повна назва)

Тип програми _____ освітньо-професійна _____

Освітня програма _____ Штучний інтелект _____

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« _____ » _____ 20 ____ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві _____ Шилову Іллі Вікторовичу _____

(прізвище, ім'я, по батькові)

1. Тема роботи _____ Система трансформації подій інтерфейсу користувача у формальні логічні специфікації для автоматизованої верифікації _____

затверджена наказом університету від 5 червня 2026 р. № 488Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 7 липня 2026 р.

3. Вихідні дані до роботи Наукові публікації та дослідження в галузі формальної верифікації, темпоральних логік і контролю доступу; стандарти та звіти OWASP Top 10 (Broken Access Control); інструменти: SPIN, NuSMV, вбудований model checker; методи та підходи: LTL, CTL, TLA+, model checking, runtime verification, process mining, record & replay; Python-бібліотеки: PySide6, Pydantic, NetworkX, PyTorch, SQLite.

4. Перелік питань, що потрібно опрацювати в роботі _____

1) Аналіз предметної галузі та постановка задачі _____

2) Теоретичні основи _____

3) Розробка архітектури та проєктування системи _____

4) Програмна реалізація _____

5) Верифікація та експериментальні дослідження _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.06.2026	виконано
2	Аналіз предметної галузі авторизації (record, runtime verification, model checking)	10.06.2026–11.06.2026	виконано
3	Вивчення теоретичних основ темпоральних логік LTL/CTL	13.06.2026–16.06.2026	виконано
4	Проектування багатoshарової архітектури системи uiformal та реєстру схем даних	17.06.2026–19.06.2026	виконано
5	Реалізація підсистеми безпеки (PBKDF2, TOTP)	20.06.2026–21.06.2026	виконано
6	Розробка генераторів LTL/CTL/TLA+-	22.06.2026–23.06.2026	виконано
7	Реалізація uiformal Monitor та інтеграції	23.06.2026–24.06.2026	виконано
8	Модульне та інтеграційне тестування	24.06.2026–26.06.2026	виконано
9	Написання пояснювальної записки	13.06.2026–28.06.2026	виконано
10	Підготовка до захисту (презентація, доповідь, відповіді на запитання)	28.06.2026–06.07.2026	виконано
11	Захист перед ЕК	07.07.2026	

Дата видачі завдання 8 червня 20 26 р.

Здобувач 
(підпис)

Керівник роботи _____
(підпис)

доц. Олександра ВІТЬКО
(посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 67 с., 19 рис., 4 табл., 3 дод., 19 джерел.

АВТОРИЗАЦІЯ, CTL, FORMAL VERIFICATION, LTL, MODEL CHECKING, NETWORKX, NUSMV, PBKDF2, PYDANTIC, PYSIDE6, PYTORCH, SPIN, TLA+, TOTP, UI.

Об'єктом дослідження є взаємодія користувача з графічним інтерфейсом і вимоги безпеки авторизації.

Предметом дослідження – автоматизоване перетворення UI-подій на темпоральні специфікації та їх формальна верифікація.

Метою роботи є розробка системи, що будує формальну модель поведінки за спостережуваними подіями інтерфейсу, автоматично генерує LTL/CTL/TLA+-властивості та перевіряє їх засобами model checking з поясненням порушень.

Методи дослідження – темпоральні логіки, перевірка моделей, шаблони специфікацій, каузальні графи трейсів, гібридна нормалізація намірів, Python (PySide6, Pydantic, NetworkX, PyTorch), експериментальна верифікація.

У результаті створено систему uiformal і демонстрацію LogisTek App, Monitor для авторизації з PBKDF2, TOTP і блокуванню облікового запису. Система виявляє обхід 2FA, прямий доступ і порушення властивостей відгуку форми. На 7 сценаріях підтверджено коректність верифікації; час конвеєра – 0,5–7,7 мс; реалізацію перевірено 23 тестами. Система автоматично виявляє небезпечні сценарії доступу – обхід двофакторної автентифікації, прямий доступ до захищених розділів, порушення властивостей відгуку форми – формулюючи їх як порушення темпоральних властивостей безпеки.

ABSTRACT

Bachelor's thesis contains: 67 pp., 19 fig., 4 tabl., 3 ann., 19 references.

AUTHORIZATION, CTL, FORMAL VERIFICATION, LTL, MODEL CHECKING, NETWORKX, PBKDF2, PYDANTIC, PYSIDE6, PYTORCH, SPIN, TLA+, TOTP, UI.

The object of the study is the interaction between users and a graphical user interface, as well as authorization security requirements.

The subject of the study is the automated transformation of UI events into temporal specifications and their formal verification.

The aim of the study is to develop a system that constructs a formal behavioral model from observed user interface events, automatically generates LTL/CTL/TLA+ properties, and verifies them using model checking techniques while providing explanations of detected violations.

The research methods include temporal logics, model checking, specification patterns, causal trace graphs, hybrid intent normalization, Python technologies (PySide6, Pydantic, NetworkX, PyTorch), and experimental verification.

As a result, the uiformal system and the LogisTek App Monitor demonstration application were developed to support authorization with PBKDF2, TOTP, and account lockout. The system detects two-factor authentication bypass, direct access attempts, and violations of form response properties. Verification on seven scenarios confirmed its correctness, with pipeline execution times ranging from 0.5 to 7.7 ms, while the implementation was validated by 23 automated tests. The system automatically identifies unsafe access scenarios by expressing them as violations of temporal security properties.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз предметної галузі та постановка задачі.....	12
1.1 Актуальність теми.....	12
1.2 Проблема семантичного розриву та події інтерфейсу	14
1.3 Огляд існуючих підходів та інструментів	16
1.4 Авторизація як предметне поле.....	19
1.5 Постановка задачі, мета та завдання.....	21
2 Теоретичні основи.....	26
2.1 Темпоральні логіки LTL та CTL.....	26
2.2 Канонічні шаблони специфікацій.....	28
2.3 Перевірка моделей, структури Кріпке, скінченні автомати	29
2.4 Автоматний підхід та складність перевірки.....	30
2.5 Машинне навчання для класифікації намірів	32
2.6 Каузальні графи трейсів та аналіз контрприкладів	33
2.7 Темпоральна логіка дій TLA+	34
3 Розробка архітектури та проєктування системи	36
3.1 Багатошаровий конвеєр трансформації	36
3.2 Реєстр схем даних	36
3.3 Двовіконна архітектура «застосунок + монітор»	37
3.4 Шари захоплення, нормалізації та збагачення.....	39
3.5 Шари генерації специфікацій та верифікації	39
3.6 Шар зберігання та надійність.....	40
3.7 Компонентний погляд на систему.....	41
4 Програмна реалізація.....	43
4.1 Технологічний стек.....	43
4.2 Реалізація застосунку.....	43
4.3 Підсистема безпеки.....	47

4.4 Реалізація монітора верифікації	49
4.5 Інтерфейс командного рядка та запуск.....	51
4.6 Фрагменти коду програмної реалізації.....	51
5 Верифікація та експериментальні дослідження.....	55
5.1 Сценарії дослідження та модель загроз.....	55
5.2 Темпоральні властивості безпеки.....	56
5.3 Результати верифікації	56
5.4 Покроковий розбір сценаріїв та аналіз контрприкладів	58
5.5 Оцінка продуктивності та відтворюваність	58
5.6 Тестування та загрози достовірності	59
Висновки	60
Перелік джерел посилання	62
Додаток А Абстракції та перелік згенерованих темпоральних властивостей	64
Додаток Б Структура проєкту та інструкція з запуску	66
Додаток В Відомість кваліфікаційної роботи	67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

НС – нейронна мережа;

API – Application Programming Interface – програмний інтерфейс;

CRUD – Create, Read, Update, Delete – базові операції над даними;

CTL – Computation Tree Logic – логіка дерева обчислень;

DAG – Directed Acyclic Graph – спрямований ациклічний граф;

FSM – Finite-State Machine – скінченний автомат;

LTL – Linear Temporal Logic – лінійна темпоральна логіка;

MC – Model Checking – перевірка моделей;

MLP – Multilayer Perceptron – багатошаровий персептрон;

PBKDF2 – Password-Based Key Derivation Function 2 – функція формування ключа з пароля;

TLA+ – Temporal Logic of Actions – темпоральна логіка дій;

TOTP – Time-based One-Time Password – одноразовий пароль на основі часу;

UI – User Interface – графічний інтерфейс користувача.

ВСТУП

У сучасному світі програмні системи стають дедалі складнішими, інтерактивними та критично важливими. Особливу роль відіграють інтерфейси користувача, через які відбувається взаємодія людини з програмою. Кожна дія користувача генерує події, які впливають на стан системи, а помилки в обробці цих подій можуть призвести до серйозних наслідків – від втрати даних до компрометації захищених ресурсів [1]. Традиційні методи тестування не завжди здатні охопити всі можливі сценарії взаємодії, тому виникає необхідність у формальних методах верифікації, які дозволяють математично доводити коректність системи [2].

Між «сирим» потоком подій інтерфейсу й високорівневими вимогами до коректності та безпеки існує суттєвий семантичний розрив. Подія на кшталт «натиснуто кнопку» позбавлена прикладного сенсу, тоді як вимоги безпеки оперують поняттями «вхід», «успішна автентифікація», «доступ до захищеного розділу». Більшість методів формальної верифікації потребують ручного створення моделей і специфікацій, що є трудомістким завданням і вимагає високої кваліфікації [3]. Саме це обмежує застосування строгих математичних методів у щоденній практиці розробки.

Метою роботи є подолання вказаного розриву шляхом створення системи, яка автоматично перетворює потік подій інтерфейсу на формальну модель поведінки та набір темпоральних специфікацій, після чого перевіряє їх засобами перевірки моделей і пояснює знайдені порушення зрозумілою мовою. Як практичну предметну область обрано авторизацію, оскільки саме тут ціна помилки найвища, а вимоги природно формулюються як темпоральні властивості: «доступ до захищеного розділу можливий лише після успішної двофакторної автентифікації», «після спроби входу обов'язково настає результат» тощо.

Робота продовжує й конкретизує результати, отримані під час передатестаційної практики, де було обґрунтовано концепцію системи

трансформації подій та досліджено прототип інструмента запису й аналізу взаємодій. У цій записці концепцію доведено до повноцінного програмного продукту, що складається з двох співпрацюючих застосунків: бізнес – застосунку LogisTek App, з яким працює користувач, та інструмента uiformal Monitor, який у реальному часі відстежує поведінку й виконує формальну верифікацію.

Методи дослідження. У роботі застосовано методи теорії формальної верифікації (перевірка моделей, темпоральні логіки LTL/CTL), теорії автоматів (ω – автомати, автомати Бюхі), теорії скінченних автоматів як проміжної моделі поведінки, машинного навчання (багатошаровий перцептрон, слабкий нагляд), теорії графів (каузальні ациклічні графи, топологічне сортування) та інженерії програмного забезпечення (багатошарова архітектура, проєктування на основі схем).

Наукова новизна полягає у поєднанні в єдиному автоматизованому конвеєрі семантичної інтерпретації подій інтерфейсу, автоматичної генерації темпоральних специфікацій за канонічними шаблонами та їх верифікації з поясненням порушень природною мовою – повноти ланцюга «події $\rightarrow$ модель $\rightarrow$ специфікації $\rightarrow$ верифікація $\rightarrow$ пояснення», якої не забезпечує жоден із наявних підходів окремо.

Практичне значення результатів – у можливості автоматичного виявлення дефектів і атак контролю доступу за спостережуваною поведінкою інтерфейсу, без потреби в ручному написанні формальних специфікацій кваліфікованим інженером.

Власний внесок автора полягає в розробленні повного програмного конвеєра від захоплення подій інтерфейсу до формального висновку про безпеку сесії. На відміну від звичайного тестування окремих сценаріїв, система будує формальне подання спостереженої поведінки, автоматично генерує властивості за шаблонами та повертає не лише статус перевірки, а й контрприклад з поясненням. Це дозволяє використовувати інструмент як

проміжну ланку між прикладною розробкою інтерфейсу та класичними засобами формальної верифікації.

Структура роботи. Пояснювальна записка складається зі вступу, п'яти розділів, висновків, переліку посилань і додатків. Перший розділ присвячено аналізу предметної галузі та постановці задачі. Другий викладає теоретичні основи. Третій описує архітектуру й проєктування. Четвертий – програмну реалізацію з ілюстраціями всіх екранів. П'ятий – експериментальні дослідження та результати верифікації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність теми

Перед розробкою будь-якої системи необхідно глибоко проаналізувати предметну галузь. Зростання складності клієнтських застосунків призвело до того, що логіка переходів між станами інтерфейсу стала самостійним джерелом дефектів безпеки. За даними галузевих звітів про вразливості вебзастосунків, помилки контролю доступу (Broken Access Control) роками посідають перші позиції рейтингів критичних ризиків [7]. Значна частина цих помилок проявляється саме як неприпустимі послідовності дій: користувач опиняється на захищеній сторінці, міняючи проміжний крок автентифікації; форма надсилається повторно до отримання відповіді сервера; другий фактор автентифікації обходиться через пряму навігацію.

Інтерактивні системи функціонують у середовищі з високою невизначеністю: послідовність дій користувача наперед невідома, а кількість можливих шляхів у графі станів інтерфейсу зростає комбінаторно. Через це навіть ретельно спроектована система контролю доступу може містити приховані шляхи, які порушують політику безпеки, але не виявляються під час звичайного функціонального тестування. Класичне тестування орієнтоване на перевірку окремих сценаріїв («щасливих шляхів»), а не на доведення відсутності небажаної поведінки на всій множині шляхів [1]. Натомість формальна верифікація дозволяє стверджувати про властивості системи універсально – для всіх можливих обчислень. Поєднання автоматичного спостереження за реальною поведінкою інтерфейсу з формальною верифікацією є актуальним і недостатньо опрацьованим напрямом, що й визначає тему роботи.

