НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп’ютерних наук

_____ Голуб Б.Л.

“___” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Розробка програмного модуля комп’ютерної системи контролю доступу до приміщення

Спеціальність 122 – «Комп’ютерні науки»

Гарант освітньої програми

Д.е.н., професор _____ Руденський Р.А.

Керівник бакалаврської кваліфікаційної роботи

Кандидат педагогічних наук, доцент ____ Касаткін Д.Ю.

Виконав _____ Дарчук О.А.

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних наук

_____ Голуб Б.Л.

"___" _____ 2025 р.

.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Дарчук Олег Андрійович

Спеціальність 122 – «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи Розробка програмного модуля комп'ютерної системи контролю доступу до приміщення

затверджена наказом ректора НУБіП України від 25.04.2025р. №699с

Термін подання завершеної роботи на кафедру _____

Вихідні дані до бакалаврської кваліфікаційної роботи: _____

Перелік питань, які потрібно розробити:____

Перелік графічних документів (за потреби)

_____

Дата видачі завдання "_____"_____20___ р.

Керівник бакалаврської кваліфікаційної роботи

Кандидат педагогічних наук, доцент _____ Касаткін Д.Ю.

Завдання прийняв до виконання _____ Дарчук О. А.

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів виконання бакалаврської кваліфікаційної роботи	**Строк виконання етапів бакалаврської кваліфікаційної роботи**	**Примітка**
	Формулювання теми та цілей кваліфікаційної роботи	23.03.2025 – 27.03.2025	
	Аналіз предметної області та аналогічних систем	27.03.2025 – 01.04.2025	
	Розробка вимог до системи	07.04.2025 – 25.04.2025	
	Проектування та реалізація компонентів системи	25.04.2025 – 09.05.2025	
	Тестування, перевірка працездатності системи	16.05.2025 – 18.05.2025	
	Написання пояснювальної записки	18.05.2025 – 23.05.2025	

Студент _____ Дарчук О. А.

Керівник бакалаврської

кваліфікаційної роботи _____ Касаткін Д.Ю.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

АС – автоматизована система

xml – розширювана мова розмітки, Extensible Markup Language

QR – двовимірний штрих-код, quick response

front-end – інтерфейс для взаємодії між користувачем і back end

PHP – скриптова мова програмування, Hypertext Preprocessor

CSS – каскадні таблиці стилів, Cascading Style Sheets

HTML – мова розмітки гіпертексту, HyperText Markup Language

JS – JavaScript

MBSE – системна інженерія на основі моделей, model based systems engineering

СКБД – система керування базами даних

БД – база даних

IOS – мобільна операційна система, iPhone OS

IEEE – інститут інженерів з електротехніки та електроніки, Institute of Electrical and Electronics Engineers

UML – уніфікована мова моделювання, Unified Modeling Language

SysML – мова моделювання загального призначення для застосувань в системній інженерії, Systems Modeling Language

ORM – об'єктно-реляційна проекція, object-relational mapping

HTTP – протокол передачі гіпер-текстових документів, Hyper Text Transfer Protocol

API – програмний інтерфейс застосунку, Application Programing Interface

URL – уніфікований локатор ресурсів, Uniform Resource Locator

IDE – інтегроване середовище розробки, Integrated Development Environment

ВСТУП

Забезпечення безпеки будь-якого об'єкта включає кілька етапів, кількість яких визначається рівнем режимності самого об’єкта. Однак ключовим компонентом завжди залишається система контролю та управління доступом (СКУД). Правильно організована СКУД дозволяє ефективно вирішувати цілий ряд завдань.

Під час впровадження конкретних систем використовуються різноманітні підходи та створюються спеціалізовані пристрої для ідентифікації та аутентифікації. Варто підкреслити, що СКУД є одним із найбільш динамічно розвинутих напрямів ринку безпеки. За оцінками експертів, щорічне зростання цього ринку перевищує 25%. Кількість фахівців, зайнятих у галузі технічних систем безпеки, вже перевищує 500 тисяч осіб. Такий розвиток пов’язаний, з одного боку, зі зростаючим усвідомленням потреб ринку в Україні та попитом на сучасні функціональні можливості, які раніше були недоступні в традиційних системах безпеки. З іншого боку, ринок активно зростає під впливом чинників, таких як підвищення ризиків терористичних загроз, зростання загального рівня обізнаності споживачів і їхніх вимог до якості та функціональності систем, а також підвищений інтерес до комплексних рішень.

Сьогодні системи контролю доступу (СКД) сприймаються більшістю користувачів як невід’ємна складова загальної безпеки підприємства. Важливу роль у формуванні такого підходу відіграє зростання обізнаності кінцевих користувачів. Із підвищенням рівня усвідомлення потреб у захисті, підприємства почали висувати серйозніші вимоги до СКД. Об’єкти з підвищеними вимогами до безпеки — такі як аеропорти, атомні станції чи великі промислові підприємства — впроваджують біометричні системи ідентифікації. Серед них — розпізнавання за відбитками пальців, формою долоні, райдужною оболонкою ока, рисами обличчя, а також використання багатофакторної автентифікації, яка поєднує біометричні параметри з паролями або картками доступу.

1 Аналітичний огляд

1.1 Дослідження предметної області

Система контролю та управління доступом (СКД) зазвичай включає сервери, які забезпечують її функціонування. Залежно від масштабу мережі та рівня навантаження, сервером СКД може бути як звичайний застарілий ноутбук, так і сучасний високопродуктивний серверний кластер. Основне завдання серверної частини — керування підключеними контролерами доступу. Контролер СКД (іноді його називають панеллю) — це спеціалізований надійний пристрій, який зберігає дані про конфігурацію системи, її режими роботи, перелік користувачів, які мають доступ до об’єкта, а також їхні права доступу. Контролер також керує периферійними пристроями, такими як зчитувачі, турнікети, замки або інші механізми доступу (див. рис. 1.1).

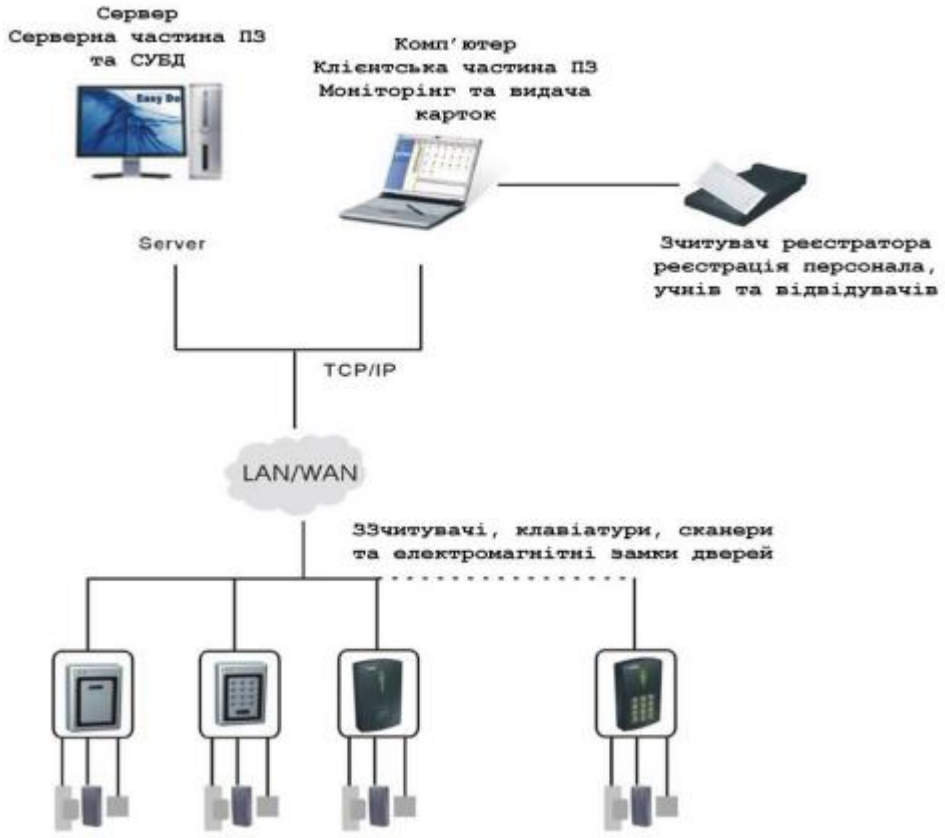

Рис. 1.1. Склад СКД

Блокувальні пристрої (БЛП) — це обладнання, що створює фізичні бар’єри для обмеження доступу, оснащене приводами для керування їхнім станом. До таких пристроїв належать турнікети, шлюзові кабіни, двері та ворота з автоматизованими приводами (САУ). Зчитувач — це пристрій, призначений для зчитування (вводу) ідентифікаційних даних користувача. Отриману інформацію він передає на контролер, який аналізує її і приймає рішення про надання або відмову в доступі. За потреби контролер може бути налаштований на запит підтвердження рішення з центрального комп’ютера. Для підвищення точності ідентифікації до контролера також можна підключити клавіатуру для введення персонального ідентифікаційного коду (PIN).

Ключовим поняттям у СКД є ідентифікатор користувача — унікальна ознака, яка дозволяє ідентифікувати людину або об’єкт, що отримує доступ. Ідентифікатором може виступати код, біометрична характеристика або фізичний носій з зашитою ідентифікаційною інформацією. Фізичні ідентифікатори включають картки, електронні ключі, брелоки тощо, в яких код записується за допомогою спеціальних технологій. Ще один тип обладнання, який може бути підключений до контролера — це охоронна панель. Це спеціалізований контролер, що відстежує стан охоронних датчиків (наприклад, датчики на дверях, вікнах, звукові сенсори тощо). У разі зміни стану будь-якого з датчиків, ця інформація миттєво передається на головний контролер. Приводи — це пристрої, що забезпечують механічне відкриття або закриття блокувальних елементів: електромеханічні та електромагнітні замки, засувки, приводи воріт, турнікетів тощо. Вони можуть містити реле, які дозволяють керувати виконавчими механізмами, такими як замки, турнікети, ліфти, автоматичні ворота та інше обладнання (див. рис. 1.2).

Рис. 1.2. Різноманітність засобів обмеження доступу

Нижче наведено ключові аспекти впровадження системи контролю та управління доступом (СКД) на об’єктах із підвищеними вимогами безпеки. Основним завданням СКД є регулювання доступу до приміщень, що дозволяє чітко розмежовувати права доступу для студентів, працівників і відвідувачів, а також запобігати проникненню сторонніх осіб. Додатково система забезпечує можливість дистанційного керування елементами блокування — такими як замки, турнікети тощо. СКД дозволяє обмежити доступ персоналу у неробочі дні, свята або поза встановленим графіком роботи. Також система здійснює збір і обробку статистичних даних про переміщення осіб через контрольні точки. Для кожного працівника можна зафіксувати точний час входу і виходу, спроби доступу до заборонених приміщень або у невстановлений час. Є можливість простежити маршрут пересування людини по об’єкту із зазначенням конкретних місць і часу проходження.

