

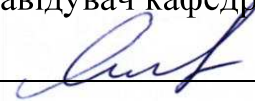
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ



к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "магістр"

на тему:

Системний аналіз технологій створення комплексів контролю доступу на
основі розподілених систем

Виконав: студент групи 6145 м



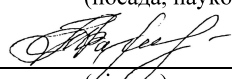
(підпис)

Шведов В.Г.

Керівник роботи:

К.Т.Н., ДОЦЕНТ

(посада, науковий ступінь вчене звання)



(підпис)

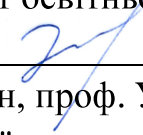
Фаріонова Т.А.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій
Спеціальність 124 "Системний аналіз"
Освітня програма "Системний аналіз"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми


д.ф-м.н, проф. Ушкац М.В.

"___" _____ 2024 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ на здобуття ступеня вищої освіти "магістр"

Студенту Шведову Владиславу Геннадійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Системний аналіз технологій створення комплексів контролю доступу на основі розподілених систем

Керівник роботи _____

Затверджені наказом ректора № 1055-уч. від "08" жовтня 2024 року.

2. Термін подання роботи: 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем, матеріали наукового стажування

4. Перелік питань, що належать до розробки (найменування розділів)

Дослідження засобів розподілених систем контролю доступу, аналіз предметної області та постановка задачі, Аналіз особливостей СКУД, Класифікація СКУД, Огляд методів реалізації СКУД, Вимоги до СКУД, що створюється, Проект інформаційного забезпечення, проект програмного забезпечення, проект технічного забезпечення, Побудова прототипу системи, Розрахунок економічної ефективності, Охорона праці та навколишнього середовища

5. Перелік презентаційних матеріалів Основні поняття та загальна схема СКУД, Види СКУД, Обробка події в СКУД, Послідовність операцій при обробці подій в СКУД, Діаграми діяльності в СКУД, Концептуальна модель даних, Фізична модель даних, Діаграма класів, Діаграма розгортання системи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Фаріонова Т.А	05.09.2024	10.12.2024
2	Фаріонова Т.А.	05.09.2024	10.12.2024
3	Фаріонова Т.А.	05.09.2024	10.12.2024
4	Фаріонова Т.А	05.09.2024	14.11.2024
5	Фаріонова Т.А.	05.09.2024	14.11.2024
6	Фаріонова Т.А.	05.09.2024	14.11.2024

7. Дата видачі завдання 05 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Дослідження та аналіз предметної галузі		
2	Розробка концепції		
3	Розділ з розробки проектних рішень по системі (та її частинах)		
4	Реалізація проекту та розробка робочої документації		
5	Обчислення економічної ефективності		
6	Вирішення питань охорони праці та навколишнього середовища		
7	Оформлення пояснювальної записки		
8	Подання магістерської роботи на перевірку на плагіат		
9	Подання магістерської роботи рецензенту		
10	Подання магістерської роботи на державний захист		

Студент


(підпис)

Шведов В.Г.

(ПІБ)

Керівник роботи


(підпис)

Фаріонова Т.А.

(ПІБ)

Анотація

Представлена магістерська робота присвячена дослідженню технологій та засобів створення інформаційних комплексів ідентифікації осіб з селекцією доступу на основі використання розподілених систем. В процесі роботи був проведений аналіз існуючих технологій та систем контролю управління доступом, визначені особливості та основні вимоги. Розроблено структуру системи, модель даних.

Система була реалізована на мові програмування Java.

Магістерська робота включає в себе _ сторінок, _ ілюстрацій, _ додатків.

Ключові слова: Java, СКУД, модель даних, верифікація, ідентифікація

Abstract

This master diploma work was written to develop and analyzing of information complex, which was madden for identifying person with access selection based on distributed system. During working time was analyzed existing system of ACS, was developed system structures and data models.

The system was implemented in the programming language Java.

Master diploma work includes pages __ illustrations _ applications.

Keywords: Java, ACS, data model, verification, identification

ЗМІСТ

ВСТУП

1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ РОЗПОДІЛЕНИХ СИСТЕМ КОНТРОЛЮ ДОСТУПУ, АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз існуючих засобів ідентифікації осіб у системах контролю управління доступу (СКУД)

1.2 Аналіз особливостей СКУД

1.3 Класифікація СКУД

1.3.1 Автономні СКУД

1.3.2 Централізовані (мережеві) СКУД

1.3.3 Класи функціональних характеристик СКУД

1.4 Огляд методів реалізації

1.5 Формування вимог до СКУД

1.6 Постановка задачі

2 РОЗРОБКА ПРОЕКТНИХ РІШЕНЬ СИСТЕМИ КОНТРОЛЮ ДОСТУПУ

2.1 Проект інформаційного забезпечення

2.1.1 Розробка діаграми варіантів використання

2.1.2 Розробка діаграми послідовності

2.1.3 Розробка діаграми діяльності

2.1.4 Модель даних системи контролю доступу

2.1.5 Фізична модель

2.2 Проект технічного забезпечення

2.2.1 Розробка діаграми класів

2.2.2 Розробка діаграми пакетів

2.2.3 Розробка діаграми розгортання

2.3 Проект програмного забезпечення

2.3.1 Мова програмування

2.3.2 База даних

2.3.3 Аспектно-орієнтоване програмування (АОП)

2.3.4 Середовища розробки програми

3 РЕЗУЛЬТАТИ РОЗРОБКИ

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Вступ

4.2 Розрахунок витрат на створення та експлуатацію інформаційної системи

4.3 Економічна ефективність розробки і впровадження системи

5 ОХОРОНА ПРАЦІ

5.1 Загальні визначення та актуальність питання охорони праці

5.2 Аналіз небезпечних і шкідливих факторів при роботі з ПК

5.3 Розрахунок системи загального освітлення приміщення з ПК

5.4 Розробка заходів щодо забезпечення сприятливих умов праці при роботі з ПК

6 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

6.1 Вступна частина

6.2 Забруднення навколишнього середовища від підприємства виготовлення паперу

6.3 Заходи збереження природних ресурсів

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ

ДОДАТОК Б – ТЕХНІЧНИЙ ОПИС

ДОДАТОК В – ТЕКСТ ПРОГРАМНИХ МОДУЛІВ

ВСТУП

В даний час для багатьох організацій і підприємств стало характерним питання розподілення контролю доступу. Контроль доступу - це перший ешелон системи безпеки компанії, який дозволяє розмежовувати і регулювати обіг цінного обладнання та доступ співробітників до конфіденційної службової інформації.

Особливо ця проблема стала актуальною для великих організацій де порушення безпеки може завдати величезних матеріальних збитків, як самим організаціям, так і їх клієнтам. Тому ці організації змушені особливу увагу приділяти гарантіям безпеки. В наслідок чого виникла проблема захисту і контролю доступу в приміщення. І зараз ця проблема являє собою сукупність тісно пов'язаних підпроблем в галузях права, організації, управління, розробки технічних засобів, програмування і математики. Одна з центральних завдань проектування системи захисту полягає в надійному і ефективному управлінні доступом на об'єкт захисту.