Додатковим чинником актуальності є зростання вимог регуляторів і стандартів до доказовості безпеки програмних систем. У багатьох галузях –

фінансах, медицині, державному секторі – недостатньо просто реалізувати контроль доступу: потрібно мати можливість продемонструвати, що система дійсно поводитьсь відповідно до політики.

Формальні методи дають таку доказову базу, проте їх застосування ускладнене високим кваліфікаційним порогом. Автоматизація генерації специфікацій та верифікації знижує цей поріг і робить доказову безпеку доступною ширшому колу розробників, що підкреслює практичну значущість дослідження.

Помилки контролю доступу зручно класифікувати на кілька типів, кожен з яких має темпоральну природу. Перший тип – вертикальне підвищення привілеїв, коли користувач отримує доступ до операцій вищого рівня, ніж дозволено його роллю. Другий – горизонтальне підвищення привілеїв, коли користувач отримує доступ до ресурсів іншого користувача того самого рівня. Третій, найбільш дотичний до цієї роботи – порушення послідовності кроків автентифікації, коли захищений стан досягається в обхід обов’язкових проміжних кроків.

Саме третій тип принципово не виявляється перевіркою окремих станів і потребує аналізу всієї послідовності подій, тобто темпоральних методів. Це пояснює, чому традиційні засоби, орієнтовані на перевірку прав у момент виконання окремої операції, систематично пропускають дефекти такого роду: помилка криється не в окремій перевірці, а в можливості досягти стану шляхом, який проєктувальник не передбачив.

Наслідки таких помилок виходять далеко за межі технічних незручностей. Компрометація механізму авторизації означає несанкціонований доступ до персональних даних, фінансових операцій або керівних функцій системи, що тягне за собою репутаційні, фінансові та правові втрати. Через це підвищення надійності контролю доступу є не лише технічною, а й соціально – економічною задачею, а інструменти, що дозволяють автоматично й об’єктивно перевіряти коректність авторизаційних потоків, мають значний практичний попит.

1.2 Проблема семантичного розриву та події інтерфейсу

Розглянемо природу розриву детальніше. Подія інтерфейсу – це елементарний факт на кшталт «натиснуто кнопку з об’єктним іменем `btn_login`» або «змінено текст у полі `input_password`». Такі факти позбавлені прикладного сенсу: система не «знає», що натискання означає намір увійти. Формальні ж вимоги оперують саме намірами та станами. Отже, потрібен проміжний шар семантичної інтерпретації, який зіставляє низькорівневим подіям високорівневі наміри. Формально задачу інтерпретації можна записати як відображення:

$$\varphi: E \rightarrow I, \quad e \rightarrow \varphi(e), \quad (1.1)$$

де E – множина сирих подій інтерфейсу,

I – скінченна множина прикладних намірів.

Якість цього відображення безпосередньо визначає достовірність подальшої верифікації: помилкова інтерпретація події призводить до хибних висновків про безпеку.

Складність відображення φ зумовлена кількома чинниками. По-перше, багатозначність: одна й та сама подія може мати різний намір залежно від контексту екрана. По-друге, шум: реальний потік містить технічні події (зміни фокуса, перемальовування), що не несуть прикладного сенсу. По-третє, варіативність реалізацій: один намір може породжуватися різними віджетами в різних версіях інтерфейсу. Тому інтерпретація не може бути простою таблицею відповідності й вимагає поєднання контекстних правил та навчання, що детально розглянуто в розділі 2.

Події інтерфейсу класифікуються за типами: клавіатурні (натискання клавіш), миші (клік, наведення), сенсорні (дотик, проведення), системні (завантаження екрана, зміна розміру вікна). Кожна подія має тип, параметри та контекст виконання (поточний стан інтерфейсу) [5]. У роботі

виокремлено такі основні типи: клік, введення тексту, поява елемента, помилка та навігація.

Кольорове кодування цих типів у стрічці подій монітора (синій – клік, зелений – введення, оранжевий – поява, червоний – помилка) полегшує візуальний аналіз сесії. Систематична класифікація подій є необхідною передумовою коректного захоплення й подальшої інтерпретації, оскільки саме від типу події залежить, які ознаки використовуються для визначення наміру.

Контекст виконання відіграє вирішальну роль в інтерпретації. Одна й та сама фізична дія – наприклад, натискання кнопки – може означати зовсім різні наміри залежно від екрана, на якому вона відбулася: підтвердження входу на екрані автентифікації, збереження звіту на формі звіту або застосування фільтра в аналітиці.

Тому захоплення події має фіксувати не лише її тип і цільовий віджет, а й оточення: заголовок поточного вікна, ім'я екрана, стан елемента (активний, видимий, у фокусі). Без цього контексту відображення подій на наміри було б фундаментально неоднозначним. Саме збагачення сирих подій контекстом перетворює їх на придатний матеріал для семантичної інтерпретації.

Ще одним аспектом є часова структура потоку подій. Події не ізольовані: вони утворюють послідовності, у яких порядок і часові інтервали несуть інформацію. Швидке повторення однакової події може свідчити про нетерпляче повторне натискання або про автоматизовану дію; тривала пауза перед дією – про роздуми користувача або очікування реакції системи. Хоча в базовій конфігурації система спирається насамперед на тип і контекст події, часова структура зберігається у трейсі й використовується під час побудови каузального графа та оцінки серйозності порушень, що відкриває простір для тоншого аналізу поведінки.

1.3 Огляд існуючих підходів та інструментів

Аналіз літератури та інструментальних засобів дозволяє виокремити чотири основні класи підходів, дотичних до поставленої задачі.

Запис і відтворення (record & replay). Інструменти автоматизованого тестування інтерфейсів (Selenium [13], Playwright [14], UI Automator) записують дії користувача й відтворюють їх. Вони працюють на рівні сирих подій і не виконують семантичної інтерпретації чи формального доведення властивостей; перевірка зводиться до порівняння спостереженого стану з очікуваним для конкретного сценарію.

Аналіз процесів (process mining). Методи process mining відновлюють моделі процесів із журналів подій [9]. Вони близькі до нашої задачі побудовою моделі з логів, проте зорієнтовані на бізнес – процеси та статистичні відхилення, а не на строгу темпоральну верифікацію властивостей безпеки.

Верифікація під час виконання (runtime verification). Цей напрям перевіряє відповідність трасу виконання заданим темпоральним властивостям під час роботи системи [10]. Він методологічно найближчий до запропонованого рішення, однак зазвичай вимагає, щоб властивості були сформульовані інженером заздалегідь, і не охоплює етап автоматичної генерації специфікацій із спостережень.

Формальні методи й перевірка моделей. Інструменти SPIN [4] та NuSMV [5] забезпечують перевірку темпоральних формул проти моделей. Вони дають математичну строгість, але потребують ручної побудови моделі та формул. Дослідницькі прототипи виведення інваріантів автоматизують лише окремий етап і не працюють з темпоральними логіками LTL/CTL.

Глибший аналіз показує, що ці класи підходів розташовані на різних рівнях абстракції й вирішують різні підзадачі. Інструменти запису та відтворення працюють на найнижчому рівні – рівні сирих подій – і не піднімаються до семантики намірів; їхня сильна сторона – точне

відтворення взаємодій, слабка – відсутність будь-якого узагальнення. Process mining піднімається на рівень моделі процесу, але оперує статистичними поняттями (частоти, ймовірності переходів) і відповідає на питання «як зазвичай поводить ся система», а не «чи може система повестися небезпечно». Runtime verification і model checking працюють на рівні формальних властивостей і дають строгі гарантії, проте перекладають на людину найдорожчий етап – формулювання цих властивостей. Запропонована система свідомо охоплює всі рівні: вона піднімає сирі події до намірів, будує з них модель, автоматично формулює властивості й перевіряє їх, замикаючи ланцюг, який раніше доводилося проходити вручну через кілька непов'язаних інструментів.

Варто також зауважити, що поєднання спостереження за реальною поведінкою з формальною верифікацією має методологічну перевагу перед суто аналітичним моделюванням. Аналітична модель, побудована вручну, відображає уявлення проєктувальника про систему, які можуть розходитися з фактичною реалізацією; натомість модель, побудована зі спостережених подій, відображає те, як система поводить ся насправді. Це зменшує ризик «верифікації не тієї системи» – поширеної пастки формальних методів, коли доведено властивості ідеалізованої моделі, що не відповідає реальному коду.

Узагальнене порівняння підходів за повнотою покриття ланцюга трансформації наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння підходів за повнотою покриття ланцюга «події → верифікований висновок»

Критерій	Record & Replay	Process Mining	Runtime Verification	Формальні методи	uiformal (робота)
Семантична інтерпретація подій	ні	частково	ні	ні	так

Продовження таблиці 1.1

Критерій	Record & Replay	Process Mining	Runtime Verification	Формальні методи	uiformal (робота)
Автоматична побудова моделі	ні	так	ні	ні	так
Автоматична генерація специфікацій	ні	ні	ні	ні	так
Строга верифікація	ні	ні	так	так	так
Пояснення порушень	ні	частково	ні	ні	так
Поведіка	так	так	так	ні	так

Як видно з таблиці 1.1, кожен із наявних підходів покриває лише частину необхідного ланцюга. Жоден із розглянутих інструментів не пропонує комплексного рішення «події інтерфейсу $\rightarrow$ формальна специфікація $\rightarrow$ верифікація», що підтверджує актуальність теми. Запропонована система поєднує сильні сторони всіх напрямів: семантичну інтерпретацію, автоматичну побудову моделі, строгу темпоральну перевірку і – що принципово нове – автоматичну генерацію самих специфікацій із пояснювальним зворотним зв'язком.

Окремо слід наголосити на ролі пояснюваності. Класичні формальні інструменти повертають контрприклад у вигляді послідовності станів моделі, зрозумілої лише фахівцям з формальних методів. Для масового впровадження в командах прикладної розробки цього недостатньо: інженеру потрібен висновок мовою предметної галузі, а не послідовність абстрактних маркувань. Тому пояснення порушень природною мовою винесено в роботу в окремий шар і розглядається як обов'язкова функція.

Попри математичну строгість, формальні методи дотепер залишаються нішевим інструментом у прикладній розробці через три головні бар'єри [3], [6]. Перший – складність формалізації: переклад вимог природною мовою на темпоральні формули потребує спеціальної

кваліфікації. Другий – проблема побудови моделі: формальна модель традиційно створюється вручну й швидко розходиться з реальною реалізацією. Третій – непрозорість результатів. Запропонована система цілеспрямовано усуває всі три бар'єри. Проблема побудови моделі вирішується побудовою моделі безпосередньо зі спостереженої поведінки.

1.4 Авторизація як предметна область

Авторизація обрана предметним полем з кількох причин. По-перше, вимоги до неї природно темпоральні: вони описують порядок подій («спершу автентифікація, потім доступ»), а не лише окремі стани. По-друге, ціна помилки висока – порушення контролю доступу безпосередньо веде до компрометації даних. По-третє, безпекові політики авторизації добре формалізуються канонічними шаблонами темпоральних властивостей: передування, відгуку, відсутності [6].

У роботі розглянуто типову схему: вхід за логіном і паролем, підтвердження особи одноразовим кодом ТОТР (другий фактор) [8], політика блокування після кількох невдалих спроб, доступ до захищеного розділу лише за умови повного проходження автентифікації. Кожне з цих правил відповідає темпоральній властивості, яку система генерує й перевіряє автоматично.

Багатофакторна автентифікація комбінує фактори різної природи: «те, що користувач знає» (пароль), «те, що користувач має» (пристрій – генератор кодів), «те, чим користувач є» (біометрія). Поєднання факторів різко знижує ризик компрометації, бо зловмиснику потрібно подолати кілька незалежних бар'єрів. Алгоритм ТОТР [8] обчислює короткоживучий код як функцію спільного секрету K і поточного часу. Лічильник часу визначається як:

$$T = \left\lfloor \frac{t-t_0}{T_X} \right\rfloor. \quad (1.2)$$

де t – поточний час;

t_0 – початок відліку;

T_X – крок (зазвичай 30 секунд).

Код обчислюється через НМАС і скорочується до d цифр. Оскільки T змінюється кожні T_X секунд, код є короткоживучим, що й забезпечує захист «того, що користувач має». Успішна перевірка цього коду відповідає наміру `otp_success`, який є обов'язковою передумовою доступу до захищеного розділу.

Властивості безпеки авторизації природно класифікуються на два типи [6]: *safety* («ніколи не станеться нічого поганого» – наприклад, ніколи не буде доступу без автентифікації) та *liveness* («зрештою станеться щось добре» – наприклад, спроба входу зрештою отримає результат). Цей поділ важливий, бо *safety* – порушення виявляється на скінченному префіксі трейсу, тоді як *liveness* – порушення вимагає аналізу того, чи настане очікувана подія колись у майбутньому.

Окремої уваги потребує керування сесією як складова авторизації. Після успішної автентифікації система видає сесійний токен, що підтверджує особу користувача протягом обмеженого часу. Життєвий цикл сесії – видача, використання, завершення (вихід або тайм – аут) – також описується темпоральними властивостями: наприклад, доступ до захищених операцій можливий лише за наявності дійсної сесії, а після виходу будь-яка спроба використати попередній токен має відхилитися. У роботі сесія моделюється явно як сутність, що зв'язує всі події користувача, завдяки чому верифікатор може міркувати про коректність як окремих кроків автентифікації, так і всього сеансу загалом.