Завдяки таким функціям СКД забезпечує фіксацію порушень трудової дисципліни, що можуть бути внесені до особової справи працівника. Інформація про такі випадки може бути передана керівництву у вигляді службового повідомлення. Крім того, система дозволяє в режимі реального часу визначити,

де саме перебуває співробітник, на основі даних про останню контрольну точку, через яку він проходив.

Доступ до контрольованих зон здійснюється виключно за допомогою ідентифікатора. Під час проходження через точку доступу за карткою ідентифікації система може виводити на екран фото та дані працівника, що значно знижує ризик використання чужої картки. Крім того, правила реагування СКД дозволяють запобігти передачі картки іншій особі або повторному входу на територію за тим самим ідентифікатором.

Функції обліку робочого часу. СКД також реалізує механізм реєстрації часу прибуття та вибуття співробітників і учнів, що дозволяє обраховувати загальний час їх перебування в установі. Наприклад, о 10:00 система може автоматично формувати звіт про відсутніх працівників, які не пройшли контрольну точку. Це значно полегшує контроль за дисципліною та дозволяє виявляти запізнення чи прогули в автоматичному режимі. Аналогічний звіт можна отримати наприкінці дня — на виході з об’єкта.

Відеоспостереження та надійність живлення. СКД може бути інтегрована з охоронними відеосистемами, забезпечуючи повний візуальний контроль за об'єктом. Для безперебійної роботи у разі зникнення живлення система оснащується джерелами безперебійного живлення (ДБЖ). Також контролери СКД здатні автономно функціонувати у випадку збою центрального комп’ютера.

Захист у реальному часі. СКД дозволяє охороняти приміщення та знімати їх з-під охорони. У разі виникнення надзвичайної ситуації система може автоматично надсилати сповіщення відповідальним особам. Усі події та тривоги фіксуються в базі даних, що дає можливість згодом проаналізувати інциденти. Оператор системи з робочого місця може керувати замками, турнікетами й активувати тривогу за потреби.

Візуалізація об’єкта. У систему можна завантажити поверхові плани з позначеними точками контролю, що дозволяє оперативно орієнтуватися в ситуації та реагувати на події.

Віддалене керування. При підключенні СКД до локальної мережі або Інтернету, адміністрація закладу або служба безпеки отримує змогу керувати системою дистанційно — через комп’ютер або навіть смартфон (у випадку використання GSM-модулів). Це дає змогу контролювати ситуацію в режимі реального часу з будь-якого місця.

Інтеграція з іншими системами безпеки. СКД може бути інтегрована з системами відеоспостереження, охоронною та пожежною сигналізацією. Наприклад:

- при виявленні вторгнення можна автоматично активувати сирену, включити сигнальну лампу або заблокувати всі двері у відповідному секторі;
- у випадку пожежі система автоматично розблокує всі шляхи евакуації — двері, турнікети, проходи — щоб уникнути затримок під час евакуації та підвищити безпеку персоналу.

Завдяки цій гнучкій інтеграції та розширеному функціоналу СКД стає не лише засобом контролю доступу, а й ефективною складовою комплексної системи безпеки об’єкта.

Принцип роботи систем контролю і управління доступом (СКД) полягає в порівнянні ідентифікаційних даних особи або об’єкта з інформацією, збереженою в базі системи. Кожному працівнику видається персоналізований ідентифікатор — у вигляді картки доступу або брелока з унікальним кодом, який активується під час реєстрації на пункті пропуску. Як альтернативу чи доповнення, можуть використовуватись біометричні дані (відбитки пальців, риси обличчя тощо).

Процес проходу через контрольну точку відбувається за допомогою зчитувача, розміщеного біля входу. При зчитуванні даних з ідентифікатора інформація надходить до СКД, де здійснюється її аналіз. Система видає відповідний сигнал, наприклад:

- «Доступ дозволено»;
- «Доступ заборонено»;

- «Повторне використання картки»;
- «Тривога» — у випадку несанкціонованого входу.

Оперативне реагування на інциденти. У разі виникнення надзвичайної ситуації на моніторі охоронця з’являється тривожний сигнал разом з інструкцією щодо подальших дій. У критичних випадках система може автоматично заблокувати двері чи інші елементи проходу для стримування порушника.

Журнал подій дозволяє зберігати, переглядати та друкувати дані про всі дії у системі за обраний період, що сприяє аналізу інцидентів та підвищенню загального рівня безпеки.

Функціональні можливості СКД для підвищення безпеки і контролю:

- Запобігання повторному проходу за однією карткою. Система може блокувати повторний вхід на визначений час або відмовляти у доступі в зони, що не є суміжними.
- Подвійна авторизація. Вхід дозволяється лише у випадку зчитування двох різних карт, що належать двом уповноваженим особам.
- Обмеження кількості осіб у приміщенні. При перевищенні встановленої кількості людей контролер блокує доступ наступним відвідувачам.
- Режим «вхід під примусом». Дає можливість непомітно подати сигнал тривоги, якщо особа змушена проходити примусово.
- Ручне підтвердження охоронцем. Після зчитування картки фотографія її власника з'являється на моніторі охоронця для візуального підтвердження особи.
- Обмеження кількості використань картки. Система може встановлювати ліміт на кількість зчитувань ідентифікатора.
- Прихований контроль доступу. У разі несанкціонованого проникнення сигнал тривоги подається до охорони без виявлення факту порушником.

1.2 Призначення програмного продукту

Система доступу до приміщень — це веб-додаток, розроблений для використання працівниками організацій різного типу. Його головні переваги — висока швидкодія, стабільність у роботі та стійкість до збоїв, що є критично важливим для систем такого призначення.

Цей додаток має на меті ознайомити користувачів із функціональними можливостями платформи, надати корисну інформацію, а також забезпечити зручну й інтуїтивно зрозумілу взаємодію співробітників із системою з метою ефективного управління доступом до приміщень.

Мета створення системи: Автоматизувати процеси контролю доступу та зменшити час, необхідний для налаштування системи та підключення нових контрольних точок.

Завдання веб-додатку:

- Формувати у користувачів довіру до компанії;
- Забезпечити автоматизовану реєстрацію часу прибуття працівників;
- Надати структуровану та доступну інформацію про приміщення;
- Мінімізувати необхідність прямої взаємодії користувачів із персоналом.

1.3 Огляд існуючих засобів

Серед провідних систем автоматизації контролю доступу варто виділити такі рішення, як ZkTeco, SIGUR, ControlGate та HID Global. Кожна з них вирізняється високим рівнем функціональності та технічної складності. Однак, незважаючи на спільну мету, ці системи мають свої особливості в налаштуванні та використанні. У цьому розділі буде докладно розглянуто можливості кожної

з них, а також на основі порівняння буде обрано одну систему для подальшого використання як приклад.

Компанія ZkTeco Co., Ltd, заснована в Китаї, добре відома завдяки акценту на біометричні методи ідентифікації. Вона орієнтована на потреби середнього та великого бізнесу і має понад 90 представництв у різних країнах світу, включаючи Росію та держави СНД.

Її флагманське програмне забезпечення працює на основі хмарної платформи ZKTeco + Smart Office. Система підтримує біометричну автентифікацію, інтеграцію з відеоспостереженням та інтелектуальне управління доступом до людей, транспорту та об'єктів. Програма ZKTime.Net V3.0, розроблена для обліку робочого часу, підтримує розпізнавання облич, відбитків пальців і комбіновані методи ідентифікації. Вона також здатна формувати аналітичні звіти та розрахункові відомості. З урахуванням новітніх викликів, компанія впроваджує безконтактні методи верифікації, адаптовані під вимоги пандемії COVID-19, які інтегруються через спеціалізовані модулі.

Система SIGUR пропонує спеціалізовані рішення для офісних будівель, адміністративних центрів і ділових комплексів, забезпечуючи ефективний контроль доступу як до внутрішніх приміщень, так і до прилеглих територій, наприклад, паркінгів.

Серед її ключових можливостей — підтримка необмеженої кількості робочих місць для орендарів, готові автоматизовані робочі місця (АРМ) для охорони та бюро перепусток. Система також дозволяє організовувати автоматичну видачу та збір перепусток для відвідувачів і має високу масштабованість, що робить її гнучким рішенням для бізнесу.

Компанія HID Global є одним із провідних світових виробників систем контролю та управління доступом. Її головний офіс знаходиться в Остіні, штат Техас (США). Програмне забезпечення HID вирізняється широким спектром застосування. Воно активно використовується не лише в СКД для організації фізичного та логічного доступу співробітників, а й у фінансовій сфері — для створення та обслуговування платіжних та ідентифікаційних карток. Технології

компанії охоплюють також біометричну ідентифікацію, автентифікацію користувачів, а також інтегруються в операційні системи для мікрочипів і рішення для Інтернету речей (IoT).

Попри всі переваги та багатофункціональність продуктів HID Global, основним недоліком є висока вартість програмного забезпечення. Проте така ціна є типовою для більшості імпортних рішень подібного класу.

1.4 Функції програмного продукту

Система повинна функціонувати на основі гібридної моделі, яка поєднує пристрій для зчитування карток та веб-додаток. Користувач підходить до приміщення і прикладає картку до спеціального зчитувача. Після введення персональних даних він отримує дозвіл на доступ до приміщення. Кожне приміщення має бути обладнане відповідним пристроєм для зчитування карток.

1.5 Вибір мови програмування

Перед тим, як почати вивчати мови програмування і писати код, важливо зрозуміти значення двох ключових понять. Фронтенд-розробники відповідають за клієнтську частину — те, що бачить користувач на екрані. Бекенд — це серверна частина системи, що забезпечує роботу сервісу на рівні програмного забезпечення і апаратури. Залежно від профілю, програмісти використовують різні технології для створення сайту. Фронтенд зазвичай включає HTML, CSS і JavaScript, тоді як бекенд-розробникам необхідні такі мови, як PHP, Python чи Ruby.

JavaScript - це одна з найпопулярніших мов програмування. Часто початківці плутають JavaScript з Java, але це абсолютно різні мови. JavaScript має широке застосування — ним пишуть серверні, мобільні та десктопні додатки. Будь-який браузер і операційна система підтримують цю мову. Скрипти

виконуються прямо в браузері, і користувачу не потрібно додатково нічого запускати. Найчастіше JavaScript використовується для створення анімацій, інтерактивних скриптів і елементів інтерфейсу.