Система контролю управління доступом (СКУД)– це системи які оточують нас, буквально, всюди. Найпростішим прикладом розповсюдженої системи контролю доступу є під'їздний домофон, який дозволяє власникові квартири здійснювати пропуск осіб у під'їзд за допомогою дистанційного керування замикаючим пристроєм. Отож, звернемося до історії створення.

Найпростіша початкова СКУД - це звичайні двері з механічним замком і оптичним "глазком" - проста і дуже надійна система контролю доступу. До цього ж розряду систем доступу відноситься і звичайний механічний шлагбаум і ланцюгові загородження, які піднімаються охоронцем вручну. Контроль тут здійснюється безпосередньо людиною.

Далі слідує "кодовий замок" - поєднання найпростішого контролера з кнопковою клавіатурою, що працює на відкриття електромеханічного (електромагнітного) замка дверей.

З розвитком технологій з'являється ще один найпростіший контролер - керований "таблеткою" Touch Memory зі спеціального зчитувача за принципом

програмування "свій - чужий", що дозволяє відкриття електромагнітного замка дверей тільки "таблеткам", які внесені до його пам'яті.

Отже, система контролю і управління доступом - це сукупність технічного обладнання та програмних засобів, спрямованих на забезпечення контролю над переміщенням співробітників, відвідувачів чи транспорту по території об'єкту, що охороняється.

Предметом даної роботи є аналіз існуючих систем контролю та управління доступом, їх реалізація та принципи роботи.

Найпростішими електронними системами доступу є кодонабірні панелі або автономні зчитувачі карток. Більш складні системи, що включають, як правило, один або кілька комп'ютерів, можуть контролювати доступ в залежності від поточного часу, дня тижня і святкових днів. Така система веде протокол всіх подій, що відбулися: проходів певного користувача через певні двері в певний час, спроб несанкціонованого доступу і т.д.

Програмне забезпечення дозволяє отримати різні види звітів про події, що відбулися в системі за певний відрізок часу. Крім того, різні системи можуть мати широкий набір додаткових функцій, наприклад, захист від несанкціонованого проникнення в приміщення шляхом передачі карти іншій особі і т.д.

Для організацій з великим числом співробітників і відвідувачів в рамках системи контролю доступу можуть бути створені робочі місця для оперативного випуску пропусків. Дані про відвідувачів або нових співробітників вносяться в базу даних системи контролю доступу, визначаються їх повноваження по доступу в певні приміщення і видаються картки-перепустки. Час дії цих карт може бути обмеженим.

При всьому різноманітті можливих структур побудови систем контролю доступу всі вони побудовані на базі ідентифікації чого-небудь.

Процедура управління доступом зводиться до взаємного впізнання користувача і системи і встановлення факту допустимості використання ресурсів конкретним

користувачем відповідно до його запиту. Засоби управління доступом забезпечують захист об'єкта, який охороняється як від неавторизованого використання, так і від несанкціонованого обслуговування системою. Захист реалізується процедурами ідентифікації, встановлення автентичності та реєстрації звернень. [1]

Основною метою даної роботи є розробка інформаційного комплексу ідентифікації осіб з селекцією доступу на основі використання розподілених систем.

Для досягнення поставленої мети в роботі необхідно розробити:

- Проект інформаційного забезпечення
 - Розробка діаграми варіантів використання
 - Розробка діаграми послідовності
 - Розробка діаграми діяльності
 - Модель даних системи контролю доступу
 - Фізична модель
- Проект технічного забезпечення
 - Розробка діаграми класів
 - Розробка діаграми пакетів
 - Розробка діаграми розгортання
- Проект програмного забезпечення
 - Мова програмування
 - База даних
 - Аспектно-орієнтоване програмування (АОП)
 - Середовища розробки програми

1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ РОЗПОДІЛЕНИХ СИСТЕМ КОНТРОЛЮ ДОСТУПУ, АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз та дослідження існуючих систем контролю управління доступу (СКУД)

В фізичній безпеці термін «контроль доступу» має на увазі обмеження доступу до власності, в будівлі або приміщення, доступ до яких дозволено тільки для авторизованих людей. Фізичний контроль доступу може бути досягнутий шляхом використання людини (охоронець, службовець в приймальні), через механічні способи, такі як замки на двері, або через технологічні засоби, такі як системи доступу, засновані на ідентифікаторах - картах доступу, електронних ключах, штрих-кодах, біометричній інформації.

Для багатьох організацій СКУД це не тільки дань моді, а й гостра необхідність в сучасній системі, яка дозволяє:

- забезпечити додаткову безпеку підприємству;
- зменшити затрати на фізичну охорону;
- надати ієрархічний доступ по групам доступу;
- отримувати різні звіти в необхідному форматі для статистики, служби безпеки чи кадрової служби;
- при наявності функції контролю робочого часу – отримувати інформації про співробітників, які запізнились чи взагалі були відсутні на роботі.

Нерідко головним критерієм вибору системи є вартість обладнання і програмного забезпечення. В багатьох випадках ціна обладнання, яку пропонують фірми, може скласти суму, що в багато разів перевищує первісну суму, затрачену на обладнання і програмне забезпечення. Так, наприклад, модернізація системи може спричинити заміну «слабкого» контролера, купівлю та установку додаткових програмних модулів та оплату за модернізацію системи.

Таким чином при розробці системи санкціонованого доступу потрібно звертати увагу на таку характеристику як модульність. Модульність дозволяє створювати систему під індивідуальні вимоги кожного замовника. Перевагами модульності є :

- гарантія максимальної ефективності вкладених коштів - ніякої переплати за надмірний функціонал, який йде «в комплекті», але використовуватися не буде;
- мінімізація «людського фактора» при роботі з інтерфейсом системи дозволяє створювати індивідуальні налаштування під кожного конкретного користувача (оператора) і задачу. Таким чином виключається наявність в інтерфейсі клієнтської частини «зайвих», елементів, які безглуздо перевантажують інтерфейс, відволікаючи на себе увагу оператора і тим самим впливають на зручність роботи з системою.

Особливу увагу при виборі СКУД потрібно звернути на наступні характеристики, які можна поділити на дві групи – технічні та нетехнічні.

До технічних характеристик відносять:

- модульність;
- об'єм вбудованої пам'яті, яка дозволяє контролеру пам'ятати події в режимі off-line, допустимі штрих-коди чи номери карток, групи доступу співробітників і т.ін.. Вимірюється в кількості подій.
- інтерфейс для зв'язку з комп'ютером (RS-232, RS-485, Ethernet);
- інтерфейс для зв'язку між контролерами;
- можливість інтеграції з іншими системами (пожежними, охоронними, відео нагляду та ін..), в тому числі імпорту даних;
- кількість точок проходу для одного контролера.

До нетехнічних характеристик відносять:

- функція «Антипасбек»;
- розмежування доступу по групах співробітників;
- аварійне відкриття дверей, турнікетів;

- експорт транзакцій;
- облік робочого часу.

Для аналізу були обрані наступні СКУД.