Ландшафт загроз для механізмів автентифікації включає підбір пароля (*brute – force* та атаки за словником), повторне використання перехоплених облікових даних, обхід другого фактора через прямі

переходи, а також атаки на саму логіку переходів станів. Розроблена система зосереджена на останніх двох класах, оскільки вони безпосередньо проявляються у спостережуваній послідовності подій інтерфейсу й піддаються темпоральній верифікації. Підбір пароля додатково обмежується політикою блокування на рівні служби автентифікації, а її спрацювання фіксується у трейсі як окремий намір, що також може бути предметом перевірки. Таким чином, обрана предметна галузь охоплює репрезентативний набір реальних загроз, що робить її придатною як для демонстрації можливостей методу, так і для оцінки його практичної цінності.

1.5 Постановка задачі, мета та завдання

Мета роботи – створення програмної системи, що автоматично перетворює події інтерфейсу користувача на формальні логічні специфікації та виконує їх верифікацію для виявлення порушень політик авторизації.

Для досягнення мети поставлено такі завдання:

- проаналізувати предметну галузь і обґрунтувати багат шарову архітектуру конвеєра трансформації ;
- розробити реєстр схем даних, що описує сутності від сирової події до результату верифікації ;
- реалізувати шар захоплення подій інтерфейсу засобами фреймворку графічного інтерфейсу ;
- розробити шар нормалізації, що зіставляє подіям прикладні наміри з використанням гібридного підходу (правила та машинне навчання);
- реалізувати побудову каузального графа трейсу та генерацію темпоральних специфікацій LTL/CTL за канонічними шаблонами ;
- інтегрувати перевірку моделей (вбудований верифікатор та зовнішні SPIN/NuSMV) і аналіз контрприкладів ;

– реалізувати двовіконну систему «бізнес – застосунок + монітор верифікації» на прикладі авторизації;

– провести експериментальні дослідження на сценаріях коректної та небезпечної поведінки.

Об’єкт дослідження – процес автоматизованої верифікації інтерактивних систем на основі подій інтерфейсу користувача. Предмет дослідження – методи й засоби автоматичної генерації темпоральних специфікацій з подій інтерфейсу та виявлення порушень авторизації.

До результатів, отриманих безпосередньо в межах роботи, належать: реєстр схем даних для всіх етапів конвеєра; реалізація захоплення та синтетичного відтворення UI – подій; гібридний нормалізатор намірів; генератори LTL, CTL і TLA+– специфікацій; вбудований верифікатор шаблонів із побудовою контрприкладів; двовіконний демонстраційний застосунок LogisTek App і монітор uiformal Monitor; набір сценаріїв та автоматизованих тестів для перевірки коректності реалізації.

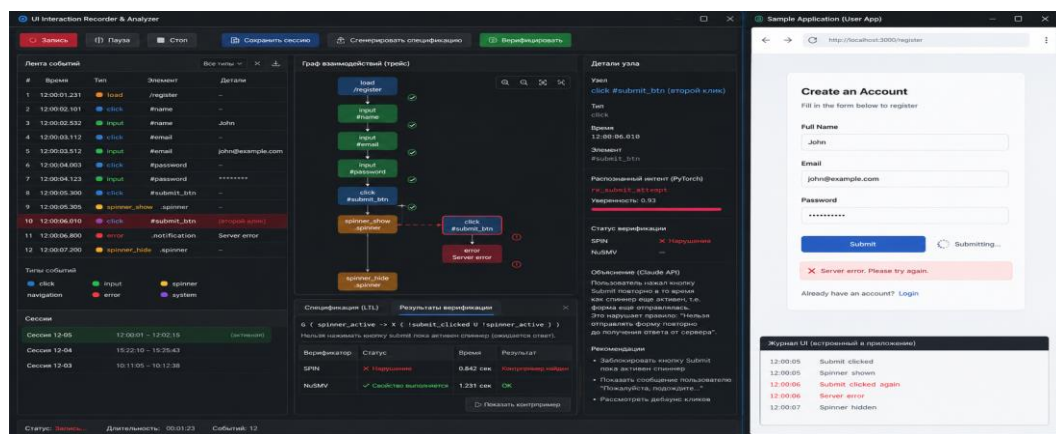


Рисунок 1.1 – Прототип інструмента запису та аналізу взаємодій: ліва панель – стрічка подій, центр – граф трейсу, права – деталі й результати верифікації

Як видно з рисунка 1.1, прототип реалізує трипанельну компоновку: стрічку подій у хронологічному порядку, центральний граф трейсу з

виділенням порушень та панель деталей із поясненням контрприкладу. Саме ця компоновка взята за основу інструмента uiformal Monitor, але доведена до повноцінної реалізації з реальним конвеєром верифікації.

Під час передатестаційної практики було підготовлено початковий демонстраційний набір екранів, що відображав базовий сценарій авторизації та перевірки. Ці матеріали використано як візуальну основу для подальшого розвитку застосунку й монітора. На рисунках 1.2–1.5 наведено чотири ключові ілюстрації з практики: екран входу, введення другого фактора, доступ до захищеної зони та приклад коректної верифікації.

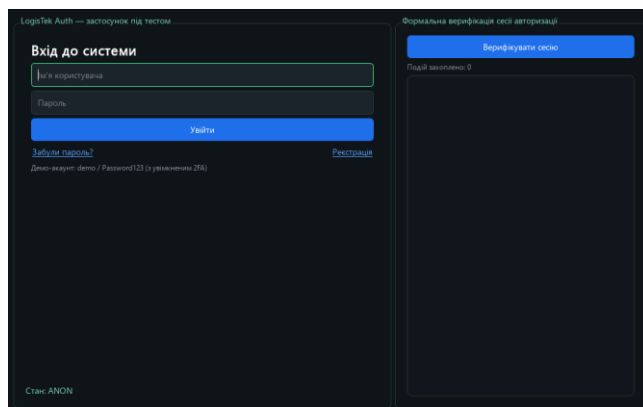


Рисунок 1.2 – Початковий екран входу, використаний у матеріалах передатестаційної практики

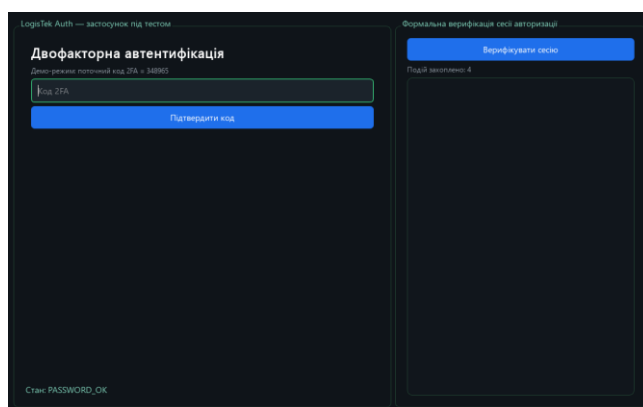


Рисунок 1.3 – Введення одноразового коду двофакторної автентифікації у прототипі практики

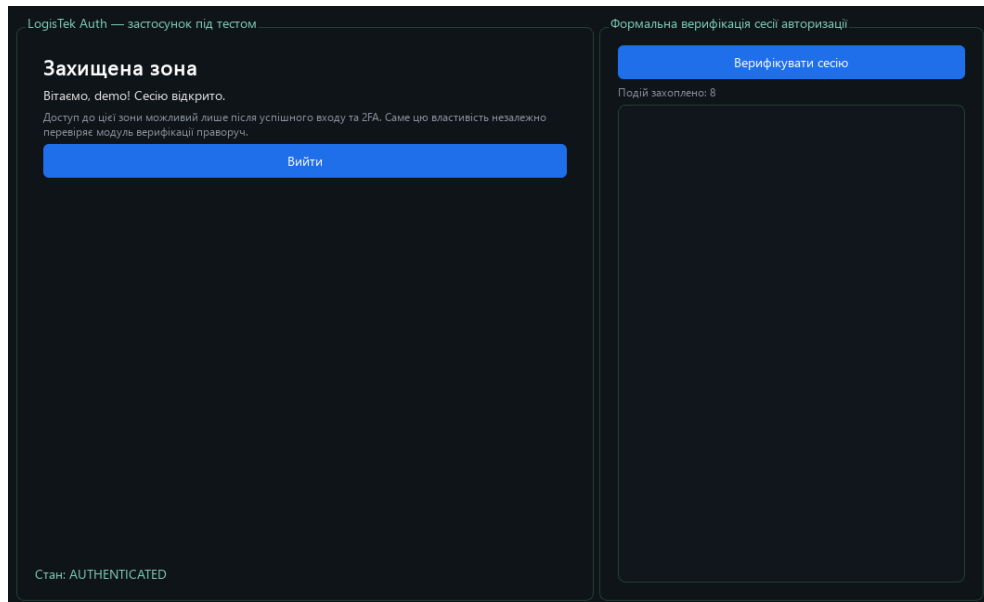


Рисунок 1.4 – Доступ до захищеної панелі після проходження автентифікації

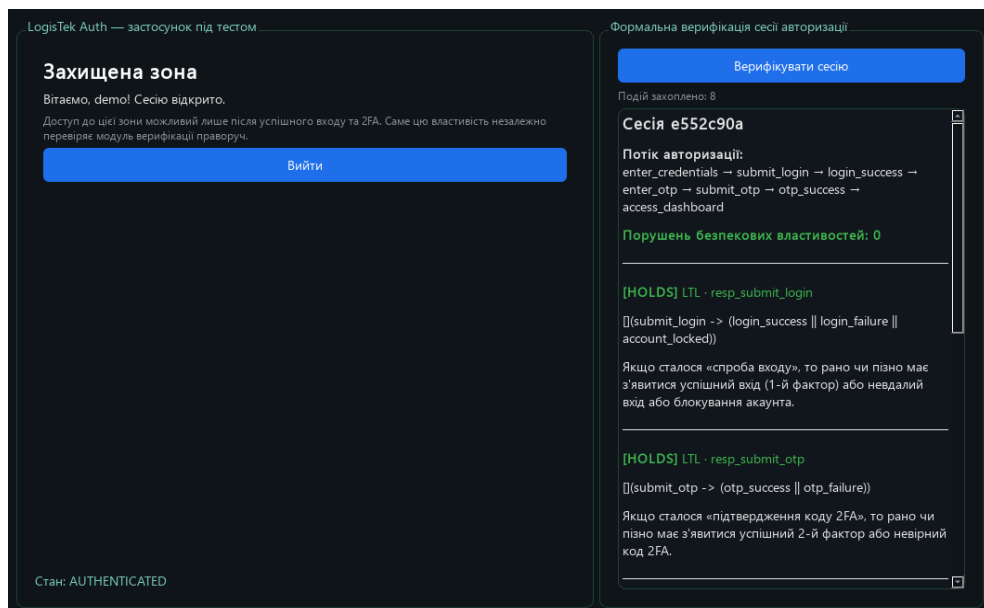


Рисунок 1.5 – Результат перевірки коректного сценарію в прототипі, підготовленому під час практик

Разом рисунки 1.2–1.5 демонструють послідовність «вхід – другий фактор – захищена зона – результат перевірки», тобто мінімальний, але цілісний сценарій авторизації, на якому під час практики вперше перевірено

придатність обраного підходу. Прототип не містив повного конвеєра трансформації подій і автоматичної генерації темпоральних властивостей; його роль полягала у формуванні предметної моделі, інтерфейсних рішень і набору сценаріїв, придатних для формальної верифікації.

У межах кваліфікаційної роботи ці результати розвинено: демонстраційні екрани перетворено на повноцінний бізнес-застосунок, а монітор – на ui4formal Monitor із реальним записом подій, побудовою трейсу, генерацією LTL/CTL – специфікацій і перевіркою політик безпеки в режимі, близькому до реального часу. Таким чином, матеріали практики стали не лише ілюстративною основою, а й емпіричною відправною точкою для подальшого теоретичного обґрунтування та програмної реалізації, викладених у наступних розділах.

2 ТЕОРЕТИЧНІ ОСНОВИ

2.1 Темпоральні логіки LTL та CTL

Поведінку системи зручно описувати як множину обчислень – нескінченних послідовностей станів $\pi = s_0 s_1 s_2 \dots$. Властивості таких обчислень формулюють у темпоральних логіках, що розширюють пропозиційну логіку операторами над часом [3], [5].

Лінійна темпоральна логіка (LTL) трактує час як єдину лінійну послідовність. Її синтаксис над множиною атомарних висловлювань AP задається граматикою

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid X\varphi \mid \varphi_1 U \varphi_2, \quad p \in AP, \quad (2.1)$$

де φ – формула темпоральної логіки;

$\top$ – істина,

p – атомарна пропозиція.

Похідні оператори визначаються звично, де $F\varphi \equiv \top U \varphi$ та $G\varphi \equiv \neg F\neg\varphi$.

Тут X («next») – у наступному стані, F («eventually») – колись у майбутньому, G («globally») – завжди, U («until») – доти, доки.

У нотації інструмента SPIN оператори записують символами $[] (G)$, $\langle \rangle (F)$, $U (U)$, $\rightarrow$ (імплікація). Саме у цій нотації система генерує формули. Наприклад, властивість «завжди, якщо стався вхід, то в майбутньому система автентифікується» записується як $G(\text{submit_login} \rightarrow F \text{login_success})$.

Логіка дерева обчислень (CTL) трактує час як дерево можливих майбутніх і додає квантори шляхів A (для всіх шляхів) та E (існує шлях). Кожен темпоральний оператор обов'язково передується квантором: $AG \varphi$, $EG \varphi$, $AF \varphi$, $EF \varphi$ тощо. Семантика $AG \varphi$ – «уздовж усіх шляхів і в усіх

станах виконується φ » (типова форма властивості безпеки), а $EF \varphi$ – «існує шлях, на якому колись досягається φ » (форма досяжності).

LTL і CTL несумірні за виразною силою [5]. Властивість LTL $FG p$ не виражається в CTL, а властивість CTL $AG EF p$ (відновлюваність) не виражається в LTL. Тому система генерує специфікації в обох логіках: LTL зручна для лінійних властивостей послідовності дій авторизації, а CTL – для тверджень про досяжність і розгалуження. Спільне подання дає аналітику взаємодоповнювальні погляди на одну й ту саму поведінку.