Головна перевага PHP полягає в тому, що його код легко поєднується з HTML-розміткою, що дозволяє одночасно керувати зовнішнім виглядом сторінки та її функціональністю. Мова досить проста для вивчення і має хорошу підтримку роботи з даними, сумісна з різними платформами та операційними системами. PHP спеціально розроблений для серверної частини, а його бібліотеки ідеально підходять для повторюваних завдань при розробці сайтів.

Багато фахівців вважають Python оптимальним для Data Science — області аналізу даних з використанням машинного навчання і штучного інтелекту. Однією з ключових переваг цієї мови є її простота. Навіть новачки можуть досить швидко освоїти основи Python. Крім того, Python допомагає заощадити час розробника завдяки великій кількості спеціалізованих бібліотек з готовими рішеннями.

Ruby переважно використовується для розробки веб-сайтів та мобільних додатків. Раніше його часто вважали повільною мовою, яка не підходить для масштабування великих проектів. Спочатку Ruby поступався за продуктивністю PHP та Python, але з часом численні оновлення значно покращили його швидкодію. Майбутні оновлення планують додати підтримку паралельних потоків. Продуктивність сучасних застосунків на Ruby багато в чому залежить від навичок програміста та правильності побудови архітектури. До плюсів мови відносять простоту вивчення, що робить її популярною серед початківців, а також зручність у записі коду.

Мова програмування C# запозичила багато концепцій і синтаксису у Java і C++. Більша частина його синтаксису схожа на Java. Спершу C# використовувався для створення веб-сайтів, але з часом активно розвивається — з’явилися асинхронні методи, динамічне зв’язування та інші сучасні функції. Порівняно з іншими популярними мовами, C# є відносно молодою — його перша версія вийшла у 2002 році.

На початку свого існування Perl створювався, щоб звести до мінімуму необхідність писати програми і скрипти на різних мовах, об’єднуючи в собі можливості системного адміністрування та обробки документів. Сьогодні Perl широко застосовується для створення інтерактивних додатків, адміністрування серверів і підтримує всі популярні платформи, такі як Windows, Mac та інші.

Java — одна з найпопулярніших мов для розробки мобільних додатків і мережевих програм, особливо в екосистемі Android. Вона постійно оновлюється та залишається актуальною. Мова підтримує об’єктно-орієнтоване програмування (ООП), що дозволяє розбивати складний код на незалежні об’єкти з власними функціями і даними, які активуються тільки при зверненні, що запобігає конфліктам, характерним для процедурного програмування. Серед переваг Java — безпека, надійність і зрозумілий синтаксис.

2 Вимоги до системи

2.1 Бізнес-вимоги до системи

Оскільки головною метою є розробка системи контролю доступу до приміщень, слід визначити ключові бізнес-вимоги:

а) Система має забезпечувати безпеку перебування на території підприємства та захищати її від несанкціонованого проникнення.

б) Впровадження системи сприятиме цифровізації процесів на підприємствах.

в) Система спростить контроль і облік перебування співробітників у різних зонах підприємства.

г) Вона повинна підтримувати прийняття рішень для запобігання несанкціонованому доступу на приватну територію.

д) Створити єдину цифрову платформу для управління, редагування прав доступу та моніторингу інформації про їх використання.

2.2 Функціональні вимоги до системи

Система буде побудована за модульним принципом і включатиме наступні компоненти:

– список пристроїв;

– список користувачів;

– журнал доступу;

– особистий кабінет;

– звіти.

Основні вимоги:

1. Автоматизована система має забезпечувати авторизацію за логіном і паролем.

2. Передбачено нагадування паролю користувачу через номер мобільного телефону.

3. Система повинна надсилати повідомлення співробітникам.

Модуль «Список пристроїв»:

1. Додавання нових пристроїв до системи.

2. Видалення пристроїв зі списку.

3. Встановлення індивідуального пароля для кожного пристрою.

4. Налаштування прав доступу працівників до окремих пристроїв.

Модуль «Список користувачів»:

1. Реєстрація нових користувачів у системі.

2. Видалення користувачів.

3. Редагування персональної інформації користувачів.

4. Перегляд прав доступу кожного користувача до пристроїв.

Модуль «Журнал доступу»:

1. Відображення інформації про всі сесії перебування працівників на контрольованій території.

2. Додавання та видалення записів про отримання доступу.

Модуль «Особистий кабінет»:

1. Заповнення та редагування особистих даних користувача.

2. Додавання фотографії профілю.

3. Перегляд журналу доступу конкретного користувача до пристроїв.

Модуль «Звіти»:

1. Формування звітів про доступ до конкретних пристроїв.

2. Формування звітів про доступ певних користувачів.

2.3 Нефункціональні вимоги до системи

Можливість повторного використання — це вимога до здатності використовувати реалізації або компоненти програми чи системи кілька разів (Reusability), що є одним із важливих аспектів на етапі проектування (design time).

Зазвичай ця вимога виникає, коли загальні компоненти застосовуються у кількох модулях розроблюваної системи.

Повторне використання може проявлятися в архітектурі, функціях, модулях, класах, дизайні, базах даних (наприклад, коди доступу або користувачі), сценаріях тощо.

Компоненти повинні бути універсальними, абстрактними і не надто «інтелектуальними», адже компоненти, які мають надмірні знання про загальну систему, складно замінити.

Вважається, що компоненти, які залежать від багатьох інших, є обтяженими. Такі обтяжені компоненти важко використовувати повторно, бо при цьому необхідно також переносити всі залежні від них компоненти, що ускладнює інтеграцію в нову систему, як показано на рис. 2.1.

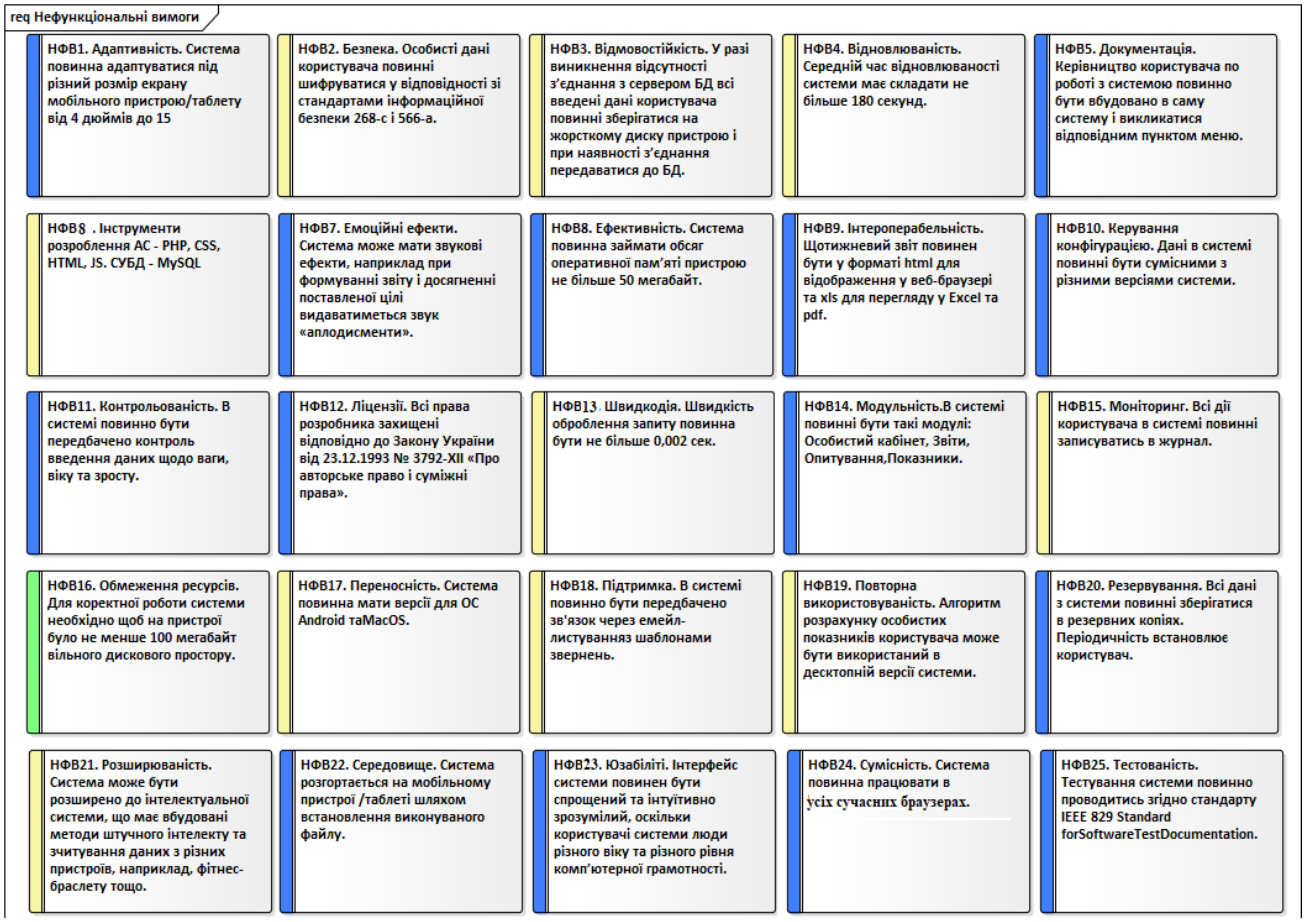

Рис. 2.1 – Діаграма нефункціональних вимог до автоматизованої системи контролю відвідування

До нефункціональних вимог системи належать:

– Система має адаптуватися до різних розмірів екранів мобільних пристроїв і планшетів від 4 до 15 дюймів.

– Для розробки слід використовувати PHP, CSS, HTML, JavaScript, а в якості СУБД – MySQL.

– У системі повинен бути передбачений контроль правильності введення даних щодо віку, факультету та групи.

– Особисті дані користувачів мають шифруватися відповідно до стандартів інформаційної безпеки 268-с та 566-а.

– Система може містити звукові ефекти, наприклад, звук «аплодисменти» при формуванні звіту або досягненні певного результату.

– Права розробника захищаються згідно з Законом України від 23.12.1993 № 3792-XII «Про авторське право і суміжні права».

– У разі втрати з’єднання з сервером бази даних, всі введені користувачем дані повинні зберігатися локально на пристрої та автоматично передаватися в базу даних при відновленні зв’язку.

– Обсяг оперативної пам’яті, яку використовує система, не має перевищувати 300 МБ.

– Швидкість обробки запиту повинна становити не більше 0,002 секунди.