1) SkiData Turnstile “Arena.Gate” – система з високим рівнем безпеки, яка використовує турнікети у повний ріст. Гарне рішення для стадіонів і місць відпочинку та розваг, яке пропонує оптимальний захист від незаконного доступу. Всі квитки зчитуються та перевіряються на місці.

2) PERCo-S-600 – система побудована на основі мережі контролерів, які підключаються до комп’ютера. Система дозволяє обладнати прохідні і двері функціями контролю доступу і обліку робочого часу.

3) Nedap AEOS – потужна, зручна для користувача система контролю доступу та управління безпекою. Структура системи AEOS є повністю модульною, що забезпечує її максимальну гнучкість і розширюваність. Це означає можливість задоволення практично будь-якої вимоги до функціонала і рівню безпеки системи, як для апаратних модулів, так і для модулів програмного забезпечення. Завдяки гнучкості структури систему при необхідності легко модернізувати і розширювати за переліком необхідного функціоналу або при зміні рівня безпеки об’єкта..

4) GATE-4000 – багатоцільова СКУД, яку можна використовувати для управління та контролю доступу на підприємстві, рішення задач ідентифікації, обліку робочого часу та ін.. Контролер СКУД може застосовуватись для однієї двохсторонньої чи двох односторонніх точок проходу. Також систему можна інтегрувати з пожежною, охоронною системами та системами відео нагляду.

5) Keri Systems – комплекс апаратних і програмних засобів СКУД Keri Systems дозволяє налагодити контроль за проходом на території, що охороняється, виготовляти перепустки, вести облік робочого часу персоналу, складати різні звіти, проводити моніторинг об’єкту в реальному часі. Система здатна оброблювати інформацію про переміщення відвідувачів чисельністю від 50 до 65000 чоловік, на один контролер.

Наведемо порівняльну таблицю (дивись таблиця 1.1) основних характеристик СКУД.

Таблиця 1.1. - Порівняльна таблиця основних характеристик СКУД

СКУД		Arena. Gate	PERCo- S-600	Nedap AEOS	GATE- 4000	Keri Systems
Технічні характеристи ки	Модульність	-	+	+	-	-
	Об'єм пам'яті контролера	1200	3500	4100	4 095	3600
	Інтерфейс для зв'язку з комп'ютером	-	RS-232	Ethernet/ RS-232	-	RS-232
	Інтерфейс для зв'язку між контролерами	-	RS-485	VPN	RS- 485/RS -422	RS-485
	Інтеграція з іншими системами	-	+	+	+	+
	Кількість точок доступу	1	8	32	254	256
Нетехнічні характеристи ки	Функція «Антипасбек»	+	+	+	+	+
	Доступ по групам	-	+	+	-	+
	Аварійне розблокування	+	-	+	-	-
	Експорт транзакцій	+	-	+	-	-
	Облік робочого часу	-	+	+	+	+

По результатам порівняння основних характеристик СКУД можна зробити висновок, що найкращою за наведеними параметрами є система Nedap AEOS. Наприклад, в порівнянні з Arena.Gate від SkiData, комплекс має можливість масштабування, так як є інтерфейс зв'язку між контролерами там з комп'ютером. Масштабування дозволяє використовувати дану систему не тільки для стадіону, а

й для об'єктів будь-якої категорії і рівня складності: від аеропортів та об'єктів нафтогазової галузі з найскладнішою структурою і найжорстокішими вимогами безпеки, до невеликих офісів і окремих автономних об'єктів.

Контроль доступу для підрядників і субпідрядників, з авторизацією в залежності від робіт і дозволів надає системі додаткової зручності при роботі та чітко розмежовує функції для осіб, які її використовують .

1.2 Аналіз особливостей СКУД

Здійснювати контроль доступу на об'єкт, спостерігати і фіксувати всі переміщення відвідувачів, вести облік робочого часу і при цьому залишатися практично непомітним, не робити помилок і не ставити зайвих питань - саме так повинен виглядати ідеальний охоронець. Однак щоб при цьому ще виключити і людський фактор, доцільно використовувати системи контролю доступу, які з успіхом виконують всі ці функції.

Отож, будемо розуміти під СКУД об'єднані в комплекси електронні, механічні, електротехнічні, апаратно-програмні та інші засоби, що забезпечують можливість доступу певних осіб в певні зони (територія, будівля, приміщення) і обмежують доступ особам, які не мають такого права.

Інтерес до систем контролю і управління доступом зростає ще й тому що наявність такої системи має важливе значення для ефективної роботи підприємства чи об'єкта, де вона встановлена.

Контроль не тільки істотно підвищує рівень безпеки, але і дозволяє оперативно реагувати на поведінку персоналу і відвідувачів. Також важливим завданням для багатьох підприємств є необхідність контролювати графік і вести облік робочого часу. Особлива увага приділяється системам, що дозволяє вибудовувати необхідні графіки чи діаграми, враховуючи критерії, що заздалегідь налаштував користувач в залежності від особливостей підприємства.

На сьогоднішній день існує дуже багато різновидів СКУД різних виробників, а також її компонентів. Незважаючи на унікальність кожної конкретної системи

контролю доступу, вона містить 4 основних елемента: ідентифікатор користувача (карта-перепустка, ключ, код), пристрій ідентифікації, керуючий контролер і виконавчі пристрої. Загальна схема зображена на рисунк 1.1

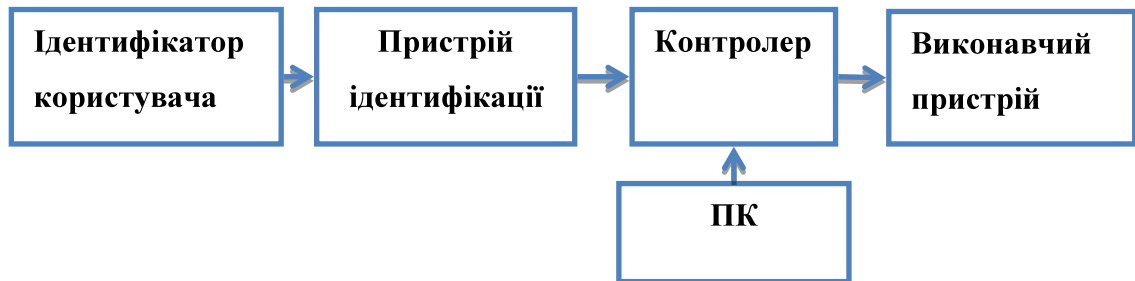


Рисунок 1.1. - Загальна схема СКУД

Взагалі, роботу системи контролю та управління доступом можна в спрощеному вигляді описати таким чином. Кожен співробітник відвідувач організації отримує ідентифікатор – електронний ключ, картку або брелок, які містять собі індивідуальний код.

Біля входу в будівлю або в приміщення, яке підлягає контролю, встановлюються зчитувачі, які зчитують інформацію, надану користувачем і передають цю інформацію в контролер системи.