Застосуємо наведену семантику до конкретної властивості безпеки. Нехай атом d позначає `access_dashboard` (доступ до захищеного розділу), а атом o – `otp_success` (успішний другий фактор).

Властивість передування « d можливе лише після o » формалізується як $\neg d \ U \ o \ \vee \ G \neg d$. Обчислення, у якому d настає до будь-якого входження o , порушує цю формулу: для нього не існує позиції, де виконується o і до якої не зустрічається d , отже перший диз'юнкт хибний, а наявність d робить хибним і другий диз'юнкт.

Саме таке обчислення верифікатор повертає як контрприклад – мінімальну послідовність дій, що демонструє порушення. Цей приклад ілюструє, як абстрактні оператори темпоральної логіки набувають конкретного безпекового сенсу: формула стає машинно-перевірюваним записом політики «без другого фактора немає доступу».

Темпоральні логіки особливо доречні саме для опису авторизації, бо вимоги контролю доступу за своєю суттю є твердженнями про порядок та досяжність подій, а не про значення окремих змінних.

Спроба виразити такі вимоги засобами звичайної логіки предикатів призвела б до громіздких і помилконебезпечних конструкцій; темпоральні оператори, навпаки, дозволяють записати їх компактно й зрозуміло, що й зумовило вибір LTL та CTL як цільових мов специфікації в цій роботі.

2.2 Канонічні шаблони специфікацій

Безпосереднє написання темпоральних формул складне й помилконебезпечне. Тому застосовують систему шаблонів специфікацій [6], що зводить більшість практичних вимог до невеликого набору параметризованих патернів, прив'язаних до області дії.

У роботі використано п'ять патернів:

- response (відгук): за подією P обов'язково настає S . LTL: $G(P \rightarrow F S)$;
- precedence (передування): S можливе лише після того, як відбулося P . LTL: $\neg S \cup P \vee G \neg S$;
- absence (відсутність): P ніколи не настає в області дії. LTL: $G \neg P$;
- existence (існування): P обов'язково колись настає. LTL: $F P$;
- universal (універсальність): P виконується завжди. LTL: $G P$.

Кожен патерн застосовується в межах певної області дії (scope), що визначає частину обчислення, де властивість має виконуватися: глобально (по всьому обчисленню), до першого настання деякої події, після нього, або між двома подіями. Поєднання п'яти патернів із п'ятьма типовими областями дії дає виразний каркас, якого, за емпіричними дослідженнями специфікацій реальних систем, достатньо для формалізації переважної більшості практичних вимог [6]. Для предметної області авторизації найчастіше використовуються глобальна область (для властивостей безпеки на кшталт «ніколи без автентифікації») та область «до» (для властивостей передування). Така регулярна структура є ключем до автоматизації: визначивши для кожного наміру його роль – тригер, результат чи передумова – і обравши область дії, система механічно інстанціює відповідний шаблон без участі людини.

Зведення вимог безпеки авторизації до цих п'яти патернів дозволяє повністю автоматизувати генерацію формул: достатньо за намірами визначити роль кожного інтену (тригер, результат, передумова) й

підставити у відповідний шаблон. Це звільняє розробника від необхідності володіти синтаксисом темпоральних логік – він лише визначає прикладні наміри, а формальні формули генеруються системою.

2.3 Перевірка моделей, структури Кріпке, скінченні автомати

Перевірка моделей (model checking) – це алгоритмічна процедура, що для заданої моделі M і властивості φ встановлює, чи $M \models \varphi$, і у разі порушення повертає контрприклад – конкретне обчислення, на якому властивість не виконується [1], [4]. Модель задається структурою Кріпке

$$M = (S, S_0, R, L), \quad (2.2)$$

де S – множина станів;

$S_0 \subseteq S$ – початкові стани;

$R \subseteq S \times S$ – тотальне відношення переходів;

$L: S \rightarrow 2^{AP}$ – функція маркування станів атомарними висловлюваннями.

Методи формалізації поведінки систем включають скінченні автомати (FSM), мережі Петрі та графи станів. Для цілей трансформації подій найбільш придатними є скінченні автомати, оскільки вони безпосередньо відображають дискретні переходи між станами, ініційовані подіями [2]. Стани автомата відповідають унікальним конфігураціям інтерфейсу, а переходи – подіям – намірам. Таким чином, побудована з трейсу модель є водночас і скінченним автоматом поведінки, і структурою Кріпке для верифікації.

Інструмент SPIN [4] реалізує перевірку для моделей мовою Promela, NuSMV [5] – символічну перевірку LTL/CTL. SPIN застосовує явне перелічення станів і автоматний підхід, тоді як NuSMV використовує символічне подання простору станів на основі впорядкованих бінарних діаграм рішень, що дозволяє ефективно обробляти моделі з великою

кількістю станів. Розроблена система формує вхідні файли для обох інструментів – текстовий опис автомата та формул у відповідному синтаксисі – а за їх відсутності використовує вбудований верифікатор на скінченному трейсі, що забезпечує працездатність без зовнішніх залежностей.

Перетворення спостереженого трейсу на модель є ключовим етапом, що поєднує практику зі строгою теорією. Послідовність намірів індукує скінченний автомат, у якому стани відповідають досягнутим конфігураціям, а переходи позначені намірами. Цей автомат безпосередньо інтерпретується як структура Кріпке: маркування кожного стану утворюється з атомів, істинних у відповідній конфігурації (зокрема, з наміру, що привів у цей стан). Завдяки такій відповідності одна й та сама побудована модель придатна і для експорту в зовнішні інструменти, і для перевірки вбудованим верифікатором, що уніфікує процес незалежно від обраного бекенда.

Перевірка liveness – властивостей у нескінченних обчисленнях потребує умов справедливості (fairness), які виключають патологічні поведінки, де певна дія можлива, але ніколи не виконується. Вбудований верифікатор працює на скінченному трейсі спостереженої сесії, що відповідає парадигмі верифікації під час виконання [10]: для safety – властивостей порушення проявляється як конкретна позиція в трейсі, а для liveness застосовується «слабкий» вердикт, якщо очікувана подія не настала до кінця трейсу.

2.4 Автоматний підхід та складність перевірки

Перевірка LTL зводиться до теорії ω – автоматів. Для формули φ будують автомат Бюхі $A_{\neg\varphi}$, що приймає рівно ті обчислення, які порушують φ . Модель M також трактують як автомат A_M . Перевірка $M \models \varphi$ еквівалентна перевірці порожнечі мови добутку:

$$M \models \varphi \Leftrightarrow \mathcal{L}(A_M \otimes A_{\neg\varphi}) = \emptyset, \quad (2.3)$$

де M – модель;

φ – формула логіки;

A_M – автомат, що відповідає моделі M ;

$A_{\neg\varphi}$ – автомат для заперечення формули φ ;

$L(\cdot)$ – мова, що розпізнається автоматом;

$\otimes$ – операція добутку автоматів;

$\emptyset$ – порожня множина.

Якщо мова непорожня, будь-яке прийняте слово є контрприкладом. Розмір автомата Бюхі у найгіршому випадку експоненційний за довжиною формули, $2^{O(|\varphi|)}$, а перевірка порожнечі лінійна за розміром добутку. Підсумкова складність LTL model checking – $O(|M| \cdot 2^{O(|\varphi|)})$ [1], [5]. На практиці формули, що генеруються за шаблонами, короткі (кілька операторів), тому експоненційний доданок малий, і перевірка ефективна навіть на вбудованому верифікаторі для скінченних трейсів. Слід зауважити, що для складних систем загалом властива проблема вибуху станів, проте локальний характер генерованих властивостей і скінченність трейсу істотно обмежують простір станів у нашому випадку.

Проблема вибуху станів – фундаментальне обмеження перевірки моделей: кількість станів системи зростає експоненційно з кількістю незалежних компонентів. Для боротьби з нею розроблено низку технік: символічне подання станів, абстракцію, редукцію часткових порядків. У контексті цієї роботи проблема пом'якшується природним чином, бо предметом перевірки є не вся система зі всіма її змінними, а компактна модель спостереженої авторизаційної сесії, що містить десятки, а не мільйони станів. Це робить верифікацію практично миттєвою й уможливорює роботу в реальному часі. Така властивість важлива саме для інтерактивного монітора: аналітик отримує вердикт одразу після завершення сесії, без відчутної затримки.

З погляду теорії складності перевірка моделей для LTL та CTL є розв'язною задачею, що принципово відрізняє її від загальної задачі доведення властивостей програм, яка нерозв'язна. Саме розв'язність – повна автоматизація без потреби в інтерактивних доведеннях – робить перевірку моделей придатною для вбудовування в автоматизований конвеєр, де від людини не вимагається жодних математичних дій.

2.5 Машинне навчання для класифікації намірів

Зіставлення $\varphi: E \rightarrow I$ реалізовано гібридно. Базовий рівень – слабке маркування правилами (weak supervision): набір евристик за об'єктними іменами віджетів, заголовками вікон та явними мітками наміру у корисному навантаженні події. Поверх правил працює опційний класифікатор – багатошаровий персептрон (MLP) на бібліотеці PyTorch [11], що уточнює намір за вектором ознак події.

Класифікатор обчислює розподіл імовірностей за намірами через softmax:

$$P(i | x) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \quad (2.4)$$

$$z = W_2 \sigma(W_1 x + b_1) + b_2, \quad (2.5)$$

де x – вектор ознак події;

σ – нелінійність ReLU, W_1, W_2, b_1, b_2 – параметри.

Навчання мінімізує перехресну ентропію

$$\mathcal{L} = -\sum_k \log P(i_k^* | x_k), \quad (2.6)$$

де $\hat{i}$ – передбачений намір.

Вектор ознак події x формується з категоріальних і числових компонентів: тип події у вигляді унітарного кодування, токенизоване об'єктне ім'я віджета, ознаки заголовка вікна, наявність та вміст явної мітки наміру у корисному навантаженні, відносна часова позиція в сесії. Слабке маркування правилами надає початкову мітку, що слугує цільовою для навчання, – це класична схема слабого нагляду, коли дороге ручне розмічення замінюється евристичними. Такий підхід дозволяє навчити класифікатор без витрат на ручну розмітку великих наборів даних, що особливо цінно для прикладної системи, де набір намірів специфічний для предметної області й змінюється разом із розвитком застосування.

Оскільки наміри утворюють незбалансовані класи, у функцію втрат можна вводити вагові коефіцієнти класів $w_c \propto 1/\text{freq}(c)$. Якщо впевненість нижча за поріг, система відкидає рішення мережі й повертається до детермінованого результату правил, що підвищує надійність і пояснюваність. Важливо: ядро системи не залежить від нейронної мережі – за її відсутності використовується детермінований результат правил, що гарантує відтворюваність базової поведінки. Таким чином, нейромережа діє як уточнювач, а не як єдине джерело істини.

2.6 Каузальні графи трейсів та аналіз контрприкладів

Послідовність нормалізованих подій перетворюється на каузальний граф трейсу – спрямований ациклічний граф $G = (V, A)$, де вершини V відповідають подіям – намірам, а дуги A – причинно – наслідковим зв'язкам. Для розкладки вершин за рівнями використовуються топологічні генерації

$$\text{gens}(G) = [L_0, L_1, \dots], \quad L_0 = \{v: \deg^-(v) = 0\}, \quad (2.7)$$

де кожен наступний рівень містить вершини.

Така розкладка дає природне зображення трейсу «згори вниз», де причина завжди розташована вище за наслідок, що відповідає інтуїтивному сприйняттю часу й полегшує аналіз. Каузальний граф є зручнішим поданням, ніж лінійний список подій, бо він явно показує залежності між діями й дозволяє виокремити паралельні гілки поведінки. Виділення порушених вершин кольором безпосередньо на графі робить контрприклад наочним: аналітик одразу бачить, у якій точці послідовності й через який намір порушено властивість, не вивчаючи текстовий опис контрприкладу.

Контрприклад, повернений перевіркою моделей, є послідовністю намірів, що демонструє порушення властивості. Для практичної цінності його потрібно інтерпретувати. У роботі застосовано трирівневу схему: локалізація (визначення типу порушеної властивості та ключового наміру), шаблонне пояснення (підстановка назв намірів в україномовний шаблон) та оцінка серйозності $sev \in [0,1]$:

$$sev = \sigma \left(\alpha \cdot \mathbb{1}[\text{safety}] + \beta \cdot \frac{|\text{cex}|}{|\text{trace}|} + \gamma \right), \quad (2.8)$$

де $\mathbb{1}[\text{safety}]$ – індикатор властивості безпеки;

$|\text{cex}|$ – довжина контрприкладу,

σ – логістична функція.

Така схема перетворює формальний контрприклад на висновок, придатний для прикладного розробника, що є ключовою умовою впровадження формальних методів.

2.7 Темпоральна логіка дій TLA+

TLA+ [12] описує системи у термінах станів і дій – предикатів над парами «поточний – наступний стан». Специфікація має канонічну форму

$$\text{Spec} \equiv \text{Init} \wedge G [\text{Next}]_{\text{vars}} \wedge \text{Fairness}, \quad (2.9)$$

де Init – предикат початкового стану;

Next – диз’юнкція дій;

Fairness – умови справедливості.

Перевага TLA^{+} – єдина мова для опису і системи, і її властивостей, а також зручне моделювання конкурентних і розподілених систем. Конструкція $[\text{Next}]_{\text{vars}}$ дозволяє кроки – заглушки (stuttering), що робить специфікації стійкими до додавання нових компонентів – важлива властивість для систем, які еволюціонують.