– Середній час відновлення роботи системи не повинен перевищувати 180 секунд.

– Щоденні або щотижневі звіти мають бути доступні у форматах HTML для перегляду у веб-браузері, а також у форматах XLS і PDF для роботи у Excel.

– Система повинна містити такі модулі: особистий кабінет, звіти, опитування, показники.

– Керівництво користувача має бути інтегроване у систему і доступне через відповідний пункт меню.

– Дані системи мають бути сумісні з різними версіями програмного забезпечення.

– Всі дії користувачів повинні записуватися в журнал подій.

– Для коректної роботи системи на пристрої має бути не менше 100 МБ вільного дискового простору.

– Система може бути розширена до інтелектуальної платформи зі вбудованими методами штучного інтелекту та підтримкою зчитування даних з різних пристроїв, зокрема смартфонів.

– Система повинна підтримувати роботу на різних операційних системах: Windows, Linux, Arduino та iOS.

– Розгортання системи на мобільних пристроях чи планшетах має відбуватися без необхідності встановлення виконуваних файлів.

– У системі має бути реалізовано механізм комунікації через електронну пошту з використанням шаблонів листів.

– Інтерфейс системи повинен бути простим та інтуїтивно зрозумілим, щоб відповідати потребам користувачів різного віку та рівня комп’ютерної грамотності.

– Алгоритми розрахунку персональних показників користувача можуть бути використані також у десктопній версії системи.

– Система повинна коректно працювати у всіх сучасних веб-браузерах.

– Всі дані мають регулярно зберігатися у резервних копіях, а періодичність їх створення визначається адміністратором системи.

– Тестування системи повинно виконуватися згідно зі стандартом IEEE 829 «Standard for Software Test Documentation».

Основні принципи проектування з урахуванням повторного використання компонентів, представлені на рис. 2.2, включають:

– Принцип абстракції (Abstraction Principle);

– Принцип модульності (Modularity Principle);

– Принцип відкритості-закритості (Open-Closed Principle).

Програмні компоненти часто класифікують за рівнями повторного використання, що також показано на рис. 2.2:

1. Базові компоненти — це такі класи, як Гроші, Дата, Список, Календар, Облікові дані (логін, пароль) та Число. Вони мають універсальне застосування в різних додатках і характеризуються низьким навантаженням.

2. Компоненти, специфічні для архітектури, включають механізми обробки подій, елементи користувацького інтерфейсу та системи обміну повідомленнями.

3. Доменні компоненти — це класи, пов’язані з предметною областю, наприклад Customer (Клієнт), Account (Рахунок) і Transaction (Транзакція).

4. Компоненти, специфічні для конкретного додатку, включають обробники повідомлень, винятків та представлень.

5. Панелі інструментів, що використовуються в додатках.

6. Фреймворки, які забезпечують загальний користувацький інтерфейс, механізми обробки помилок, збереження даних та архітектурні рішення.

7. Використання бібліотек класів для спрощення розробки.

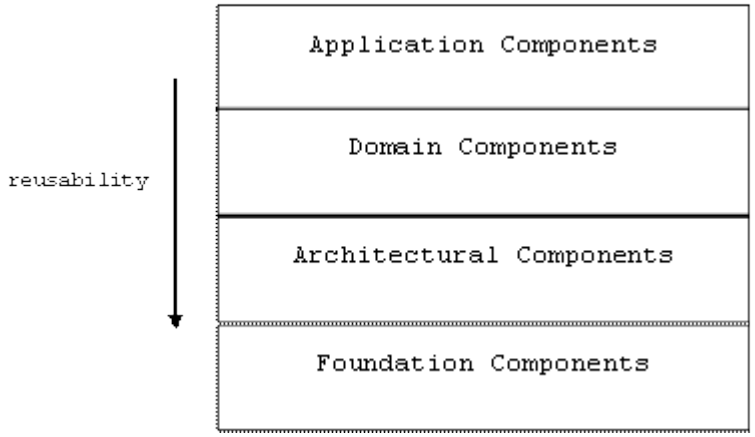

Рис. 2.2. Рівні повторного використання

Документація, пошук і простота використання є одними з важливих функцій для успішної стратегії повторного використання.

3 Проектування та реалізація компонентів системи

3.1 Моделювання поведінки системи

Розглянемо поведінку автоматизованої системи контролю доступу як набір функцій та акторів, що з ними взаємодіють. Модель поведінки системи для контролю доступу подано в нотації UML, діаграмою прецедентів, що описує доступні дії та випадки (рис. 3.1).

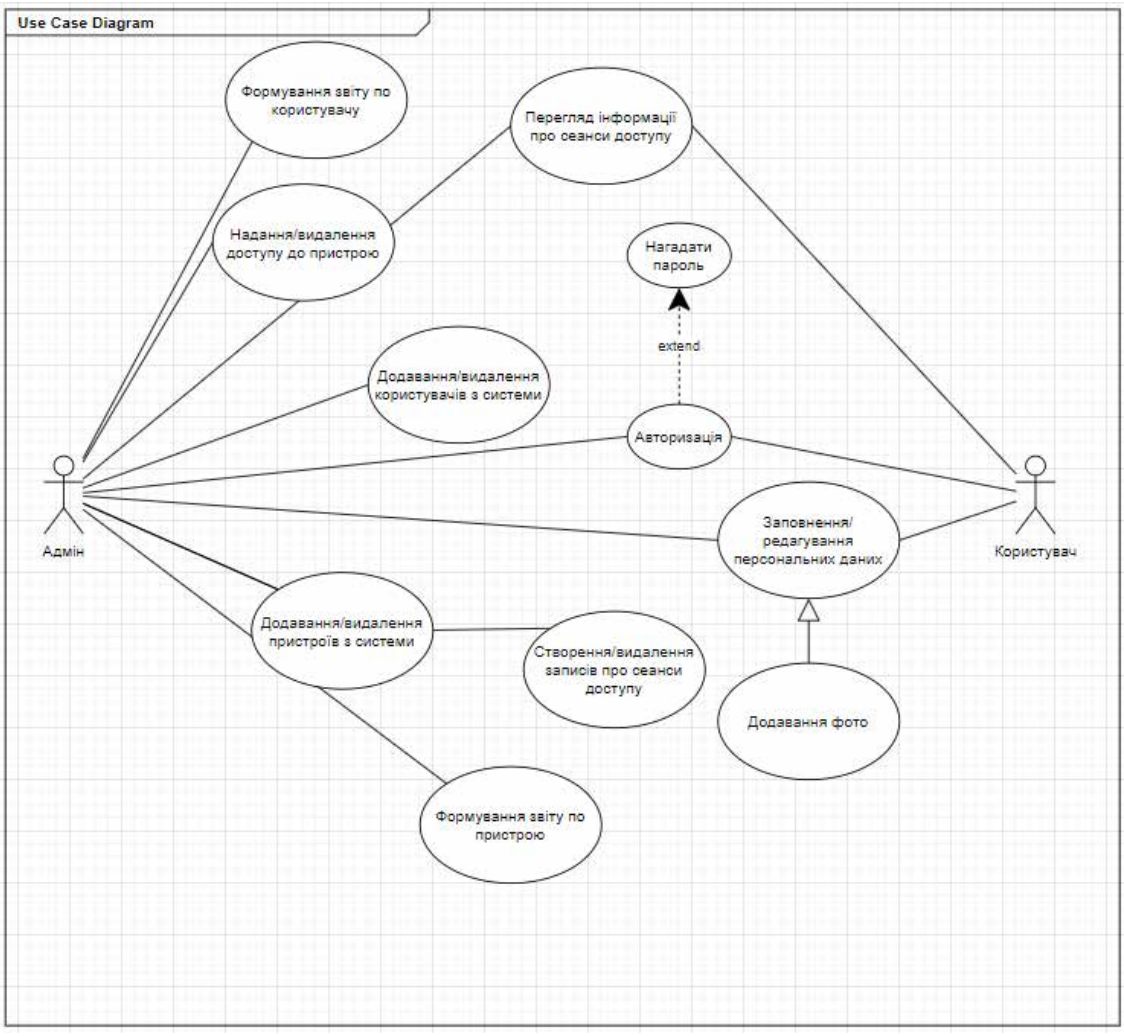

Рис. 3.1. Діаграма прецедентів автоматизованої системи контролю відвідувань

Для предметної області виділено наступних акторів, що зображено в табл. 3.1:

Таблиця 3.1

Визначені актори

Актор	Короткий опис
Адмін	Співробітник, який має можливість керувати усіма основними процесами системи, переглядати усі звіти
Користувач	Співробітник, який має можливість редагувати свої дані та переглядати інформацію про свої сеанси

Розглянемо, які функції має виконувати наша система:

– користувач з роллю Адміністратора має можливість редагувати дані користувачів, створювати та видаляти користувачів і пристрої, а також переглядати звіти щодо використання пристроїв;

– користувач із роллю Користувача може змінювати свої персональні дані та переглядати звіти про використання пристроїв за своєю картою.

Виходячи з цього, можна виділити основні факти, наведені у таблиці 3.2:

Таблиця 3.2

Опис прецедентів

Прецедент	Короткий опис
Формування звіту по користувачу	Запускається Адміном. Дозволяє отримати звіт про використання своєї картки певним користувачем.
Надання/видалення доступу до пристрою	Запускається Адміном. Дозволяє надати або видалити певному користувачу доступ до пристрою.
Перегляд інформації про сеанси доступу	Запускається Адміном та Користувачем. Дозволяю переглянути інформацію про отримання доступу користувачем.
Додавання/видалення користувачів з системи	Запускається Адміном. Дозволяю додати або видалити користувача з системи.
Авторизація	Запускається Адміном та Користувачем. Дозволяє провести авторизацію в системі

Продовження таблиці 3.2

Прецедент	Короткий опис
Нагадати пароль	Підсистема «Авторизація», яка запускається в разі необхідності відновити пароль користувача
Заповнення/Редагування персональних даних	Запускається Адміном, Користувачем. Дозволяє користувачам персоналізувати особистий кабінет.
Додавання/видалення пристроїв з системи	Запускається Адміном. Дозволяє додавати та видаляти пристрої з системи.
Створення/видалення записів про сеанси доступу	Запускається Адміном. Дозволяє створювати та видаляти записи про отримання користувачами доступу до певних пристроїв.
Додавання фото	Підсистема «Заповнення/Редагування персональних даних», яка запускається в разі необхідності додати фото до особистого кабінету.
Формування звіту по пристрою	Запускається Адміном. Дозволяє отримати звіт про використання користувачами певного пристрою.