У системі кожному коду поставлена у відповідність інформація про права власника ідентифікатора. На основі зіставлення цієї інформації і ситуації, при якій був пред'явлений ідентифікатор, система приймає рішення: контролер відкриває або блокує двері (замки, турнікети). Всі факти пред'явлення ідентифікаторів та пов'язані з ними дії (проходи, тривоги і т.д.) фіксуються в контролері і зберігаються в комп'ютері. Інформація про події може бути використана надалі для отримання певних звітів.

В даний час системи охорони та контролю доступу піддаються інтелектуалізації. Іншими словами, фахівці працюють над передачею максимально можливої кількості функцій системи по збору і обробки інформації, а також прийняттю рішень комп'ютерам або апаратних засобів.

Рішення про санкціонований доступ на об'єкт чи відмову в доступі приймається СКУД на основі аналізу інформації, яку пред'являє користувач системи. В залежності від виду інформації, що надає користувач, процес аутентифікації може бути реалізований різними способами.

Вхідна інформація може бути різного виду. Це може бути невід'ємна характеристика користувача, тобто біометричні дані, такі як райдужна оболонка та сітчатка ока, відбиток пальців, геометрія долоні і т.ін. Також це може бути відтворення голосом з певною інтонацією якоїсь фрази, яка становить пароль для входу чи написання деякої фрази, яке в даному випадку буде ідентифікувати особу по начерку чи особливостям написання літер. За таких обставин рішення про надання доступу на об'єкт буде прийматись на основі аналізу ідентифікаційного образу шляхом його порівняння з еталонним.

Надання доступу також може бути проведене шляхом «запит-відповідь». В такому випадку людина, для того щоб отримати доступ повинна знати пароль чи свій ідентифікаційний код в системі, або ж пред'явити код, який може бути зашитий в електронний ключ чи бути штрих-кодом картки і.

Найчастіше в ролі подібного ідентифікатора виступає пластикова картка з магнітною смугою, куди занесена особиста інформація. Проходячи через контролюючий пристрій, власник підносить картку до зчитувача, в результаті при успішному зчитуванні інформації з карти та її верифікації в системі, користувач отримує доступ на об'єкт, що охороняється. Однак такий ключ не підійде для великих компаній, так як вимагає деякого часу на здійснення контролю доступу, а значить, буде сповільнювати прохід на територію. В даному випадку кращим рішенням стане використання сучасних радіочастотних безконтактних проксиміті-карт. Також досить поширеним варіантом ключів є брелок TouchMemory, що представляє собою металеву таблетку з чіпом всередині.

СКУД, які приймають рішення по певній ознаці користувача, є найпростішими. Вони не забезпечують в повній мірі високих вимог до безпеки в

сучасних системах захисту від несанкціонованого доступу. В цілях підвищення рівня захисту об'єкта застосовуються організація доступу по сукупності декількох сутностей, які користувач повинен пред'явити системі. Так наприклад, доступ може бути наданий по пред'явленні ідентифікаційної карти із вводом PIN-коду. Більш високий рівень захищеності системи реалізується шляхом аналізу біометричних характеристик користувача в сукупності з пред'явленням ідентифікатора, що дозволяє здійснити жорстку «прив'язку» ідентифікатора до конкретного користувача. Такий метод ідентифікації використовує його характерні унікальні фізіологічні чи поведінкові особливості. Новітні системи зараз стали використовувати додаткову ідентифікацію по фотографії.

Однією з особливостей СКУД є її архітектура. Вона зазвичай буває трьох видів : розподілена, централізована, та змішана.

Розподілена система характеризується тим, що кожна будівля, що підлягає охороні повинна мати свій контролер, який забезпечуватиме автономне функціонування системи кожної будівлі чи об'єкту на випадок, якщо станеться порушення зв'язку між окремими об'єктами, що є її плюсом. Тобто виведення із строю одного контролера не вплине на роботу інших.

Відмінна особливість СКУД з такою архітектурою це те, що база даних ідентифікаторів, всі дані про події в системі містяться на кожному окремому контролеру, який встановлений на об'єкті, а не на одному, як це зроблено в централізованій архітектурі.

Контролери розподіленої СКУД зазвичай виконують керування зовнішніми пристроями, такими як турнікети, реле, охоронна сигналізація.

Що стосується централізованої архітектури СКУД, то її контролери є суто логічні, на відміну від контролерів розподіленої архітектури. Вони не керують зовнішніми пристроями, замками, а також не мають входів для підключення зчитувачів СКУД.

Керувати зовнішніми пристроями в такому випадку – задача для зовнішніх модулів та релейних блоків. Потім дані отримані цими модулями передаються в центральний контролер через визначений канал передачі даних. Це може бути інтерфейс RS-485 чи мережевий інтерфейс LAN, який з'явився достатньо недавно.

Головний центральний контролер зберігає всю інформацію про події в системі та базу ідентифікаторів.

Декілька таких контролерів можна об'єднати в мережу, це дає можливість створитиСКУД достатньо великого масштабу.

Змішана архітектураСКУД – це власне об'єднання централізованої системи з розподіленою. Такі системи отримуються шляхом додавання в інтерфейсні модулі централізованої системи пам'ять для зберігання ідентифікаторів та подій.

Така архітектураСКУД має найвищий ступінь безпеки. Оскільки центральний контролер керує великим числом дверей, і пошкодження лінії зв'язку між ним і модулями управління кінцевими пристроями може призвести до блокування значної частини або навіть всієї системи. Локальний зчитувач або проміжний блок, що має вбудований буфер пам'яті, в цьому випадку переходить в автономний режим управління доступом (на своїй ділянці).

Ще однією важливою особливістюСКУД є її масштабованість. Таку систему можна оптимально налаштувати чи то для великого підприємства, чи то для малого офісу. Таким чином маючи офіс, в якому потрібно встановити 2 точки доступу, не потрібно буде витрачати гроші на купівлюСКУД, яка підійде для контролювання цілого стадіону.

Управління масштабованістю забезпечується шляхом додавання нових функцій, якщо стосується програмного забезпечення (наприклад функції архівування подій, що стались в системі), або ж шляхом додавання модулів розширення (наприклад, розширення базової пам'яті контролера шляхом додавання нового блоку пам'яті чи розширення числа вхідних портів для підключення зчитувачів).

Також ще однією особливістю масштабованості СКУД є те, що вона повинна мати відкриту архітектуру для простоти інтеграції з іншими системами (наприклад, відео нагляду чи сигналізації). Таким чином дані системи надають можливість отримання необхідної оптимальної системи для конкретного об'єкту, на якому потрібно здійснювати контроль доступу, а це в свою чергу дозволяє платити лише за функції, які дійсно будуть використовуватись на об'єкті.

1.3 Класифікація СКУД

Контроль доступу – одна із складових частин комплексного поняття, процесу забезпечення безпеки підприємства. Сучасні мережеві системи контролю доступу по своїх можливостях можуть забезпечити необхідний рівень охорони на великих об'єктах, що містять тисячі точок доступу і десятки тисяч користувачів. Окрім цього, системи служать основою для побудови інтегрованих систем безпеки, об'єднуючих охорону і пожежну сигналізацію, засоби відео контролю.