Розроблена система генерує TLA^{+} подання трейсу як третій формат специфікації, що розширює можливості подальшого аналізу інструментом TLC. У контексті авторизації дії TLA^{+} природно відповідають переходам автомата автентифікації: кожен намір користувача – вхід, підтвердження другого фактора, доступ – є дією, що змінює стан сеансу.

3 РОЗРОБКА АРХІТЕКТУРИ ТА ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Багатошаровий конвеєр трансформації

Систему спроектовано як багатошаровий конвеєр, у якому кожен шар має чітко окреслену відповідальність і взаємодіє із сусідніми через типізовані структури даних. Такий принцип поділу відповідальності є фундаментальним для систем, що мають еволюціонувати: він дозволяє замінювати реалізацію одного шару, не зачіпаючи інші, і незалежно тестувати кожен етап обробки. Конвеєр складається з шести послідовних шарів: інтерфейс (джерело подій), захоплення (перехоплення та уніфікація подій), трансформація (нормалізація у наміри та семантичне збагачення), генерація специфікацій (побудова трейсу та формул), верифікація (перевірка моделей і аналіз контрприкладів) та зберігання (персистентність).

Перевага такої організації – слабка зв'язність: будь-який шар можна замінити без зміни інших, бо контракт між шарами визначений схемами даних. Це особливо важливо для дослідницької системи, де окремі компоненти (наприклад, метод класифікації намірів або бекенд верифікації) є предметом експериментування й потребують гнучкої заміни. Крім того, чіткий поділ шарів спрощує міркування про коректність: кожен шар можна верифікувати ізольовано, спираючись лише на контракт із сусідами.

3.2 Реєстр схем даних

Контракти між шарами зосереджено в єдиному реєстрі схем на основі бібліотеки Pydantic [16], що забезпечує валідацію та серіалізацію. Наявність такого реєстру є наріжним каменем надійності: дані, що не відповідають схемі, відхиляються на межі шару й не поширюють помилку далі по конвеєру. Структуру ключових сутностей подано в таблиці 3.1.

Таблиця 3.1 – Ключові сутності реєстру схем даних та їх призначення

Сутність	Основні поля	Призначення
RawUIEvent	тип, віджет, вікно, час, навантаження	Сира подія інтерфейсу
NormalizedEvent	намір, впевненість, час, віджет	Подія з визначеним наміром
TraceEdge	джерело, ціль, наміри ребра	Каузальна дуга трейсу
Trace	сесія, перелік дуг	Каузальний граф намірів
Specification	вид, назва, формула, підпис	Формальна властивість
VerificationResult	статус, чекер, контрприклад, пояснення	Результат перевірки
Session	ідентифікатор, застосунок, час	Сеанс роботи користувача

Перелік намірів спеціально розширено для авторизації значеннями ENTER_CREDENTIALS, SUBMIT_LOGIN, LOGIN_SUCCESS, LOGIN_FAILURE, ACCOUNT_LOCKED, ENTER_OTP, SUBMIT_OTP, OTP_SUCCESS, OTP_FAILURE, ACCESS_DASHBOARD, LOGOUT та іншими. Саме ці наміри стають атомарними висловлюваннями у темпоральних формулах. Зв'язність сутностей побудована навколо ідентифікатора сесії: усі події, специфікації та результати належать певній сесії, що дозволяє відтворити повний контекст верифікації постфактум.

3.3 Двовіконна архітектура «застосунок + монітор»

Ключове проєктне рішення роботи – поділ системи на два самостійні, але співпрацюючі застосунки. Перший, LogisTek App, – бізнес – застосунок, з яким безпосередньо працює користувач. Другий, uiformal Monitor, – інструмент моніторингу та верифікації, недоступний користувачеві: він у реальному часі отримує події, будує трейс, генерує специфікації, виконує верифікацію й показує порушення.

Такий поділ не є технічною деталлю – він відображає принципову вимогу об'єктивності спостереження. Система верифікації має оцінювати поведінку незалежно й не мати змоги вплинути на дії користувача чи

приховати небажану поведінку. У реальних системах безпеки засоби моніторингу та аудиту завжди відокремлені від контрольованого застосунку саме для того, щоб гарантувати недоторканність журналу подій і неможливість його підробки з боку прикладного коду. Двовіконна архітектура моделює цей принцип: застосунок лише публікує події, а монітор незалежно їх аналізує. Обидва вікна показано поряд на рисунку 3.1.

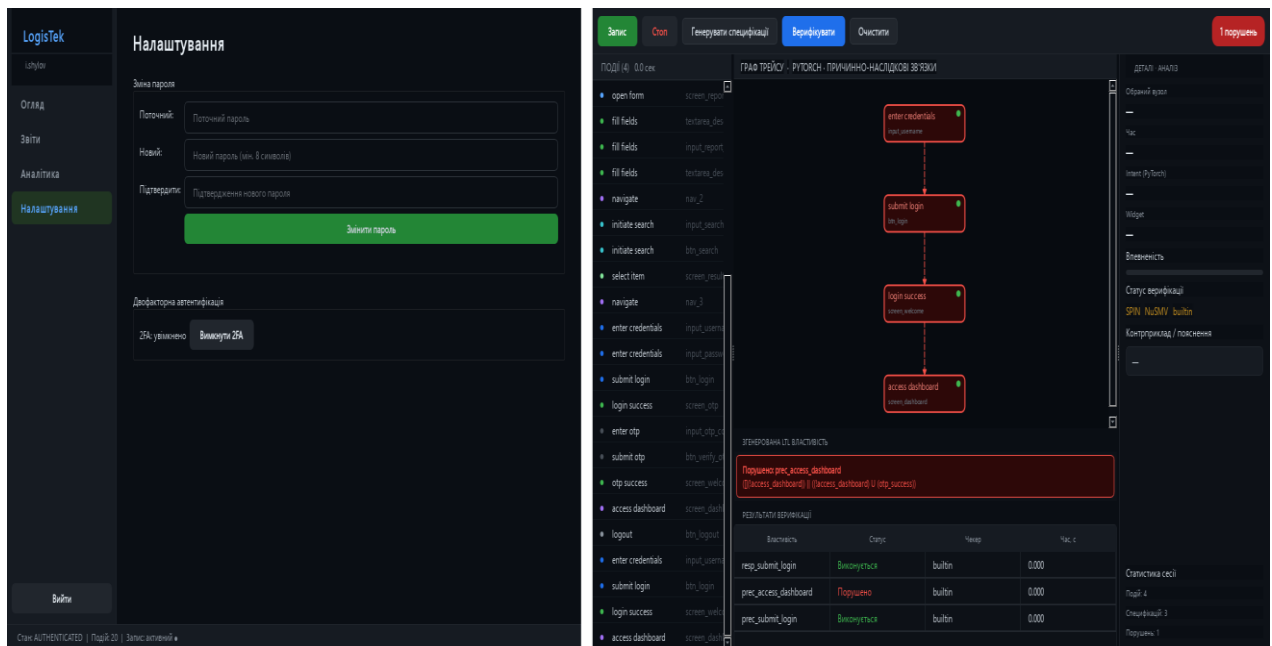


Рисунок 3.1 – Двовіконна система: ліворуч – бізнес-застосунок, праворуч – монітор верифікації, що в реальному часі відображає трейс і результати перевірки

Зв'язок між вікнами реалізовано через шину подій – об'єкт із сигналами фреймворку Qt, що передає появу нової події, зміну сесії, готовий звіт верифікації та зміну стану запису. Завдяки механізму сигналів і слотів застосунок – джерело й монітор – приймач залишаються повністю розв'язаними. Архітектура допускає й рознесення застосунків на окремі процеси чи машини: достатньо замінити внутрішню шину сигналів на міжпроцесний транспорт, не змінюючи самі застосунки, оскільки вони взаємодіють лише через абстракцію публікації/підписки. Це підтверджує

гнучкість рішення й можливість масштабування на реальні розподілені системи.

3.4 Шари захоплення, нормалізації та збагачення

Захоплення подій реалізовано глобальним фільтром подій, що перехоплює взаємодії на рівні застосунку та формує об'єкти сирих подій. Для коректної прив'язки наміру кожен значущий елемент інтерфейсу позначається інформативним об'єктним іменем і, за потреби, явною міткою наміру. Це підвищує точність подальшої нормалізації й робить захоплення стійким до косметичних змін інтерфейсу. Принциповим є те, що захоплення відбувається прозоро для застосунку – воно не вимагає змін у бізнес – логіці й не впливає на роботу користувача.

Нормалізація перетворює сиру подію на подію з наміром. Базовий рівень – функція слабкого маркування, що містить набори правил для авторизаційних екранів і дій, а також токени для розпізнавання введення облікових даних і одноразового коду. За наявності навченої моделі результат правил уточнюється класифікатором. Після нормалізації працює семантичний збагачувач, що згортає поспіль однакові наміри, розв'язує невідомі наміри за контекстом сусідів та позначає фази взаємодії. Цей етап критично важливий для якості верифікації: реальний потік подій сповнений технічного шуму, і без його усунення граф трейсу був би переобтяжений несуттєвими деталями, що ускладнило б як автоматичний аналіз, так і сприйняття результату людиною.

3.5 Шари генерації специфікацій та верифікації

Будівник трейсу формує каузальний граф за послідовністю намірів. Генератори створюють формальні властивості за канонічними шаблонами. Для авторизації введено спеціальні правила відгуку й передування: доступ

до захищеного розділу має передуватися успішним другим фактором, а спроба входу – обов’язково завершуватися результатом. Генератор гарантує унікальність імен властивостей і додає україномовні підписи намірів для зрозумілих пояснень.

Верифікатор перевіряє специфікації проти моделі поведінки, побудованої з трейсу. Підтримуються три бекенди: вбудований, SPIN та NuSMV; вибір залежить від доступності зовнішніх інструментів. Знайдені порушення обробляє аналізатор контрприкладів, що формує пояснення за шаблонами, оцінює серйозність та, за наявності, залучає зовнішній сервіс для пояснення природною мовою. Розмежування рівнів захисту тут принципове: служба автентифікації блокує небезпечні дії під час виконання, тоді як верифікатор надає формальний доказ коректності або порушення. Поєднання двох рівнів дає й запобігання, і верифіковану гарантію – навіть якщо в runtime – логіці допущено помилку, монітор незалежно зафіксує порушення темпоральної властивості.

3.6 Шар зберігання та надійність

Персистентність забезпечує модуль на основі SQLite. Зберігаються сесії, сирі та нормалізовані події, специфікації й результати верифікації, що дозволяє повністю відтворити будь-яку сесію та її верифікацію постфактум. Надійність системи забезпечується кількома проєктними рішеннями. Цілісність даних гарантується механізмом, що унеможливорює запис подій без батьківського запису сесії. Стійкість до некоректних даних забезпечує валідація на межах шарів. Стійкість інтерфейсу досягається винесенням обчислень у фоновий потік, тож відмова під час верифікації не «підвішує» вікно монітора.

Окремим проєктним принципом є м’яка деградація: система зберігає працездатність за відсутності будь-якої зовнішньої залежності. Якщо немає навченої моделі – нормалізація спирається на правила; якщо немає

зовнішніх верифікаторів – працює вбудований; якщо немає зовнішнього мовного сервісу – пояснення формуються за шаблонами. Це гарантує відтворюваність базової поведінки незалежно від оточення, що критично для дослідницької системи, результати якої мають бути перевірюваними. Така автономність усуває проблему "нестабільності середовища", коли оновлення або недоступність сторонніх API унеможлиблює точне повторення експерименту.

3.7 Компонентний погляд на систему

З погляду компонентної декомпозиції система складається з пакетів: реєстр моделей даних, служба автентифікації зі сховищем користувачів, пакет трансформації (нормалізатор, збагачувач, будівник трейсу), генератори специфікацій, пакет верифікації (модель поведінки, перевірка моделей, аналіз контрприкладів), зберігання, графічний інтерфейс обох застосунків, комунікаційна шина, оркестратор шарів та інтерфейс командного рядка. Залежності спрямовані від прикладних пакетів до реєстру схем, який не залежить ні від кого, що відповідає принципу інверсії залежностей і робить реєстр схем стабільним ядром системи. Така організація узгоджується з усталеними практиками проєктування супроводжуваних систем і полегшує подальший розвиток.

Додатково, виділення окремого пакета трансформації дозволяє чітко розмежувати обробку сирих подій та їх семантичну інтерпретацію, що сприяє повторному використанню компонентів і спрощує тестування. Пакет верифікації ізольовано інкапсулює логіку формальних перевірок, що дає змогу розширювати підтримку нових формалізмів (зокрема LTL, CTL, TLA+) без впливу на інші частини системи. Комунікаційна шина забезпечує слабке зв'язування між модулями, підтримуючи асинхронну взаємодію та масштабованість.

Оркестратор шарів координує послідовність виконання етапів обробки – від збору подій до отримання результатів верифікації – забезпечуючи контроль потоків даних і узгодженість станів. Наявність як графічного інтерфейсу, так і інтерфейсу командного рядка підвищує гнучкість використання системи для різних категорій користувачів – від дослідників до інженерів – практиків.

У фізичній структурі проєкту перелічені пакети відображені каталогами `src/uiformal/`: `schema` – реєстр моделей, `auth` – служба автентифікації, `capture` і `transform` – захоплення та нормалізація подій, `specs` і `verify` – генерація та перевірка специфікацій, `ui` – графічні оболонки обох застосунків, `pipeline` і `cli` – оркестрація та запуск. Таке відображення «логічний компонент – програмний модуль» зменшує семантичний розрив між архітектурною схемою та кодом і спрощує супровід: зміни на рівні одного шару локалізуються в межах відповідного пакета без каскадного впливу на решту системи. Окремо варто наголосити, що `App` і `uiformal Monitor` спільно використовують однакові компоненти конвеєра, відрізняючись лише прикладною логікою та способом представлення результатів, що підтверджує доцільність двовіконної архітектури, описаної у підрозділі 3.3.