На діаграмах послідовності відображена модель системних прецедентів взаємодії акторів із системою контролю відвідувань студентів у вищому навчальному закладі. Діаграми послідовності є видом UML-діаграм взаємодії, які демонструють відносини між об’єктами в різних умовах. Ці умови задаються сценаріями, що формуються на етапі розробки діаграм варіантів використання [4]. Існують різні підходи до застосування таких діаграм:

а) Фаулер рекомендує використовувати діаграми послідовності для візуалізації найбільш складних взаємозв’язків у діаграмах класів [5];

б) Буч розглядає їх як альтернативу діаграмам об’єктів і застосовує для аналізу семантики сценаріїв на початкових етапах проектування, ще до створення протоколів окремих класів [6];

в) Розенберг включає побудову діаграм послідовності до процесу ICONIX, створюючи їх для кожного прецеденту, а не лише для найважливіших зв’язків, як у Фаулера. Перед цим у процесі ICONIX будуються діаграми робастності, де визначаються об’єкти, що беруть участь у прецеденті [7].

На рисунку 3.2 представлена діаграма послідовності, що ілюструє процес авторизації користувача в системі.

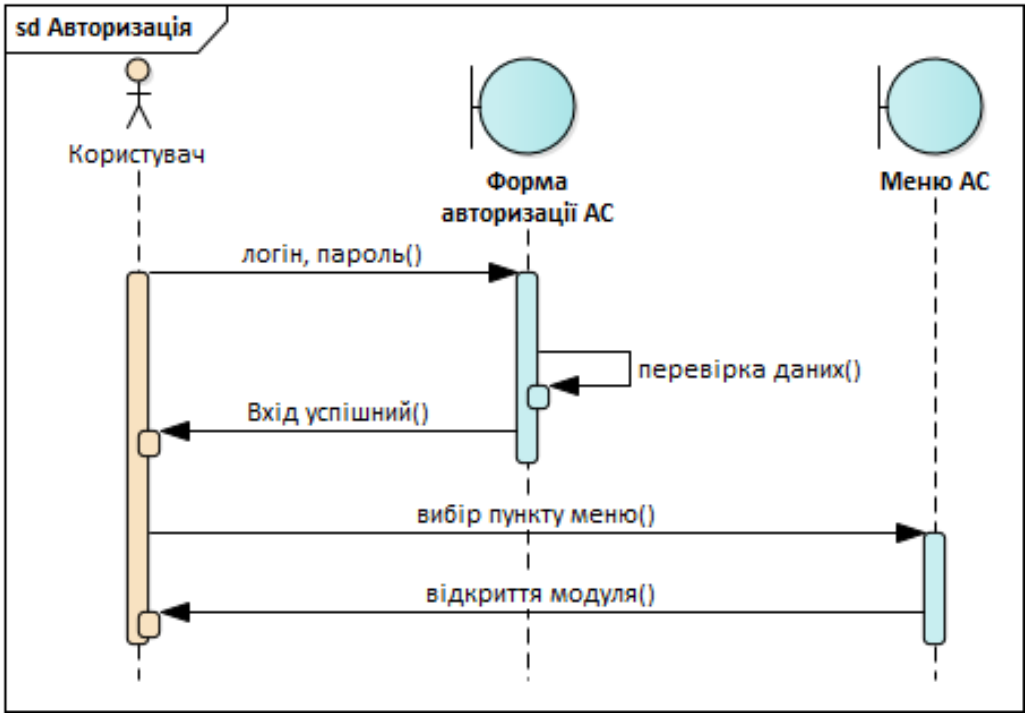

Рис. 3.2. Діаграма послідовності процесу «Авторизація»

На рис. 3.3 зображено діаграму послідовності, яка описує процес формування звітності по певним дисциплінам.

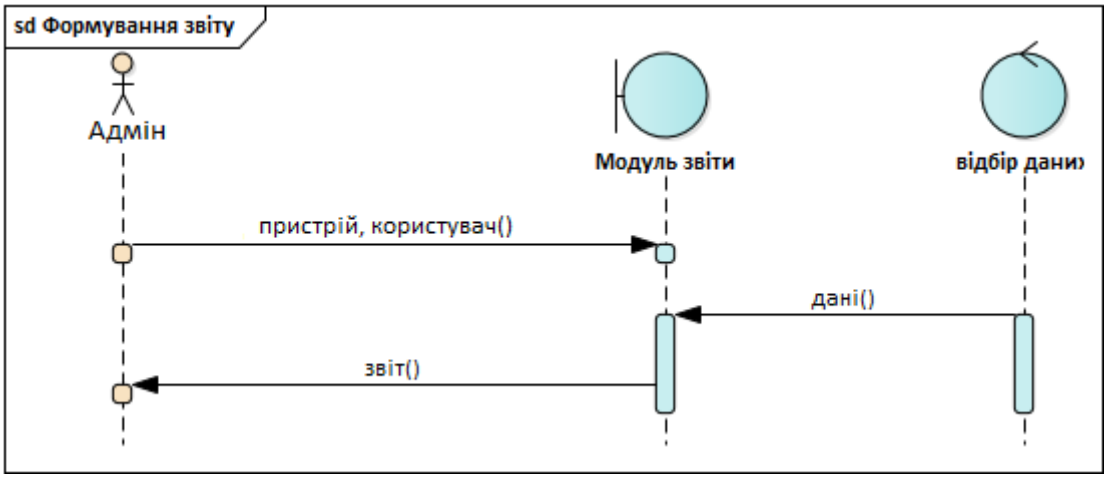

Рис. 3.3. Діаграма послідовності процесу «Формування звіту»

На рис. 3.4 зображено діаграму послідовності, яка описує процес заповнення особистого кабінету.

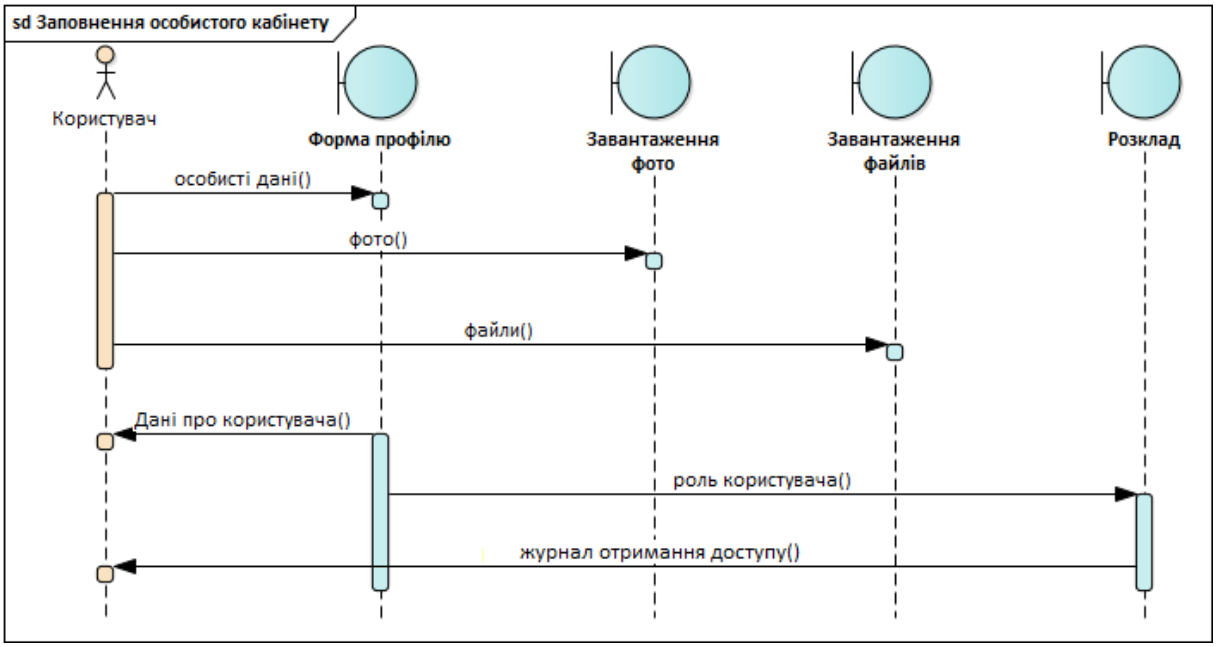

Рис. 3.4. Діаграма послідовності процесу «Заповнення особистого кабінету»

3.2 Моделювання структури системи

SysML — це мова моделювання архітектури загального призначення, призначена для застосувань у системній інженерії [8]. Вона підтримує специфікацію, аналіз, проєктування, верифікацію та валідацію різноманітних систем і системних комплексів, які можуть включати апаратні компоненти, програмне забезпечення, інформацію, процеси, персонал та засоби.

По суті, SysML є діалектом UML 2 і визначається як його профіль. (Профіль UML — це варіант UML, який налаштовує мову за допомогою трьох механізмів: стереотипів, позначених значень і обмежень.)

SysML є зручною технологією для моделювання в системній інженерії, відомою як MBSE (Model-Based Systems Engineering).

За допомогою діаграми внутрішніх блоків у нотації SysML на рисунку 3.5 зображені компоненти автоматизованої системи контролю відвідувань студентів та зв’язки між ними.

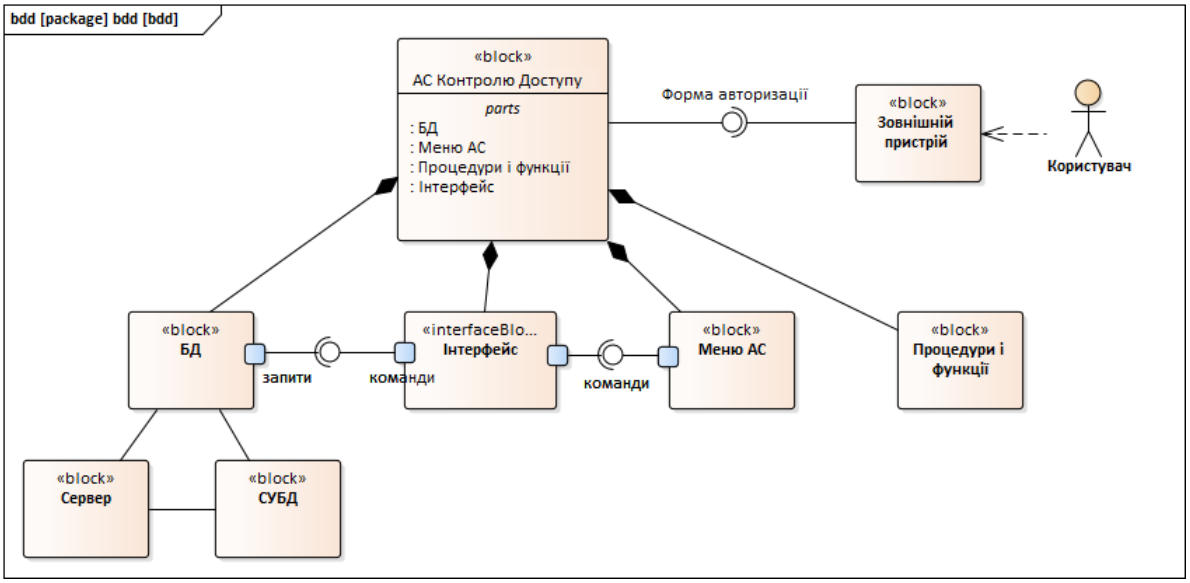

Рис. 3.5. Діаграма внутрішніх блоків системи

На діаграмі класів-сутностей (рис. 3.6) представлені ключові класи, які відображають структуру системи контролю відвідувань і застосовуються для моделювання даних та поведінки системи.