В залежності від способу управління, функцій, які виконує система, рівню захищеності системи і т.д. СКУД класифікують за такими ознаками:

- способом управління керуючими пристроями, що перекривають доступ;
- числом керуючих точок доступу;
- видом об'єктів контролю;
- рівнем захищеності системи від несанкціонованого доступу;
- функціональними ознаками.

Розглянемо більш детально кожен групу класифікації.

За способом управління керованими пристроями, що перекривають доступ (двері, турнікети, шлюзами і т.д.) СКУД поділяються на:

- автономні – для управління одним або декількома керованими пристроями без передачі інформації на центральний пульт і без контролю з боку оператора;
- централізовані (мережеві) – для управління керованими пристроями з обміном інформацією з центральним пультом і контролем та управлінням

системою з боку оператора;

- універсальні – включають функції як автономних, так і мережевих систем, що працюють в мережевому режимі під управлінням центрального пристрою управління та які переходять в автономний режим при виникненні відмов у мережевому обладнанні, в центральному пристрої або при обриві зв'язку.

За кількістю точок доступу і користувачів :

- малі – одиничні точки доступу, застосовуються для локалізованих систем контролю та керування доступу (офіси, склади);

- середні – десятки точок доступу і тисячі користувачів, застосовуються для розподілених систем (банки, підприємства, установи, готелі);

- великі – сотні точок доступу і десятки тисяч користувачів (крупні промислові підприємства, аеропорти).

За функціональними характеристиками СКУД поділяються на 4 класи:

- СКУД 1-го класу;

- СКУД 2-го класу;

- СКУД 3-го класу;

- СКУД 4-го класу.

За видом об'єктів контролю, які здійснюють:

- контроль доступу фізичних об'єктів;

- контроль доступу до інформації.

За рівнем захищеності системи від несанкціонованого доступу до інформації:

- нормальної;

- підвищеної;

- високої.

За стійкістю до руйнівних дій:

- стійкі до злому;

- стійкі до вибуху.

Демо більш детальний опис деяким видам класифікації СКУД.

СКУД за способом управління керуючими пристроями поділяються на:

- автономні — для управління одним або декількома керованими пристроями без передачі інформації на центральний контролер і без контролю з боку оператора;
- централізовані (мережеві) — для управління керованими пристроями з обміном інформацією з центральним контролером і контролем та управлінням системою з боку оператора;
- універсальні — включають функції як автономних, так і мережевих систем, що працюють в мережевому режимі під управлінням центрального пристрою управління та які переходять в автономний режим при виникненні відмов у мережевому обладнанні, в центральному пристрої або при обриві зв'язку.

1.3.1 Автономні СКУД

За допомогою автономної системи контролюється одна або кілька точок доступу. У цьому випадку інформація не передається на центральний контролер управління. Контроль з боку оператора відсутній. Співробітники організації отримують доступ тільки при наявності спеціальних карт, ключів або при введенні правильного PIN-коду за допомогою клавіатури. Інформація (код) зі зчитувача перевіряється контролером. При підтвердженні коду контролер дає доступ (відкривання дверей, підняття шлагбаума і т.д.).

Автономні (локальні) системи вирішують проблему забезпечення контролю і управління доступом в окремому приміщенні. На рисунок 1.2 наведена структурна схема автономної системи.



Рисунок 1.2. - Автономна система контролю доступу

До складу автономної системи входять:

- автономний контролер (зберігає в собі всю базу даних ідентифікаторів і управляє роботою всіх елементів системи);
- зчитувач (магнітний, штриховий);
- кнопка відкриття дверей (зазвичай використовується для відкриття замку зсередини);
- дверний контакт (забезпечує правильність роботи всієї системи);
- електрозамки, засувки, турнікети (виконавчий пристрій).

Автономні СКУД використовують на невеликих підприємствах чи об'єктах, де стоїть конкретне завдання — чітке регулювання процесу доступу. Оперативний монтаж і налаштування такої системи може здійснити навіть непрофесіонал. Обслуговування також не викликає труднощів.

Основні мінуси автономних систем зводяться до того, що вони не здатні забезпечити оперативну обробку даних, швидке отримання звітів і становлять певну складність при програмуванні. Однак зазначені недоліки автономного режиму перекриваються багатьма очевидними перевагами СКУД.

Основна перевага автономних систем — низька вартість. Відсутність дорогих елементів і кабельних з'єднань дозволяють істотно економити при створенні СКУД. Автономні системи виявляються дешевші і простіші в експлуатації, але при цьому складають конкуренцію мережевим СКУД: зчитувачі, замикаючі пристрої та інші елементи системи часто виявляються ідентичними.

1.3.2 Централізовані (мережеві) СКУД

Мережеві СКУД забезпечують управління і контроль доступу на об'єктах з територіально розподіленими точками проходу (великі підприємства, банки, порти, великі установи, і т.п.).

Місця де найчастіше необхідний контроль доступу:

- прохідні;
- складські та офісні приміщення;
- режимні об'єкти;
- об'єкти на вулиці, стоянки (ворота, шлагбауми для автостоянок).

При мережевій системі контролю доступу інформація з локальних пристроїв надходить на центральний контролер управління, ведення звітності і управління в даному випадку відбувається на сервері. Мережева система контролю доступу дозволяє отримати повну інформацію про пересування кожного співробітника, що знаходиться на контрольованій території: точка входу/виходу, час, дата, код доступу. Інформацію при необхідності можна витягти в будь-який момент.

Мережева СКУД складається з тих же елементів, що і автономна система контролю доступу, але до неї ще додається сервер. Структурна схема наведена на рисунок 1.3.