Компонентна організація безпосередньо підтримує перевірюваність розробки: модульні тести охоплюють автентифікацію, нормалізацію, генерацію специфікацій і верифікацію окремо, а інтеграційні сценарії перевіряють узгоджену роботу оркестратора. Завдяки цьому можна ізольовано змінювати.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Технологічний стек

Систему реалізовано мовою Python 3 із такими ключовими бібліотеками: PySide6 [15] – офіційні прив'язки Qt 6 для побудови графічного інтерфейсу обох застосунків та механізму сигналів і слотів комунікаційної шини; Pydantic [16] – реєстр схем даних; PyTorch [11] – класифікатор намірів і оцінка серйозності контрприкладів; NetworkX [17] – побудова та аналіз каузального графа трейсу; SQLite – персистентне зберігання. Зовнішні опційні залежності – SPIN [4] та NuSMV [5] – використовуються для перевірки моделей.

Вибір Python зумовлено зрілою екосистемою наукових і прикладних бібліотек та зручністю інтеграції шарів; вибір Qt – кросплатформністю та потужним механізмом подій, потрібним для захоплення взаємодій. Цей стек є усталеним для дослідницьких систем, де поєднуються інтерактивний інтерфейс, обробка даних і елементи машинного навчання, і забезпечує баланс між швидкістю розробки та продуктивністю.

4.2 Реалізація застосунку

LogisTek App – повноцінний бізнес – застосунок із кількома розділами даних та різноманітною логікою. Він спроектований так, щоб відтворювати типову структуру корпоративної системи: автентифікацію, оглядову панель, роботу з даними, аналітику та налаштування облікового запису. Така повнота важлива для дослідження, бо лише на достатньо насиченому застосунку можна продемонструвати, що метод верифікації працює не на штучних прикладах, а на реалістичному наборі взаємодій.

Робота починається з екрана входу (рисунок 4.1), де користувач вводить логін і пароль. Усі взаємодії захоплюються та публікуються у шину

подій. Після успішної перевірки пароля застосунок переходить до етапу двофакторної автентифікації (рисунок 4.2), де вводиться одноразовий код. Лише проходження цього кроку відкриває доступ до захищеної частини застосунку – це правило згодом перевіряється монітором як темпоральна властивість передування.

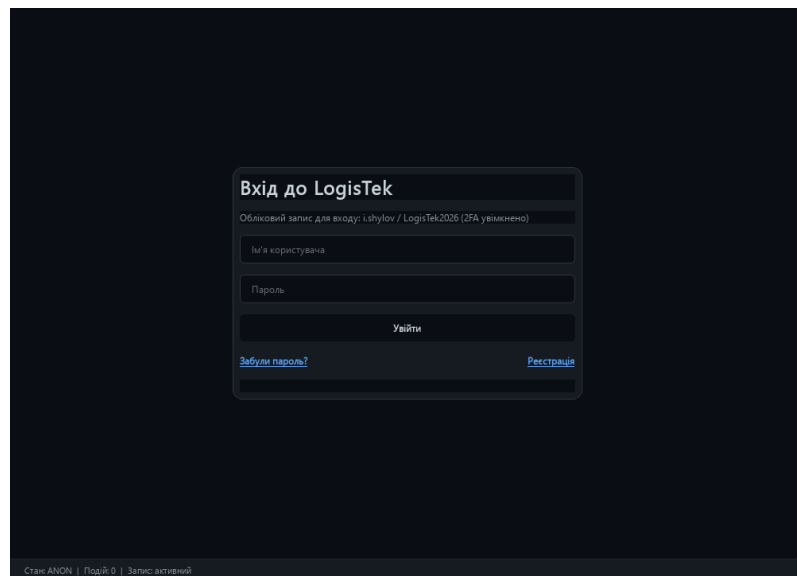


Рисунок 4.1 – Екран входу: введення логіна й пароля; кожне натискання та введення тексту формує сиру подію інтерфейсу

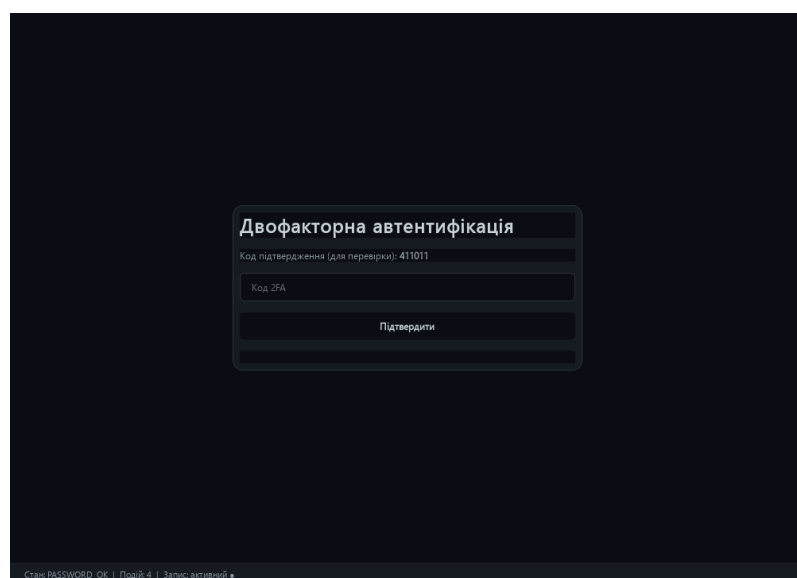


Рисунок 4.2 – Етап другого фактора: введення одноразового коду TOTP

Центральним екраном застосунку є розділ «Огляд» зі зведеними показниками та стрічкою останньої активності (рисунок 4.3). Показники й стрічка обчислюються динамічно за фактичними захопленими подіями поточної сесії, відображаючи реальний стан роботи, а не наперед задані значення. Це робить оглядову панель прозорою щодо того, які дані фактично спостерігає система.

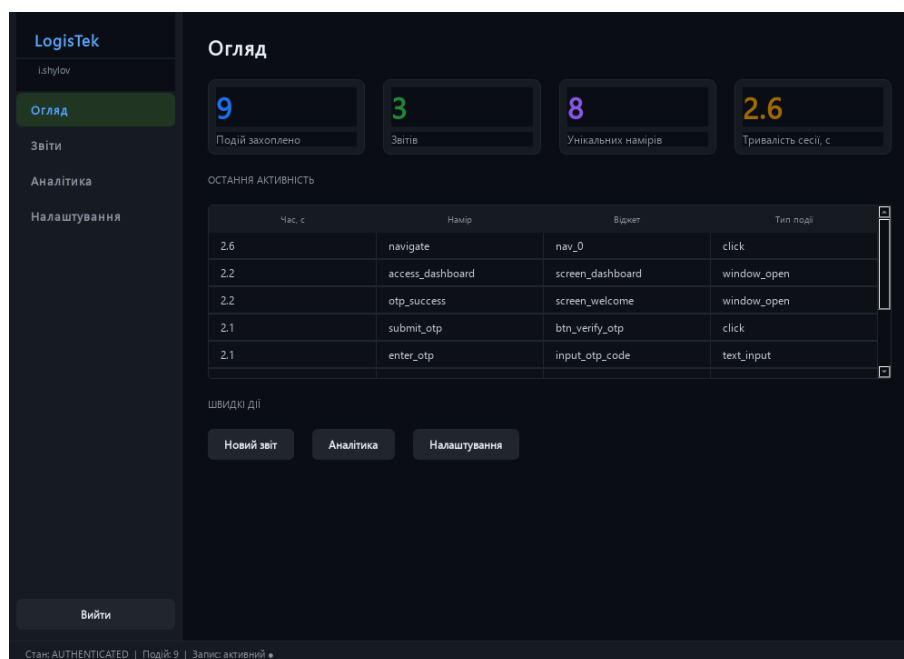


Рисунок 4.3 – Оглядова панель: зведені показники сесії та стрічка останньої активності, що формуються з реально захоплених подій

Таким чином, оглядова панель виконує не лише інформаційну, а й демонстраційну функцію: користувач бачить, що кожна дія в інтерфейсі одразу потрапляє до потоку спостереження й може бути використана для подальшої формальної перевірки. Саме тому розділ «Огляд» слід розглядати як перший робочий екран після успішної автентифікації, з якого починається типова прикладна взаємодія, що далі відображається у моніторі *uniform Monitor*.

Розділ «Звіти» реалізує операції CRUD над звітами (рисунки 4.4–4.5). Робота зі звітом охоплює відкриття форми, заповнення полів і збереження

або скасування – характерну послідовність, що дозволяє монітору перевіряти, зокрема, властивість живучості: відкрита форма зрештою має бути або підтверджена, або скасована. Розділ «Аналітика» (рисунки 4.6) виконує пошук і фільтрацію подій; у таблиці результатів відображаються розпізнані наміри, що демонструє роботу шару нормалізації безпосередньо у прикладному інтерфейсі. Розділ «Налаштування» (рисунки 4.7) містить керування обліковим записом: зміну пароля та двофакторної автентифікації.

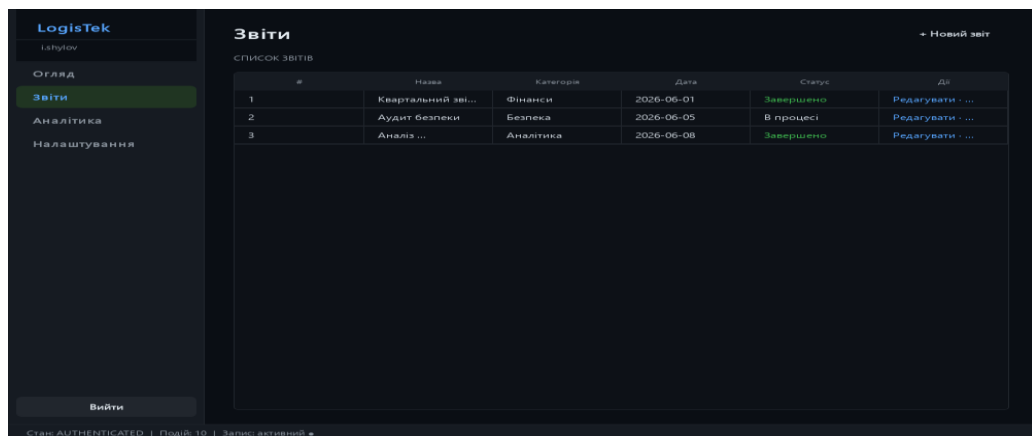


Рисунок 4.4 – Розділ «Звіти»: перелік наявних звітів з можливістю створення, редагування й видалення

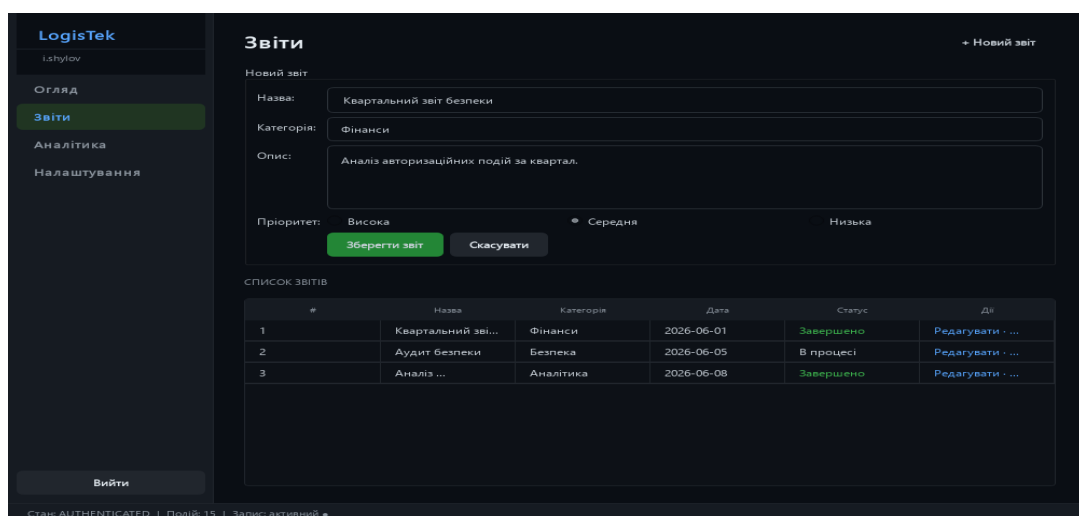


Рисунок 4.5 – Форма звіту: відкриття форми, заповнення полів та збереження утворюють характерну послідовність намірів

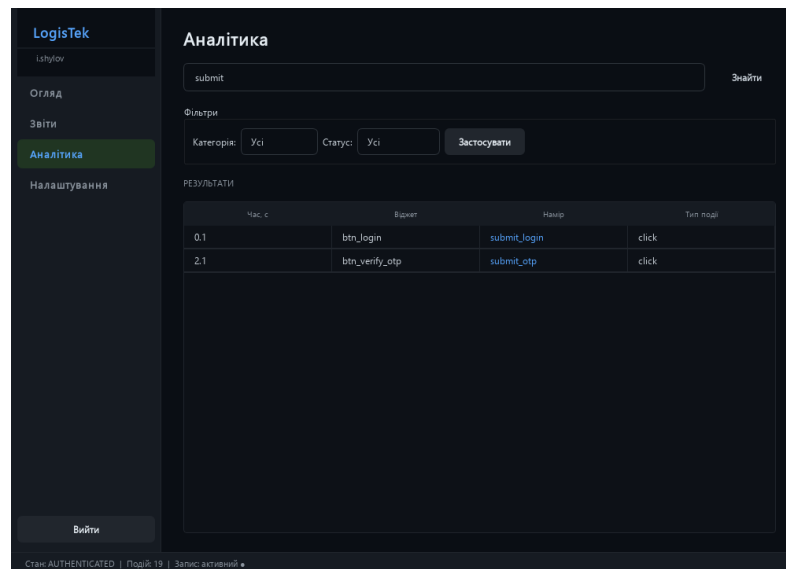


Рисунок 4.6 – Аналітика: пошук і фільтрація захоплених подій з відображенням розпізнаних намірів