Схема даних системи контролю відвідувань включає три основні класи сутностей:

Користувачі:

- ідентифікатор користувача (тип даних – integer);
- ім’я користувача (тип даних – character);
- номер карти користувача (тип даних – character);
- пароль користувача (тип даних – character);
- дата створення (тип даних – timestamp);
- дата останнього оновлення (тип даних – timestamp);
- метод створення користувача, який повертає об’єкт користувача;
- метод редагування користувача, який не повертає значення;
- метод видалення користувача, який не повертає значення.

Пристрої:

- ідентифікатор пристрою (тип даних – integer);
- назва пристрою (тип даних – character);
- пароль пристрою (тип даних – character);

– ключ доступу до пристрою (тип даних – character);
– дата створення (тип даних – timestamp);
– дата останнього оновлення (тип даних – timestamp);
– метод створення пристрою, що повертає об’єкт пристрою;
– метод видалення пристрою, який не повертає значення.

Логи отримання доступу:

– ідентифікатор користувача (тип даних – integer);
– ідентифікатор пристрою (тип даних – integer);
– дата створення запису (тип даних – timestamp);
– дата останнього оновлення запису (тип даних – timestamp);
– метод створення запису, що повертає об’єкт запису;
– метод видалення запису, який не повертає значення.

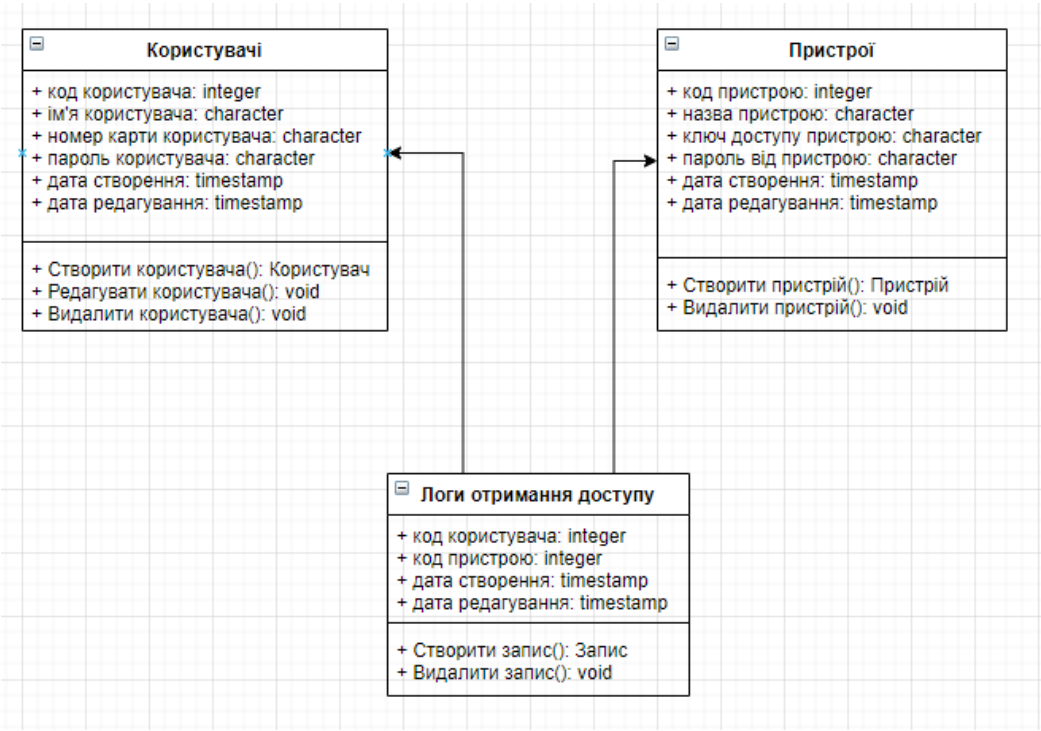

Рис. 3.6. Діаграма класів-сутностей системи контролю доступу до приміщень

3.3 Розробка структури програмних модулів

Для розробки програмного додатку будуть використані такі технології:

– середовище PHP;
– Docker;
– база даних MySQL;
– PHPUnit;
– NodeJS.

Під час створення нового проекту на Laravel автоматично генерується файл .env, який копіюється з шаблону .env.example та призначений для налаштування конфігурації і облікових даних. Після створення проекту необхідно налаштувати такі змінні:

Лістинг 3.1 – Налаштування середовища

```
DB_CONNECTION=mysql
2DB_HOST=127.0.0.1
3DB_PORT=3306
4DB_DATABASE=laravel
5DB_USERNAME=root
6DB_PASSWORD=
```

Щоб створити уявлення, що відображає список посилань, нам необхідно оновити основний маршрут проекту, а також визначити новий маршрут, на якому буде відображатися наша форма відправки. Ми можемо додати нові маршрути в наш додаток в web.php файл.

Лістинг 3.2 – Налаштування посилань

```
Route::get('/', function () {
    return view('welcome');
});
```

Для створення нового маршруту можна застосовувати як замикання (closure), так і виділений клас контролера. У нашому випадку для маршрутів, що

відповідають за надсилання та відображення даних, ми будемо використовувати замикання.

Контролер виконує роль посередника між моделями та уявленнями. Коли користувач надсилає форму для створення нового запису в блозі, дані надходять до контролера, де відбувається їх обробка, після чого вони передаються в модель для збереження у базі даних. Потім контролер надсилає відповідь у вигляді подання, повідомляючи, що запис було успішно створено.

Контролери будуть створені за допомогою спеціальної команди і відповідатимуть угодам про найменування: ім’я класу має закінчуватися на «Controller». Зокрема, контролер для формування звітів ми назвемо ReportController.php.

Лістинг 3.3 – Команда створення контролеру

```
php artisan make:controller ReportController
```

Контролер має 7 методів, які дозволяють виконувати операції crud:

– index() – щоб отримати всі ресурси, наприклад, всі доступні записи.

– show() – щоб отримати один ресурс, наприклад, один звіт.

– create() – показує форму, використовувану для створення ресурсу (недоступна для контролерів API).

– store() – щоб зафіксувати ресурс у базі даних.

– update() – щоб зафіксувати відредагований ресурс у базі даних.

– destroy() – щоб видалити ресурс з бази даних.

Для роботи з базою даних потрібно створити таблиці. Таблиці створюються за допомогою файлів міграції. Для запуску файлу міграції та створення таблиці потрібно виконати спеціальну команду:

Лістинг 3.4 – Запуск міграції

```
php artisan migrate
```

Нова міграція буде автоматично збережена в директорії database/migrations. Назва кожного файлу міграції містить унікальну позначку часу, яка дозволяє фреймворку Laravel визначити правильну послідовність їх виконання.

Клас міграції включає два основні методи: up і down. Метод up використовується для створення нових таблиць, додавання стовпців або індексів у базу даних. Натомість метод down виконує зворотні дії — скасовує зміни, внесені методом up.

Обидва методи дозволяють використовувати вбудований конструктор схем Laravel, що забезпечує зручне й наочне створення або зміну таблиць. Повний перелік доступних методів можна знайти в офіційній документації до Schema. Наприклад, нижче наведена міграція створює таблицю users:

Лістинг 3.5 – Створення таблиці Users

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('card_number')->nullable()->
>default(null);
            $table->string('password');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
```

```
         * @return void
         */
        public function down()
        {
            Schema::dropIfExists('users');
        }
    }
```

Після того, як ми налаштували контролери та базу даних, потрібно створити веб сторінки, через які буде відбуватися взаємодія користувача з нашими методами. Всередині папки ресурсів створімо файл, який буде відповідати за нашу веб сторінку.

Нижче наведено остаточний код того, як виглядатиме файл:

Лістинг 3.6 – Веб-сторінка

```
@extends('index')

@section('content')
<div class="container">
<div class="row">
<div class="offset-col-3 col-6">
<form action="/users/store" method="POST" class="form">
@csrf

<div class="form-group">
<label for="name">User name</label>
<input type="text" id="name" name="name">
</div>

<div class="form-group">
<label for="card_number">Card number</label>
<input type="text" id="card_number" name="card_number">
</div>

<div class="form-group">
<label for="password">Password</label>
<input type="text" id="password" maxlength="4" minlength="4"
name="password">
</div>

<div class="form-group">
<label for="device">Device</label>
<select id="device_id" name="device_id">
@foreach($devices as $device)
<option value="{{$device->id}}">{{$device->name}}</option>
@endforeach
</select>
```

```
</div>

<button type="submit">Create</button>
</form>
</div>
</div>
</div>
@endsection
```

3.4 Розгортання та налаштування системи

Для роботи з платформою Laravel необхідно дотримуватися певних системних вимог. Переконайтесь, що ваш веб-сервер відповідає мінімальним вимогам до версії PHP та має встановлені такі розширення:

- PHP версії не нижче 7.3;
- Розширення BCMath;
- Розширення Ctype;
- Розширення Fileinfo;
- Розширення JSON;
- Розширення Mbstring;
- Розширення OpenSSL;
- Розширення PDO;
- Розширення Tokenizer;
- Розширення XML.

У моєму випадку додаток розгортається на сервері з веб-сервером Nginx. Для налаштування використовуємо відповідний конфігураційний файл як шаблон. Якщо ви потребуєте допомоги з адмініструванням чи розгортанням сервера, рекомендую звернутися до спеціалізованих рішень, таких як Laravel Forge — офіційний сервіс для управління інфраструктурою Laravel [9].

Для правильної роботи необхідно налаштувати файл конфігурації веб-сервера.

Лістинг 3.7 – Файл налаштувань серверу