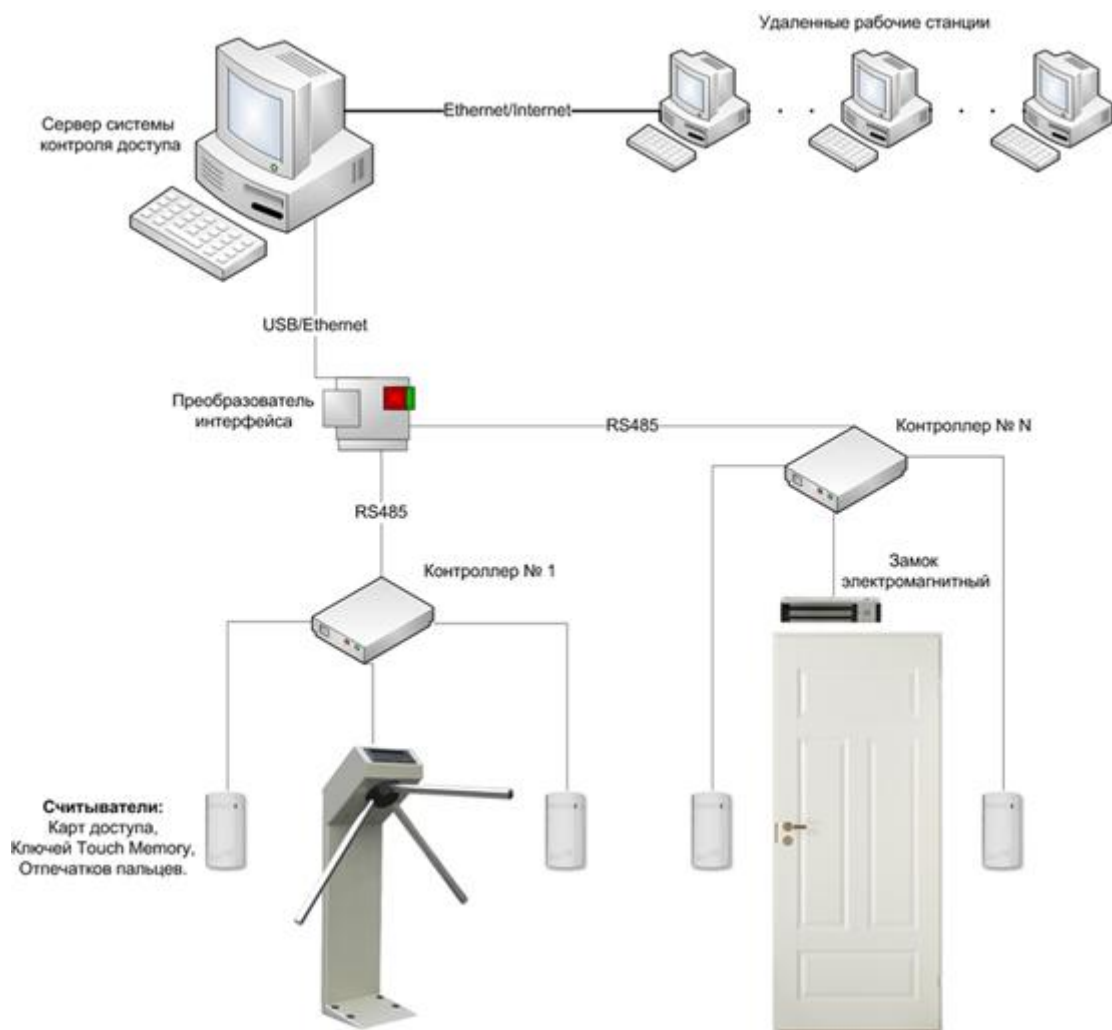


Рисунок 1.3. - Мережева система контролю доступу

В порівнянні з автономною системою, при використанні мережевої системи контролю доступу виникають додаткові можливості:

- можливість отримати деталізовану інформацію про всі переміщення персоналу та відвідувачів на території підприємства, організації;
- можливість організувати і контролювати робочий час співробітників;
- можливість відкриття всіх дверей при виникненні пожежі;
- можливість інтегрування з системою охоронної та пожежної сигналізації.

1.3.3 Класи функціональних характеристик СКУД

Розглянемо більш детально класи функціональних можливостей СКУД.

До СКУД 1-го класу відносять мало функціональні або функціональні системи, що працюють в автономному режимі та здійснюють допуск всіх

користувачів, що мають ідентифікатор. Такі системи застосовуються в тому випадку, якщо замовнику необхідно забезпечити контрольований доступ співробітників і відвідувачів, які мають відповідний ідентифікатор. При цьому не ставиться завдання контролю часу доступу і виходу з приміщення, реєстрація проходів, передача даних на центральний комп'ютер. Робота СКУД не контролюється. Зазвичай адміністратор (або особа відповідальна за пропускний режим) має право вносити в список системи коди ідентифікаторів співробітників і відвідувачів або виключати їх зі списку, а також отримувати інформацію з буфера системи.

До СКУД 2-го класу відносять монофункціональні системи, але у них вже є можливість розширення і включення їх або їх складових частин у загальну лінію зв'язку (мережевий режим). Допуск осіб може здійснюватися не лише по ідентифікатору, але ще й за датою чи певними тимчасовими інтервалами. Система здатна забезпечити автоматичну реєстрацію подій і автоматичне керування виконавчими пристроями. На об'єктах, де встановлені системами СКУД 2-го класу вони часто розглядаються лише як засоби підвищення безпеки об'єкта.

Системи 3-го класу виконують інтеграцію з іншими системами на релейному рівні. Релейний рівень передбачає наявність додаткового модуля в контролері (або додаткових входів/виходів в контролері), до якого підключаються охоронні або пожежні сповіщувачі, і релейні виходи для управління телекамерами та іншими пристроями. Подібна інтеграція застосовується, в основному, на малих об'єктах. На таких об'єктах кількість взаємодій між системами невелика, і всі вони можуть бути враховані в процесі проектування системи безпеки. Цей рівень інтеграції є простим, універсальним і досить надійним.

Системи 4-го класу це багаторівневі системи великої ємності. Відмінні особливості великих систем — наявність розвиненого програмного забезпечення, що дозволяє реалізовувати велику кількість функціональних можливостей і

високий ступінь інтеграції на програмному (системному) рівні з іншими системами охорони і безпеки.

Програмний рівень передбачає об'єднання різних систем на основі єдиної програмно-апаратної платформи, з єдиним комунікаційним протоколом і загальною базою даних

В таблиці 1.2 наведена порівняльна характеристика класів функціональних можливостей СКУД

Таблиця 1.2. – Порівняльна характеристика класів СКУД

Клас	Захист від несанкціонованого доступу	Функції	Застосування
1	Недостатній	– допуск всіх осіб, які мають відповідний ідентифікатор; – світлова/звукова індикація режимів роботи; – управління відкриттям/закриттям пристроїв загородження	На об'єктах, де потрібна лише обмеження доступу сторонніх осіб (функція замку)
2	Середній	– обмеження допуску особи, групи осіб за датою і тимчасовим інтервалам в відповідності з наявним ідентифікатором; – автоматична реєстрацію подій у власному буфері пам'яті, видача тривожних сповіщень на зовнішні оповіщувачі або внутрішній	На об'єктах, де потрібен облік і контроль присутності співробітників в дозволеній зоні. Як доповнення до наявних на об'єкті системах охорони і

		пост охорони; – автоматичне керування відкриттям/закриттям пристроїв загородження	захисту.
3	Високий	– функції СКУД 2 класу; – контроль переміщень осіб; – ведення табельного обліку і баз даних по кожному службовцю, безперервний автоматичний контроль справності складових частин системи; – інтеграція з іншими системами на релейному рівні.	На об'єктах, де потрібен табельний облік і контроль переміщень співробітників по об'єкту. Для спільної роботи з пожежними та охоронними системами .
4	Дуже високий	– функції СКУД 3 класу; – інтеграція з іншими системами безпеки та управління на програмному рівні; – автоматичне керування пристроями загородження в разі пожежі та інших надзвичайних ситуаціях.	В інтегрованих системах охорони і інтегрованих системах безпеки та управління системами життєзабезпечення.

1.4 Огляд методів реалізації

Побудова системи може здійснюватися від простого автономного контролера зі зчитувачем на одні двері до складної інтегрованої системи, що дозволяє забезпечити комплекс заходів спрямованих на збереження майна і цінностей, запобігання розкрадань фінансових і матеріально-технічних засобів, порушення працездатності підприємства та захисту персоналу і відвідувачів.