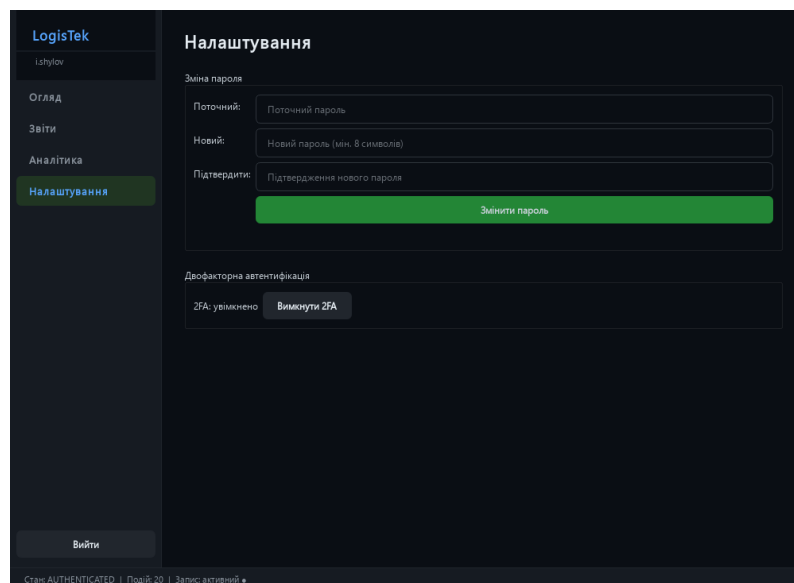


Рисунок 4.7 – Налаштування: зміна пароля та керування двофакторною автентифікацією

4.3 Підсистема безпеки

Серверну логіку авторизації інкапсульовано у двох компонентах. Сховище користувачів відповідає за зберігання та криптографію: паролі

зберігаються не у відкритому вигляді, а як похідні ключі, отримані функцією PBKDF2-HMAC-SHA256 із випадковою сіллю

$$dk = \text{PBKDF2}(\text{pwd}, \text{salt}, c, dkLen), \quad (4.1)$$

де c – кількість ітерацій.

Для кожного користувача генерується секрет TOTP, на основі якого обчислюються одноразові коди. Застосування ітеративної функції формування ключа замість простого хешування є усталеною практикою захисту паролів, що істотно ускладнює атаки перебором за словником навіть у разі витоку бази даних.

Служба автентифікації реалізує скінченний автомат: реєстрація (рисунок 4.8), вхід, перевірка другого фактора, політика блокування після перевищення ліміту невдалих спроб (рисунок 4.9), керування сесією. Політика блокування є практичним механізмом протидії атакам перебору: після кількох невдалих спроб обліковий запис тимчасово блокується, що унеможливлює автоматизований підбір пароля.

Рисунок 4.8 – Реєстрація: створення облікового запису з похідним ключем пароля та секретом TOTP

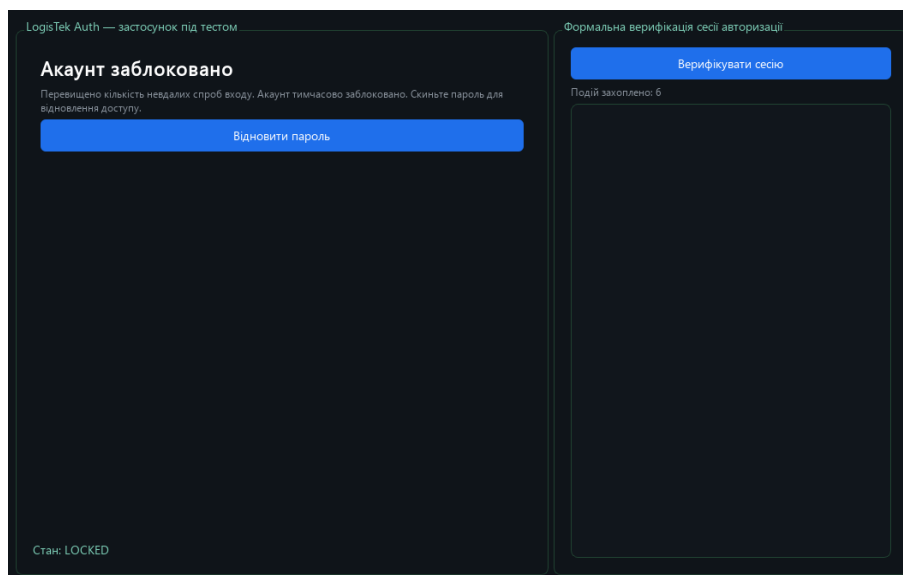


Рисунок 4.9 – Спрацювання політики блокування після перевищення ліміту невдалих спроб входу

4.4 Реалізація монітора верифікації

uniform Monitor повторює трипанельну компоновку прототипу, але з реальним конвеєром. Ліва панель – стрічка подій із кольоровим кодуванням типів. Центральна панель – граф трейсу, побудований на основі топологічних генерацій; порушення виділяються червоним. Нижня частина показує згенеровану LTL властивість і таблицю результатів верифікації. Права панель – деталі та аналіз: намір з імовірністю від класифікатора, статуси верифікаторів і пояснення контрприкладу. Панель інструментів реалізує робочий процес «почати запис → зупинити → згенерувати специфікації → верифікувати», а бейдж сумарної кількості порушень дає аналітику миттєвий індикатор стану безпеки сесії.

Для коректного відгуку інтерфейсу конвеєр виконується у фоновому потоці, тож тривала верифікація не блокує UI монітора. Інтерфейс обох застосунків витримано в єдиній темній колірній схемі з акцентними кольорами для статусів (зелений – успіх, червоний – порушення), що знижує зорове навантаження під час тривалого аналізу й реалізує принцип

передувати важливого, коли критична інформація сприймається до свідомого аналізу.

Рисунок 4.10 демонструє монітор для коректної авторизованої сесії: усі вершини графа в нормальному стані, бейдж показує нуль порушень, а таблиця результатів – статус виконання для всіх властивостей. Рисунок 4.11 показує монітор для небезпечної сесії з обходом другого фактора: вершини виділено червоним, унизу – порушена властивість передування, а в таблиці – статус порушення.

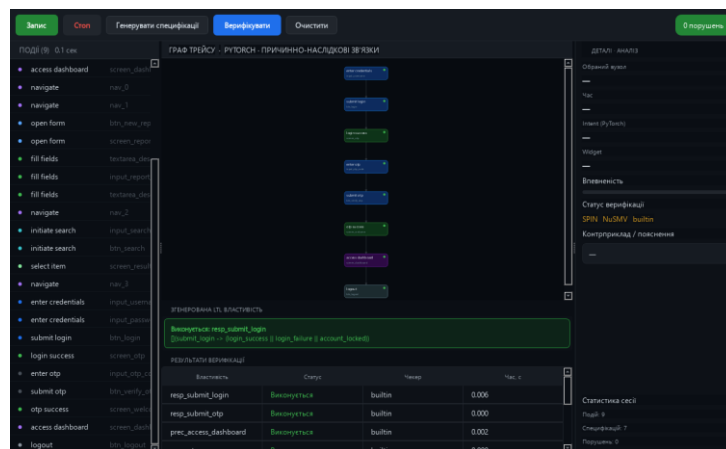


Рисунок 4.10 – Монітор для коректної сесії: повний ланцюг автентифікації пройдено, усі властивості виконуються

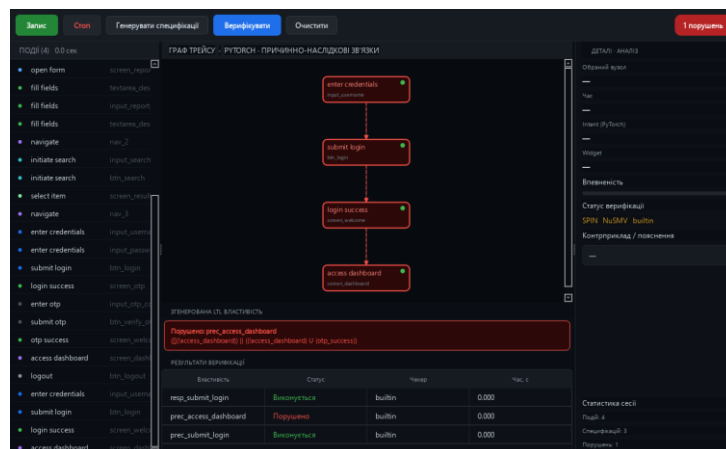


Рисунок 4.11 – Монітор для небезпечної сесії: доступ до захищеного розділу досягнуто без успішного другого фактора

4.5 Інтерфейс командного рядка та запуск

Систему обладнано інтерфейсом командного рядка з командами демонстрації синтетичних сценаріїв та одночасного запуску обох вікон. Пакет встановлюється у режимі редагування, що робить модуль системи доступним без ручного налаштування шляхів. Для коректного відображення кирилических символів у консолі налаштовано кодування UTF-8. Усі підписи інтерфейсу та пояснення порушень виконано українською мовою; систему спроектовано розширюваною – щоб додати нову предметну область чи правило безпеки, достатньо доповнити перелік намірів, додати правила нормалізації та зареєструвати новий шаблон у генераторі специфікацій, не змінюючи інші шари завдяки контрактам реєстру схем.

4.6 Фрагменти коду програмної реалізації

Нижче наведено вибірккові фрагменти коду, що ілюструють ключові частини застосунку. Лістинги не дублюють увесь проєкт, а показують саме ті місця, де реалізовано основну логіку: захист паролів і TOTP, автомат автентифікації, захоплення подій інтерфейсу, генерацію LTL-властивостей та перевірку шаблону передування (лістинги 4.1–4.6).

Лістинг 4.1 – Фрагмент реалізації захищеного зберігання паролів у модулі `uiformal.auth.store`

```
def hash_password(password: str, salt: bytes | None = None) ->
tuple[str, str]:
    salt = salt or os.urandom(_SALT_BYTES)
    dk = hashlib.pbkdf2_hmac("sha256", password.encode("utf-
8"), salt, _PBKDF2_ROUNDS)
    return salt.hex(), dk.hex()
def verify_password(password: str, salt_hex: str,
expected_hash_hex: str) -> bool:
```

Продовження лістингу 4.1

```
_, got = hash_password(password,
bytes.fromhex(salt_hex))
return hmac.compare_digest(got, expected_hash_hex)
```

Лістинг 4.2 – Перевірка першого фактора та політики блокування у AuthService

```
def login(self, ctx: AuthContext, username: str, password:
str) -> AuthOutcome:
    user = self.store.get(username)
    if user is None:
        ctx.last_outcome = AuthOutcome.NO_SUCH_USER
        return AuthOutcome.NO_SUCH_USER
    now = self._now()
    if user.locked_until > now:
        ctx.state = AuthState.LOCKED
        ctx.last_outcome = AuthOutcome.ACCOUNT_LOCKED
        return AuthOutcome.ACCOUNT_LOCKED
    if not verify_password(password, user.pwd_salt,
user.pwd_hash):
        attempts = user.failed_attempts + 1
        if attempts >= MAX_FAILED_ATTEMPTS:
            self.store.update_security(
                username, failed_attempts=attempts,
locked_until=now + LOCKOUT_SECONDS
            )
            ctx.state = AuthState.LOCKED
            ctx.last_outcome = AuthOutcome.ACCOUNT_LOCKED
            return AuthOutcome.ACCOUNT_LOCKED
        self.store.update_security(username,
failed_attempts=attempts, locked_until=0)
        ctx.last_outcome = AuthOutcome.BAD_CREDENTIALS
        return AuthOutcome.BAD_CREDENTIALS
    self.store.update_security(username, failed_attempts=0,
locked_until=0)
```

Продовження лістингу 4.2

```
ctx.username = username
if user.totp_enabled:
    ctx.state = AuthState.PASSWORD_OK
    ctx.last_outcome = AuthOutcome.REQUIRES_OTP
    return AuthOutcome.REQUIRES_OTP
ctx.state = AuthState.AUTHENTICATED
ctx.last_outcome = AuthOutcome.OK
return AuthOutcome.OK
```

Лістинг 4.3 – Центральна перевірка авторизації перед доступом до захищеної зони

```
def issue_session(self, ctx: AuthContext) -> str | None:
if ctx.state != AuthState.AUTHENTICATED:
    ctx.last_outcome = AuthOutcome.NOT_AUTHENTICATED
    return None
ctx.session_token = new_session_token()
return ctx.session_token
```

Лістинг 4.4 – Формування сирого події інтерфейсу у глобальному Qt-фільтрі

```
def _record(self, obj, event, mapped: EventType) -> None:
    widget_name = self._widget_name(obj)
    ctx = self._dom_context(obj)
    payload: dict = {}
    if mapped == EventType.KEY_PRESS and hasattr(obj, "text"):
        payload["text_len"] = len(obj.text() or "")
        mapped = EventType.TEXT_INPUT
    self.store.add(
        RawUIEvent(
            session_id=self.store.session.session_id,
            type=mapped,
            widget=widget_name,
            timestamp=time.time(),
```

Продовження лістингу 4.4

```

        dom_context=ctx,
        payload=payload,
    )
)

```

Лістинг 4.5 – Генерація LTL-формул за шаблонами відгуку та передування

```

if p.kind == PatternKind.RESPONSE:
    disj = " || ".join(p.targets)
    f = f"[]({p.trigger} -> <>({disj}))"
    return f, {"trigger": p.trigger, "targets": p.targets}
if p.kind == PatternKind.PRECEDENCE:
    befcorm = " || ".join(p.befores)
    f = f"([](!{p.after})) || ((!{p.after}) U ({befcorm}))"
    return f, {"befores": p.befores, "after": p.after}

```

Лістинг 4.6 – Виявлення порушення передування у вбудованому верифікаторі

```

def _check_precedence(self, spec: Specification) ->
tuple[VerificationStatus, list[str]]:
    befores = set(spec.metadata["befores"])
    after = spec.metadata["after"]
    path = self.model.path_from_initial(after,
forbidden=befores)

```

Наведені фрагменти показують, що контроль доступу реалізовано на двох рівнях. На рівні бізнес – застосунку метод `issue_session` не видає токен без повної автентифікації, а на рівні монітора незалежний верифікатор перевіряє, чи в спостереженому трейсі не з'явився шлях до `access_dashboard` без попереднього `otp_success`. Саме ця подвійна схема забезпечує і практичний runtime – захист, і формально обґрунтований висновок про коректність поведінки.