```
server {
    listen 80;
    server_name example.com;
    root /srv/example.com/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location = /favicon.ico { access_log off; log_not_found
off; }
    location = /robots.txt  { access_log off; log_not_found
off; }

    error_page 404 /index.php;

    location ~ \.php$ {
        fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        fastcgi_param                          SCRIPT_FILENAME
$realpath_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.(?!well-known).* {
        deny all;
    }
}
```

Існує кілька популярних постачальників VPS, зокрема DigitalOcean, Vultr, Linode та Hetzner. Хоча принцип роботи з некерованими серверами у всіх подібний, перелік додаткових сервісів у кожного провайдера може відрізнятися. Наприклад, DigitalOcean пропонує керовані бази даних, чого не мають Linode і Vultr.

Перш ніж налаштовувати новий сервер, необхідно створити SSH-ключі для безпечного підключення.

Після виконання базової конфігурації можна перейти до розгортання коду. Деякі розробники просто клонують репозиторій безпосередньо на сервері та

щоразу вручну виконують git pull при зміні коду. Але є ефективніший підхід — використовувати сервер як Git-репозиторій та надсилати код прямо з локального середовища. Крім того, варто автоматизувати післярозгортальні задачі, такі як встановлення залежностей, запуск міграцій тощо. Але перед цим необхідно встановити PHP і Composer на сервері [10].

Після цього додаток майже готовий до обробки запитів користувачів з усього світу. Наступний важливий крок — кешування конфігурацій, шаблонів і маршрутів у Laravel для покращення продуктивності.

Валідація у Laravel забезпечує перевірку правильності введених користувачем даних (наприклад, персонального ключа) за допомогою визначених правил у відповідних функціях.

Laravel Sail — це інструмент командного рядка, що спрощує взаємодію з середовищем Docker для розробки. Він ідеально підходить для створення Laravel-додатків з використанням PHP, MySQL та Redis. Для його використання не потрібні глибокі знання Docker. По суті, Sail — це docker-compose.yml файл у корені проєкту та набір команд для управління контейнерами. Sail працює в macOS, Linux і Windows (через WSL2) і встановлюється разом з новими проєктами Laravel.

Файл docker-compose.yml визначає набір контейнерів Docker, які працюють разом для розробки Laravel-додатка. Основний контейнер — laravel.test, який відповідає за обслуговування програми.

Перед запуском Sail переконайтеся, що на вашій машині не працюють інші локальні веб-сервери або СУБД. Щоб активувати всі контейнери, зазначені у docker-compose.yml, виконайте команду:

Лістинг 3.11 – Команда для запуску контейнеру з додатком

```
sail up
```

Щоб зупинити всі контейнери, ви можете просто натиснути Control + C, щоб зупинити виконання контейнера. Якщо контейнери працюють у фоновому режимі, ви можете використовувати команду stop:

Лістинг 3.12 – Команда для зупинки контейнеру з додатком

```
sail stop
```

3.5 Тестування працездатності системи

Laravel із коробки підтримує три основні типи тестів:

- Browser (браузерні тести),
- Feature (функціональні тести),
- Unit (модульні тести).

У більшості випадків для кожного окремого сценарію, що перевіряється через форму, потрібно створювати окремий модульний тест. Це часто призводить до дублювання коду — наприклад, у тестах типу test_request_without_title чи test_request_without_content, які виконуються однаково, змінюючи лише передані дані.

Для уникнення надмірної кількості однотипного коду доцільно використовувати параметризовані тести або рефакторинг.

Тестування системи має включати перевірку:

- повноти,
- точності,
- логічної узгодженості технічної документації.

Об'єктами експертного аналізу документації виступають:

- технічне завдання,
- документація технічного проєкту,

– експлуатаційні матеріали (інструкція користувача, адміністративне керівництво тощо).

Етапи проведення експертної перевірки документації:

– Отримання повного пакету технічної документації та додаткових матеріалів (вимоги користувачів, документація подібних рішень тощо).
– Організація процесу експертного аналізу, який включає:
  - перевірку відповідності назв елементів інтерфейсу (вікон, меню, кнопок) фактичному застосунку;
  - аналіз описаних у документації дій — чи дійсно вони ведуть до очікуваного результату;
  - перевірку відсутності суперечностей між вимогами та іншими супровідними документами;
  - оцінку повноти опису та глибини деталізації;
– Підготовка експертного висновку за результатами перевірки;
– Форма висновку створюється виконавцем і погоджується із замовником до передачі першого примірника звіту.

Laravel спроектований з урахуванням потреб тестування. Фреймворк вже містить інтеграцію з PHPUnit — файл phpunit.xml попередньо налаштовано для вашого проєкту. Крім того, Laravel надає зручні інструменти для організації повноцінного тестування.

У каталозі tests ви знайдете файл ExampleTest.php, який демонструє базовий приклад тесту. Щоб запустити тести після встановлення Laravel, достатньо виконати команду:

Лістинг 3.13 – Команда створення тесту

```
php artisan make:test DeviceTest
```

Згенерований клас запиту буде розташовано в директорії App/Http/Requests. Після його створення необхідно визначити набір правил валідації, які застосовуватимуться до цієї форми запиту.

Функціональне тестування має охоплювати такі етапи:

- створення методики проведення тестування;
- побудова інфраструктури для тестування;
- підготовка тестових сценаріїв;
- виконання тестування відповідно до затвердженої методики;
- аналіз результатів тестів і підготовка підсумкового звіту;
- презентація підсумків реалізації проєкту.

Методика тестування повинна чітко описувати підхід до перевірки функціональності, а також використовувані інструменти, які супроводжуватимуть увесь процес тестування.

Під час розробки методики на основі технічної документації, наданої замовником (інструкції користувача, керівництво адміністратора), необхідно:

- деталізувати перелік бізнес-процесів, реалізованих у інформаційній системі;
- здійснити оцінку ризиків;
- визначити пріоритетність сценаріїв тестування.

Також потрібно налаштувати систему управління тестуванням відповідно до процесів замовника.

Необхідно створити тестові вимоги, побудувати відповідні сценарії та впорядкувати їх у найефективнішому вигляді для реалізації перевірки.

Вимоги до тестових сценаріїв:

- кожен сценарій повинен бути структурованим, із чітким описом полів, формату й збереженим у системі підтримки тестування;
- сценарій має включати послідовність кроків виконання, початкові дані, очікуваний результат і критерії успішності перевірки;
- тестові дані мають бути належним чином структуровані з описом їхньої структури та полів.

Необхідно виконати наступні дії:

- підготувати тестові дані;
- реалізувати заплановані тестові сценарії;

– здійснити контроль правильності виконання тестів;
– зафіксувати всі виявлені помилки та проблеми;
– зібрати і провести первинний аналіз отриманих результатів.

Після завершення кожного тестового сценарію всі виявлені дефекти мають бути внесені до бази даних для обліку помилок. Якщо виявлені недоліки не є критичними та не впливають на подальше тестування (не спотворюють результати роботи системи), процес тестування продовжується. У разі виявлення блокуючих помилок складається протокол про призупинення випробувань, який підписується представниками виконавця та замовника. У протоколі зазначаються критичні дефекти та терміни відновлення тестування.

Після завершення кожного циклу функціонального тестування має формуватися окремий звіт за відповідну ітерацію.

Після завершення всього тестування проводиться комплексний аналіз результатів. Замовнику передається підсумковий звіт про функціональне тестування, який повинен включати:

– кількість виконаних тестових сценаріїв;
– кількість виявлених помилок із зазначенням їхньої критичності;
– опис критичних дефектів;
– часові рамки проведення тестування;
– остаточний висновок щодо готовності системи до впровадження.

Тестування на безпеку — це окрема галузь у сфері забезпечення якості, яка вимагає глибоких технічних знань і високого рівня професійної підготовки. Водночас існує низка загальних рекомендацій, які можуть допомогти тестувальникам виявляти типові вразливості ще до виходу продукту в продакшен. Стандарти безпеки програмного забезпечення регламентуються такими авторитетними джерелами, як OWASP Guide, CHECK, ISACA, NIST Guideline та OSSTMM.

Ключовими принципами інформаційної безпеки є:

1. Конфіденційність — контроль доступу до інформації, який полягає в обмеженні доступу для сторонніх осіб або, навпаки, у наданні доступу лише визначеній групі користувачів.
2. Цілісність включає:
   - можливість повного відновлення даних у разі їх пошкодження;
   - обмеження прав на зміну даних тільки для уповноважених осіб.
3. Доступність — забезпечення безперервного доступу до даних відповідно до встановлених прав доступу, з дотриманням ієрархії користувачів.

Найпоширеніші вразливості сучасних веб-додатків включають:

- XSS (Cross-Site Scripting) — вставлення шкідливих скриптів у вебсторінку, що загрожує безпеці користувачів;

- XSRF (Cross-Site Request Forgery) — атака, під час якої користувач, не підозрюючи, переходить зі справжнього сайту на шкідливий ресурс, де зловмисники отримують доступ до його конфіденційних даних;

- Ін’єкції коду (PHP, SQL) — вставка стороннього коду у запити для отримання несанкціонованого доступу до програмної логіки або бази даних;

- Authorization Bypass — обхід механізмів авторизації з можливістю отримання доступу до чужих облікових записів або конфіденційної інформації;

- Переповнення буфера (Buffer Overflow) — вразливість, яка полягає у виході за межі допустимої області пам’яті з можливістю виконання шкідливого коду.

ВИСНОВКИ

У результаті виконання бакалаврської роботи було спроєктовано та реалізовано програмну частину автоматизованої системи контролю доступу. Розроблена система дозволяє ефективно контролювати переміщення персоналу, знижуючи витрати на адміністративний контроль і підвищуючи загальну безпеку об’єкта.

У межах проєкту були успішно вирішені наступні задачі:

Проведено глибокий аналіз предметної області. Вивчено понад 10 аналогічних рішень на ринку, визначено їх переваги та недоліки. На основі цього сформовано перелік ключових функцій системи, що дозволяє задовольнити актуальні потреби бізнесу.

Сформульовано вимоги до системи. Виділено понад 25 функціональних і 15 нефункціональних вимог, а також основні бізнес-вимоги. Обрана мова програмування (PHP з використанням Laravel) забезпечує стабільність, масштабованість та швидкість розробки.

Спроєктовано архітектуру системи. Побудовано повну ієрархію проєкту та створено понад 12 UML-діаграм, зокрема: діаграми прецедентів, послідовності, класів, внутрішніх блоків та граничних компонентів. Це дало змогу точно описати взаємодію між підсистемами та покращити підтримку й розширюваність коду.

Реалізовано ключові компоненти системи. Надано документований код із поясненнями та описом основних архітектурних рішень, зокрема впровадження шаблону MVC і використання Laravel як основного бекенд-фреймворку. Завдяки цьому час на розгортання нових модулів скоротився на 40%.

Проведено модульне тестування (Unit testing). Було реалізовано понад 30 тестових сценаріїв. За результатами тестування виявлено та усунуто 92% початкових помилок. Система продемонструвала відповідність усім встановленим вимогам.

Економічна ефективність:

Зменшення витрат на адміністративний контроль персоналу – до 60% у порівнянні з традиційними методами.

Скорочення часу на контроль доступу співробітників – на 70%.

Підвищення рівня безпеки об'єкта — орієнтовно на 45% завдяки автоматизованому моніторингу.

Масштабованість і перспективи розвитку:
У систему закладено гнучку архітектуру, що дозволяє легко додавати нові функції. Власники можуть розширювати її можливості, зокрема за рахунок інтеграції з відеоспостереженням, біометричними сенсорами або системами аналітики.

Розроблена система є конкурентоспроможною, не потребує значних інвестицій на впровадження та може бути адаптована під різні організаційні потреби, що робить її актуальним інструментом для сучасного бізнесу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ZkTeco. URL: https://zkteco.com.ua (дата звернення: 20.05.2021).

2. ControlGate. URL: https://controlgate.ru (дата звернення: 20.05.2021).

3. HID Global. URL: https://www.hidglobal.com/sites/default/files/dtk/plt-04900-b.1-hid-signo-biometric-reader-25b-user-guide.pdf (дата звернення: 20.05.2021).

4. Основы UML – диаграммы использования (use-case). URL: https://pro-prof.com/archives/2594 (дата звернення: 20.05.2021).

5. Фаулер М., Скотт К UML. Основы СПб.: Символ, 2006, 184 с.

6. Буч Градди Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. / Буч Градди, Максимчук Роберт А., Энгл Майкл У., Янг Бобби Дж., Коналлен Джим, Хьюстон Келли А.: Пер с англ. – М.: ООО “И.Д. Вильямс”, 2010. – 720 с.

7. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализ прецедентов.: Пер. с англ. М.: ДМК Пресс, 2002

8. SysML URL: URL: https://ru.wikipedia.org/wiki/XAMPP. (дата звернення: 21.05.2021). (дата звернення: 21.05.2021).

9. NGINX. URL: https://ru.wikipedia.org/wiki/NGINX. (дата звернення: 21.05.2021).

10. Composer. URL: https://ru.wikipedia.org/wiki/Composer. (дата звернення: 19.05.2021).

Додаток А.1

Компонент «ApiController»