При реалізації системи контролю доступу потрібно враховувати наступні моменти.

Скільки точок проходу вона буде контролювати? Від цієї характеристики буде залежати вибір контролера. Так, якщо маємо одну точку проходу, контролер може бути вбудований в звичайний зчитувач ідентифікатора. А якщо маємо масштабний об'єкт, що охороняється, то контролерів може бути і декілька, кожен з яких може контролювати десятки точок проходу.

Важливою характеристикою при реалізації є інформація про кількість людей, які будуть користуватися СКУД та швидкодія системи. У великих багатоповерхових офісних центрах і готелях може знадобитися велика кількість точок доступу СКУД, але при малій інтенсивності проходів, невеликій кількості користувачів і операторів. Але якщо це прохідна великого заводу, то дуже важливим фактором тут стає швидкість роботи системи, так як будь які затримки в системі призведуть до скупчення людей на вході, створення черги, що буде створювати затримку в роботі заводу в цілому. Але існує варіант, коли в системі є велика кількість прохідних точок через які проходить велика кількість людей. В цьому випадку вимоги до системи зростають в геометричній прогресії. Така система повинна встигати швидко і одночасно опрацьовувати велику кількість запитів. Найбільш універсальною буде СКУД, структура якої дозволяє організувати від декількох одиниць до декількох тисяч точок доступу з різною кількістю користувачів та інтенсивністю проходів за рахунок застосування

декількох типів пристроїв керування, сумісних між собою по протоколу обміну даними та які працюють під управлінням єдиного програмного комплексу.

Ще однією особливістю реалізації є вид пристрою зчитування коду. Це можуть бути ручні зчитувачі, в яких ввід коду здійснюється за допомогою клавіш. Чи контактні зчитувачі. Зчитування коду в даному разі відбувається в залежності від того де буде встановлена СКУД, вона матиме різний ступінь надійності та стійкості до відмови роботи. Наприклад, у випадку пожежі потрібно забезпечити вільний вихід людей через контрольні точки, що гарантує вільну евакуацію людей з приміщення. І дуже важливо, щоб цей механізм спрацьовував безвідмовно.

Дуже важливим моментом є автономність загороджувачих пристроїв. Тобто можливість виконувати свої функції і вирішувати завдання навіть у разі втрати зв'язку з контролером СКУД. Ідеальним випадком в такому разі буде ведення журналу подій та його зберігання, а також резервування даних.

Плюс до цього, велику увагу з боку користувачів приділяється до такої властивості СКУД, як «заборона подвійного проходу». Тобто якщо у нас на підприємстві кілька прохідних точок, то неможливо, щоб по одному і тому ж пропуску (ключу, карті) був двічі виконаний вхід (ні через яку точку входу) до тих пір, поки не буде виконаний вихід.

В залежності від цілей установки системи контролю доступу з'являються вимоги до звітності. Якщо потрібно тільки розмежувати доступ в кілька приміщень для декількох співробітників, то можна обійтися взагалі без звітів. Однак, при використанні більш масштабної системи або використанні СКУД для обліку робочого часу вже обов'язково потрібна наявність звітності. Тут варто звернути увагу на способи отримання і збереження інформації. Перевагу варто віддавати системам з пам'яттю подій. Якщо з якихось причин інформація про події не була доставлена на комп'ютер (він був вимкнений і т.п.) всі події будуть збережені при відновленні зв'язку з комп'ютером. Вибір СКУД в даному випадку

визначається розрахунковими показниками кількості подій системи, які повинна "пам'ятати" система в автономному режимі.

Враховуючи всі вище наведені вимоги та характеристики до СКУД розглянемо основні принципи їх реалізацій.

Найпростіший варіант побудови СКУД – 1 точка доступу, зчитувач, комп'ютер. При вході в приміщення встановлюється зчитувач, який зчитує код. Цей код передається в комп'ютер, який виступає в якості контролера в системі. Комп'ютер в свою чергу перевіряє код на правильність та в залежності від результату верифікації коду приймається рішення про надання доступу.

Така система не є автономною, тому що при банальному відключенні живлення вона припинить своє функціонування. Але навіть при такій мінімальній кількості компонентів можна організувати додаткові функції як звітність, «антипасбек», резервування подій.

Розглянемо реалізацію автономної СКУД з однією точкою доступу через турнікет. Система складається з турнікету, зчитувача, комп'ютера, контролера.

При вході в приміщення на панелі турнікету чи біля нього встановлюється зчитувач, який передає отриманий код в контролер системи. В системі кожному коді поставлена в відповідність інформація про права доступу. На основі цієї інформації та ситуації (вхід, вихід, вхід в певну зону доступу і т.д.) система приймає рішення : контролер відкриває чи блокує прохід через турнікет.

Інформація про події в системі зберігається та може бути використана в подальшому для отримання звітів. Контролер при цьому може забезпечувати автономну роботу системи, навіть при збоях електроживлення чи інших причинах втрати зв'язку з головним комп'ютером. Завдяки вбудованій енергонезалежній пам'яті контролер може формувати базу подій, зберігати коди ідентифікаторів та назначені для них права. Це дозволяє функціонувати системі без підключення до комп'ютера певний час.

Комп'ютер в даному випадку є комп'ютером для моніторингу системи та безпосередньо комп'ютером прохідної.

1.5 Формування вимог до системи контролю доступу

Як уже зазначалось вище, системи контролю та управління доступом призначені для забезпечення санкціонованого входу в приміщення та в зони обмеженого доступу і вихід з них шляхом ідентифікації особи по комбінації різних ознак, а також для попередження несанкціонованого проходу в приміщення та в зони обмеженого доступу.

Визначивши в попередніх розділах особливості СКУД та принципи реалізації систем контролю доступу, провівши їх класифікацію можемо сформулювати основні вимоги до СКУД.

Для того щоб сформулювати вимоги до системи в цілому, слід сформулювати їх для основних елементів системи. В пункті 1.2 «Аналіз особливостей СКУД» ми визначили, що система складається з таких елементів як ідентифікатор, пристрій ідентифікації (зчитувач), контролер та виконавчий пристрій (рисунок 1.2.1). Сформулюємо вимоги до кожного компонента.

Ідентифікатори повинні забезпечувати збереження ідентифікаційної ознаки протягом всього терміну експлуатації. Конструкція, зовнішній вигляд та написи на ідентифікаторі та зчитувачу не повинні призводити до розкриття коду. Ідентифікатори повинні бути захищені від підробки та копіювання, а також від впливу зовнішніх факторів та вандалізму.

Що стосується зчитувачів, то вони повинні забезпечувати надійне зчитування коду з ідентифікаторів, перетворення його в електричний сигнал і передачу на контролер. Зчитувачі повинні бути захищені від маніпулювання шляхом перебору, підбору коду і радіочастотного сканування.