5 ВЕРИФІКАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

5.1 Сценарії дослідження та модель загроз

Для оцінки системи розроблено сім сценаріїв, поділених на дві групи. Перша група охоплює авторизацію: коректний сценарій (повне проходження входу й другого фактора з доступом до захищеного розділу); прямий доступ (спроба доступу без жодної автентифікації); обхід другого фактора (успішний логін, але доступ в обхід другого фактора); атака перебору (серія невдалих спроб входу, що має призводити до блокування). Друга група перевіряє загальні шаблони поведінки інтерфейсу: коректне відправлення форми, дефект «завислого» відправлення без результату та сценарій пошуку з вибором результату.

Перед формулюванням властивостей корисно явно описати модель загроз, які система має виявляти представлено в таблиці 5.1.

Таблиця 5.1 – Модель загроз авторизації та відповідні темпоральні властивості

Загроза	Опис	Порушувана властивість	Виявлення
Прямий доступ	Доступ без автентифікації	передування	так
Обхід 2FA	Доступ без другого фактора	передування	Так
Перебір	Серія невдалих спроб входу	політика блокування	Так
«Зависла» спроба	Спроба входу без результату	відгук	так
Передчасний другий фактор	Код до успішного логіну	передування	так

Модель загроз показує, що типові атаки й дефекти контролю доступу систематично відображаються в порушенні базових властивостей

передування та відгуку, що підтверджує адекватність обраного набору шаблонів специфікацій для предметної області.

5.2 Темпоральні властивості безпеки

Для предметної області авторизації система автоматично формує, зокрема, такі властивості. Передування доступу другому фактору – доступ можливий лише після успішного другого фактора:

$$G(\neg \text{access_dashboard}) \vee (\neg \text{access_dashboard} U \text{otp_success}). \quad (5.1)$$

Відгук на спробу входу – за кожною спробою настає результат:

$$G(\text{submit_login} \rightarrow F(\text{login_success} \vee \text{login_failure} \vee \text{account_locked})). \quad (5.2)$$

Передування другого фактора логіну та відгук на другий фактор:

$$(\neg \text{submit_otp} U \text{login_success}) \vee G(\neg \text{submit_otp}), \quad (5.3)$$

$$G(\text{submit_otp} \rightarrow F(\text{otp_success} \vee \text{otp_failure})). \quad (5.4)$$

Порушення властивості передування доступу автоматично виявляється у сценаріях обходу другого фактора та прямого доступу.

5.3 Результати верифікації

Зведені результати прогону конвеєра на семи сценаріях наведено в таблиці 5.2. У таблиці LTL/CTL/TLA+ означає кількість специфікацій,

згенерованих у відповідних формалізмах для конкретного трейсу. Час наведено для повного конвеєра: нормалізація, збагачення, побудова трейсу, генерація специфікацій, верифікація та аналіз контрприкладів. Кількісні результати прогону конвеєра на сценаріях наведено у таблиці 5.2.

Таблиця 5.2 – Кількісні результати прогону конвеєра на сценаріях авторизації та загальної взаємодії з інтерфейсом

Сценарій	Подій	Нормалізовано	LTL	LA+	Порушень	Час, мс	Порушена властивість
Коректна авторизація	9	8	7	7	0	7,718	–
Прямий доступ	2	2	1	1	1	1,587	prec access dashboard
Обхід 2FA	4	4	3	3	1	0,955	prec access dashboard
Перебір пароля	9	9	4	4	0	1,093	–
Коректна форма	8	6	5	5	0	1,193	–
Форма без результату	6	5	5	5	1	1,298	resp_submit_form
Пошук	4	2	1	1	0	0,506	–

Результат коректного сценарію підтверджується рисунком 4.10 (нуль порушень), а виявлення обходу другого фактора – рисунком 4.11 (червоне виділення й порушення властивості передування).

Прямий доступ і обхід 2FA порушують одну й ту саму ключову властивість `prec_access_dashboard`, але мають різні контрприклади: у першому випадку доступ виникає без будь-якої автентифікації, у другому – після першого фактора, але без `otp_success`.

Окремо сценарій «Форма без результату» демонструє не безпекову, а функціональну властивість живучості: після `submit_form` не з'являється допустимий результат, тому порушується `resp_submit_form`. Це показує, що конвеєр застосовний не лише до авторизації, а й до ширших класів UI – взаємодій.

Час повного прогону для кожного з них залишається на рівні од-восьми мілісекунд, що підтверджує придатність підходу для аналізу сесії без помітного впливу на роботу інтерфейсу.

5.4 Покроковий розбір сценаріїв та аналіз контрприкладів

Для коректного сценарію послідовність намірів охоплює повний ланцюг від введення облікових даних до доступу через успішний другий фактор; усі властивості виконуються. Для прямого доступу властивість передування порушується одразу, бо не існує входження успішного другого фактора, що передує доступу. Для обходу другого фактора логін успішний, але крок другого фактора пропущено, і властивість передування порушується. Для перебору після перевищення ліміту спроб спрацьовує політика блокування, а монітор фіксує послідовність перебору в трейсі. Важливо, що сценарій перебору не класифікується як порушення верифікації, оскільки очікувана політика блокування виконується: кожна спроба входу має результат, а після третьої невдачі з'являється `account_locked`.

Цей розбір показує, що різні класи загроз відображаються в порушенні різних темпоральних властивостей, а отже система здатна не лише виявляти факт небезпеки, а й класифікувати її природу. Для кожного порушення аналізатор формує пояснення зрозумілою мовою (наприклад, що доступ отримано в обхід другого фактора), додає трейс – контрприклад та оцінку серйозності. Таке пояснення робить результат верифікації придатним не лише для інженера-формаліста, а й для прикладного розробника або аналітика безпеки.

5.5 Оцінка продуктивності та відтворюваність

Завдяки коротким формулам і скінченним трейсам верифікація однієї сесії виконується за частки секунди на вбудованому верифікаторі. У проведених прогонах повний час конвеєра становив від 0,506 мс для найпростішого сценарію пошуку до 7,718 мс для повного сценарію авторизації з сімома LTL властивостями та відповідними CTL/TLA+–

поданнями. Винесення конвеєра у фоновий потік забезпечує плавність інтерфейсу незалежно від обсягу трейсу.

Усі наведені результати відтворювані. Синтетичні сценарії детерміновані й не залежать від випадковості чи зовнішніх служб. Базовий рівень нормалізації спирається на правила, тому інтерпретація намірів однакова за будь-якого запуску. Вбудований верифікатор не потребує зовнішніх інструментів, що усуває залежність від оточення. Завдяки принципу м'якої деградації результати однаково відтворюються як на повній конфігурації, так і на мінімальній. Описаний комплекс властивостей забезпечує інваріантність функціонування системи щодо умов обчислювального середовища.

5.6 Тестування та загрози достовірності

Коректність ключових компонентів підтверджено модульними та інтеграційними тестами: перевірено реєстрацію, хешування пароля, роботу другого фактора, політику блокування, роботу нормалізатора, генерацію специфікацій для кількох логік, побудову Promela – подання та виявлення порушень безпеки для відповідних сценаріїв. Під час контрольного запуску тестового набору виконано модульні тести, усі завершилися успішно.

Слід відзначити загрози достовірності результатів. Внутрішня валідність: коректність висновку залежить від точності нормалізації намірів, тому базовий рівень спирається на детерміновані правила. Зовнішня валідність: сценарії синтетичні й не вичерпують усіх реальних атак, проте покривають канонічні класи порушень контролю доступу. Конструктна валідність: обрані властивості відповідають загальновизнаним шаблонам [6], що зменшує ризик неадекватної формалізації.

ВИСНОВКИ

У роботі розв'язано актуальну задачу подолання семантичного розриву між низькорівневими подіями графічного інтерфейсу та високорівневими формальними вимогами безпеки на прикладі авторизації.

Отримано такі основні результати:

- проаналізовано предметну галузь і виявлено, що жоден із суміжних підходів не закриває повний ланцюг автоматичної верифікації авторизації за спостережуваною поведінкою ;

- спроектовано багатошаровий конвеєр із шести шарів зі слабкою зв'язністю на основі єдиного реєстру схем даних ;

- реалізовано гібридний шар нормалізації, що зіставляє подіям прикладні наміри на основі правил та опційного класифікатора, із гарантованою відтворюваністю базової поведінки ;

- реалізовано автоматичну генерацію темпоральних специфікацій LTL/CTL/TLA+ за канонічними шаблонами та їх верифікацію вбудованим верифікатором і зовнішніми SPIN/NuSMV з м'якою деградацією ;

- розроблено двовіконну систему «бізнес – застосунок + монітор» із комунікаційною шиною, що моделює об'єктивне спостереження за поведінкою користувача ;

- реалізовано підсистему безпеки на основі PBKDF2 і TOTP та аналізатор контрприкладів, що пояснює порушення зрозумілою мовою ;

- експериментально доведено на семи сценаріях, що система коректно відрізняє безпечні авторизаційні потоки від небезпечних, автоматично виявляє обхід другого фактора, прямий доступ і дефект відсутнього відгуку форми, а також підтверджує коректне спрацювання політики блокування під час перебору .

Практична цінність роботи полягає в тому, що вона робить формальну верифікацію авторизації доступною прикладному розробнику: специфікації генеруються автоматично, а порушення пояснюються природною мовою.

Напрями подальшого розвитку – підвищення точності нормалізації намірів, інтеграція з реальними вебзастосунками, розширення набору шаблонів безпеки та інтеграція методів машинного навчання для виявлення складніших патернів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Clarke E. M., Grumberg O., Peled D. A. Model checking. 2nd ed. Cambridge: MIT Press, 2018. 314 p.
2. Baier C., Katoen J.-P. Principles of model checking. Cambridge: MIT Press, 2008. 975 p.
3. Pnueli A. The temporal logic of programs. *18th annual symposium on foundations of computer science (sfcs 1977)*, Providence, RI, USA, 31 October – 2 November 1977. 1977. URL: <https://doi.org/10.1109/sfcs.1977.32> (date of access: 04.07.2026).
4. GitHub - open-spaced-repetition/srs-benchmark: a benchmark for spaced repetition schedulers/algorithms. *GitHub*. URL: <https://github.com/open-spaced-repetition/srs-benchmark> (date of access: 03.07.2026).
5. Clarke E. M., Henzinger T. A., Veith H., Bloem R. Handbook of model checking. Cham: Springer, 2018. 1234 p.
6. Dwyer M. B., Avrunin G. S., Corbett J. C. Patterns in property specifications for finite-state verification. *The 21st international conference*, Los Angeles, California, United States, 16–22 May 1999. New York, New York, USA, 1999. URL: <https://doi.org/10.1145/302405.302672> (date of access: 04.07.2026).
7. A01:2021 – broken access control. *OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation*. URL: https://owasp.org/Top10/A01_2021-Broken_Access_Control/ (date of access: 03.07.2026).
8. M'Raihi D., Machani S., Pei M., Rydell J. TOTP: time-based one-time password algorithm. 2011. URL: <https://www.rfc-editor.org/rfc/rfc6238> (date of access: 28.06.2026).
9. van der Aalst W. Process mining. Berlin, Heidelberg : Springer Berlin Heidelberg, 2016. URL: <https://doi.org/10.1007/978-3-662-49851-4> (date of access: 04.07.2026).

10. Leucker M., Schallhart C. A brief account of runtime verification. *The journal of logic and algebraic programming*. 2009. Vol. 78, no. 5. P. 293–303. URL: <https://doi.org/10.1016/j.jlap.2008.08.004> (date of access: 04.07.2026).
11. PyTorch: an imperative style, high-performance deep learning library / A. Paszke et al. *arXiv*. 2019. URL: <https://doi.org/10.48550/arXiv.1912.01703> (date of access: 04.07.2026).
12. Lamport L. Specifying systems: the TLA+ language and tools. Boston: Addison-Wesley, 2002. URL: <https://lamport.azurewebsites.net/tla/book.html> (date of access: 28.06.2026).
13. The selenium browser automation project. *Selenium WebDriver Documentation*. URL: <https://www.selenium.dev/documentation/> (date of access: 28.06.2026).
14. Fast and reliable end-to-end testing for modern web apps. *Playwright*. URL: <https://playwright.dev/> (date of access: 03.07.2026).
15. Qt for Python. *Qt Documentation*. URL: <https://doc.qt.io/qtforpython/> (date of access: 03.07.2026).
16. Pydantic documentation. *Pydantic*. URL: <https://docs.pydantic.dev/> (date of access: 28.06.2026).
17. Software for complex networks. *NetworkX Documentation*. URL: <https://networkx.org/documentation/stable/> (date of access: 28.06.2026).
18. Huth M., Ryan M. Logic in computer science: modelling and reasoning about systems. 2nd ed. Cambridge: Cambridge University Press, 2004. 427 p.
19. Introduction • SvelteKit Docs. *Svelte • Web development for the rest of us*. URL: <https://kit.svelte.dev/docs> (date of access: 03.07.2026).