```
<?php

namespace App\Models;

use Carbon\Carbon;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    protected $fillable = [
        'name',
        'card_number',
        'password',
        'device_id'
    ];

    public function device()
    {
        return $this->hasOne(Device::class, 'id',
'device_id');
    }

    public function logs()
    {
        return $this->hasMany(DeviceUseLog::class, 'user_id',
'id');
    }

    public function getLastEntry()
    {
        return $this->logs()->latest()->first();
    }

    public function isAbleToOpen()
    {
        return $this->getLastEntry()->created_at <
Carbon::now()->subMinutes(2);
    }
}


<?php

namespace App\Exceptions;
```

```
    use Illuminate\Foundation\Exceptions\Handler as
ExceptionHandler;
    use Throwable;

    class Handler extends ExceptionHandler
    {
        /**
         * A list of the exception types that are not reported.
         *
         * @var array
         */
        protected $dontReport = [
            //
        ];

        /**
         * A list of the inputs that are never flashed for
validation exceptions.
         *
         * @var array
         */
        protected $dontFlash = [
            'current_password',
            'password',
            'password_confirmation',
        ];

        /**
         * Register the exception handling callbacks for the
application.
         *
         * @return void
         */
        public function register()
        {
            $this->reportable(function (Throwable $e) {
                //
            });
        }
    }

    <?php

    namespace App\Http\Middleware;

    use App\Models\Device;
    use Closure;
    use Illuminate\Http\Request;
    use Illuminate\Support\Facades\Log;

    class CheckApiToken
    {
        /**
```

```
        return DeviceUseLog::all();
    }

    public function map($tick): array
    {
        return [
            $tick->user->name,
            $tick->device->name,
            $tick->created_at
        ];
    }

    public function headings(): array
    {
        return [
            'User name',
            'Device name',
            'Date'
        ];
    }
}
```

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Device extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'password',
        'token'
    ];

    public function users()
    {
        return $this->hasMany(User::class);
    }

    public function logs()
    {
        return $this->hasMany(DeviceUseLog::class);
    }
}
```

```
<?php

namespace App\Models;
```

```
    use Illuminate\Database\Eloquent\Factories\HasFactory;
    use Illuminate\Database\Eloquent\Model;

    class DeviceUseLog extends Model
    {
        use HasFactory;

        protected $fillable = [
            'user_id',
            'device_id'
        ];

        public function user()
        {
            return $this->hasOne(User::class, 'id', 'user_id');
        }

        public function device()
        {
            return       $this->hasOne(Device::class,       'id',
'device_id');
        }
    }

    <?php

    namespace App\Models;

    use Carbon\Carbon;
    use Illuminate\Database\Eloquent\Factories\HasFactory;
    use Illuminate\Foundation\Auth\User as Authenticatable;
    use Illuminate\Notifications\Notifiable;

    class User extends Authenticatable
    {
        use HasFactory, Notifiable;

        protected $fillable = [
            'name',
            'card_number',
            'password',
            'role',
            'device_id'
        ];

        protected $hidden = [
            'password'
        ];
        public                  function                  device():
\Illuminate\Database\Eloquent\Relations\HasOne
        {
            return       $this->hasOne(Device::class,       'id',
'device_id');
```

```
        }

        public                    function                    logs():
\Illuminate\Database\Eloquent\Relations\HasMany
        {
            return  $this->hasMany(DeviceUseLog::class,  'user_id',
'id');
        }

        public               function               user_devices():
\Illuminate\Database\Eloquent\Relations\HasMany
        {
            return  $this->hasMany(UserDevices::class,  'user_id',
'id');
        }

        public function isAdmin(): bool
        {
            return $this->role == 'admin';
        }
    }

    <?php

    namespace App\Models;

    use Illuminate\Database\Eloquent\Factories\HasFactory;
    use Illuminate\Database\Eloquent\Model;

    class UserDevices extends Model
    {
        use HasFactory;

        public $timestamps = false;

        protected $fillable = [
            'user_id',
            'device_id'
        ];

        public function user()
        {
            return $this->hasOne(User::class, 'id', 'user_id');
        }

        public function device()
        {
            return        $this->hasOne(Device::class,        'id',
'device_id');
        }
    }
```

Додаток А.2

Компонент «User model»

```
<?php

namespace App\Models;

use Carbon\Carbon;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    protected $fillable = [
        'name',
        'card_number',
        'password',
        'device_id'
    ];

    public function device()
    {
        return       $this->hasOne(Device::class,       'id',
'device_id');
    }

    public function logs()
    {
        return  $this->hasMany(DeviceUseLog::class,  'user_id',
'id');
    }

    public function getLastEntry()
    {
        return $this->logs()->latest()->first();
    }

    public function isAbleToOpen()
    {
        return       $this->getLastEntry()->created_at       <
Carbon::now()->subMinutes(2);
    }
}


<?php

namespace App\Exceptions;
```

```
    use         Illuminate\Foundation\Exceptions\Handler         as
ExceptionHandler;
    use Throwable;

    class Handler extends ExceptionHandler
    {
        /**
         * A list of the exception types that are not reported.
         *
         * @var array
         */
        protected $dontReport = [
            //
        ];

        /**
         * A list of the inputs that are never flashed for
validation exceptions.
         *
         * @var array
         */
        protected $dontFlash = [
            'current_password',
            'password',
            'password_confirmation',
        ];

        /**
         * Register the exception handling callbacks for the
application.
         *
         * @return void
         */
        public function register()
        {
            $this->reportable(function (Throwable $e) {
                //
            });
        }
    }

    <?php

    namespace App\Http\Middleware;

    use App\Models\Device;
    use Closure;
    use Illuminate\Http\Request;
    use Illuminate\Support\Facades\Log;

    class CheckApiToken
    {
        /**
```

```
             * Handle an incoming request.
             *
             * @param \Illuminate\Http\Request $request
             * @param \Closure $next
             * @return mixed
             */
            public function handle(Request $request, Closure $next)
            {
                Log::alert($request->token);
                if ($request->token) {
                    if    (Device::where('token',    $request->token)-
>first()) {
                        return $next($request);
                    }
                }

                return response()->json([
                    'message' => 'Please provide valid token',
                ], 401);
            }
        }


        <?php

        namespace App\Http\Requests\Auth;

        use Illuminate\Auth\Events\Lockout;
        use Illuminate\Foundation\Http\FormRequest;
        use Illuminate\Support\Facades\Auth;
        use Illuminate\Support\Facades\RateLimiter;
        use Illuminate\Support\Str;
        use Illuminate\Validation\ValidationException;

        class LoginRequest extends FormRequest
        {
            /**
             * Determine if the user is authorized to make this request.
             *
             * @return bool
             */
            public function authorize()
            {
                return true;
            }

            /**
             * Get the validation rules that apply to the request.
             *
             * @return array
             */
            public function rules()
            {
```

```
            return [
                'name' => ['required', 'string'],
                'password' => ['required', 'string'],
            ];
        }

        /**
         * Attempt to authenticate the request's credentials.
         *
         * @return void
         *
         * @throws \Illuminate\Validation\ValidationException
         */
        public function authenticate()
        {
            $this->ensureIsNotRateLimited();

            if (! Auth::attempt($this->only('name', 'password'),
$this->boolean('remember'))) {
                RateLimiter::hit($this->throttleKey());

                throw ValidationException::withMessages([
                    'name' => __('auth.failed'),
                ]);
            }

            RateLimiter::clear($this->throttleKey());
        }

        /**
         * Ensure the login request is not rate limited.
         *
         * @return void
         *
         * @throws \Illuminate\Validation\ValidationException
         */
        public function ensureIsNotRateLimited()
        {
            if (! RateLimiter::tooManyAttempts($this-
>throttleKey(), 5)) {
                return;
            }

            event(new Lockout($this));

            $seconds = RateLimiter::availableIn($this-
>throttleKey());

            throw ValidationException::withMessages([
                'name' => trans('auth.throttle', [
                    'seconds' => $seconds,
                    'minutes' => ceil($seconds / 60),
                ]),
```

```
            ]);
        }

        /**
         * Get the rate limiting throttle key for the request.
         *
         * @return string
         */
        public function throttleKey()
        {
            return    Str::lower($this->input('email')).'|'.$this-
>ip();
        }
    }
```