При введенні неправильного коду повинен блокуватися введення на час, величина якого задається в паспортах на конкретні види зчитувачів. Час блокування потрібно підібрати таким чином, щоб забезпечити задану пропускну спроможність при обмеженні числа спроб підбору. Так, наприклад, при трьох спробах введення неправильного коду повинне видаватися тривожне повідомлення. Для систем, що працюють в автономному режимі тривожне повідомлення передається на звуковий/світловий оповищувач, а для систем, що працюють в мережевому режимі – на центральний пульт. Тривожне повідомлення має видаватися також при будь-якому акті вандалізму. Якщо зчитувач виступає також і контролером (наприклад, в разі автономної системи), то він повинен порівнювати введений (зчитаний) ідентифікатор з тим, що зберігається в базі даних чи пам'яті.

Також потрібно мати на увазі спосіб зчитування коду. Так клавіатура забезпечує низький рівень безпеки, магнітні картки – середній, електронні ключі – високий. Найкращими з точки зору безпеки є біометричні зчитувачі.

В залежності від виду системи, контролери бувають двох видів – автономні та мережеві. Контролери, що працюють в автономному режимі, повинні забезпечувати прийом інформації від зчитувачів, обробку інформації та формування сигналів керування для виконавчих пристроїв. Щодо мережевих контролерів, то вони повинні забезпечувати:

- обмін інформацією по лінії зв'язку між контролерами і керуючим комп'ютером або провідним контролером;
- збереження пам'яті, установок, кодів ідентифікаторів при обриві зв'язку з керуючим комп'ютером, відключенні живлення і при переході на резервне живлення;
- контроль ліній зв'язку між окремими контролерами і між контролерами і керуючим комп'ютером.

Невід'ємною частиною контролеру є його програмне забезпечення, до якого також потрібно сформулювати вимоги.

Програмне забезпечення має бути захищене від навмисних впливів з метою зміни установок в системі. Тобто воно повинне мати захист від несанкціонованого доступу за допомогою паролів. Кількість рівнів доступу по паролях повинно бути не менше 3.

Рекомендовані рівні доступу за типом користувачів:

- перший ("адміністрація") - доступ до всіх функцій контролю та доступу;
- другий ("оператор") - доступ тільки до функцій поточного контролю;
- третій ("системний") - доступ до функцій конфігурації програмного забезпечення, без доступу до функцій, що забезпечують управління пристроїв виконавчих.

При введенні пароля на екрані дисплея не повинні відображатися введені символи.

Також програмне забезпечення повинне бути стійким до відключення комп'ютера. Після перезапуску програми повинна зберігатися працездатність системи та збереження встановлених параметрів.

Також потрібно звернути увагу на поведінку програми при випадковому натисканні клавіш. Такі дії не повинні призводити до відкриття загороджувачів пристроїв чи зміни діючих кодів доступу.

Щодо виконавчих пристроїв, то вони повинні відповідати наступним вимогам.

Виконавчі пристрої повинні забезпечувати відкриття/закриття запірної механізми або пристрою загородження при подачі керуючого сигналу від контролера, а також необхідну пропускну спроможність для даного об'єкта.

Умисне пошкодження зовнішніх електричних з'єднувальних ланцюгів не повинно призводити до відкривання пристрої загородження.

У разі зникнення електроживлення в виконавчих пристроях повинна передбачатися можливість живлення від резервного джерела струму, а також

механічне аварійне відкриття пристроїв загородження. Аварійна система відкриття повинна бути захищена від можливості використання її для несанкціонованого проникнення.

Виконавчі пристрої повинні бути захищені від впливу шкідливих зовнішніх факторів (електромагнітних полів, статичної електрики, нестабільної напруги живлення, пилу, вологості, температури і т.п.) і вандалізму.

Отже, підведемо висновки. Враховуючи все вищесказане, СКУД повинна відповідати наступним вимогам:

- відкриття виконавчого пристрою при зчитування ідентифікатору, доступ по якому дозволений в дану зону доступу в заданий часовий інтервал;
- захист від несанкціонованого доступу до програмних засобів контролеру для зміни/додавання/видалення ідентифікаторів, даних по подіям системи чи зміни налаштувань програми;
- збереження налаштувань та бази даних ідентифікаторів;
- при відключенні електроживлення;
- ручне, напіваавтоматичне або автоматичне відкривання виконавчих пристроїв для проходу при аварійних ситуаціях, пожежах;
- автоматичне закриття виконавчого пристрою за відсутності факту проходу через певний час після зчитування дозволеного ідентифікатору;
- видача сигналу тривоги (або блокування виконавчого пристрою на певний час) при спробах підбору ідентифікаційних ознак (коду);
- реєстрація та протоколювання всіх подій;
- автономна робота зчитувача в кожній точці доступу при відмові зв'язку з контролером.

1.6 Постановка задачі

Основною метою даної роботи є дослідження інформаційного комплексу ідентифікації осіб з селекцією доступу на основі використання розподілених систем.

Результатом розробки має бути сукупність описаних певним чином моделей і алгоритмів, що на достатньому рівні дають уявлення про логічну та фізичну структуру всіх компонентів системи.

Комплекс має забезпечувати реалізацію наступних етапів:

- заведення всіх необхідних параметрів системи (споруди, зони доступу, точки проходу);
- реєстрація ідентифікаторів відвідувачів, які будуть представляти собою штрих-коди проданих білетів та абонементів;
- прийом системою ідентифікаторів від відвідувачів;
- верифікація отриманого ідентифікатору в системі;
- формування результату.

Для досягнення поставленої мети необхідно:

- Дослідити та проаналізувати існуючі системи контролю доступу;
- визначити основні особливості системи санкціонованого доступу;
- провести огляд основних методів реалізації системи;

на основі отриманої інформації сформулювати вимоги до системи враховуючі потреби кінцевого користувача системи.

2 РОЗРОБКА ПРОЕКТНИХ РІШЕНЬ СИСТЕМИ КОНТРОЛЮ ДОСТУПУ

2.1 Проект інформаційного забезпечення

2.1.1 Розробка діаграми варіантів використання

Розробка діаграми варіантів використання переслідує мету:

1) Визначити загальні межі і контекст модельованої наочної області на початкових етапах проектування системи.

2) Сформулювати загальні вимоги до функціональної поведінки проектованої системи.

3) Розробити початкову концептуальну модель системи для її подальшої деталізації у формі логічних і фізичних моделей.

4) Підготувати початкову документацію для взаємодії розробників системи з її замовниками і користувачами.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді множини сутностей або акторів, що взаємодіють з системою за допомогою варіантів використання. При цьому актором або дійовою особою називається будь-яка сутність, що взаємодіє з системою ззовні. У свою чергу, варіант використання служить для опису сервісів, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, ініційованих системою при діалозі з актором. При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів з системою.

Основними поняттями діаграм варіантів використання є: діюча особа, варіант використання, зв'язок. Діюча особа (актор) – зовнішня стосовно розроблювального програмного забезпечення сутність, що взаємодіє з ним з метою одержання або надання якої-небудь інформації. Діючими особами можуть бути користувачі, інше програмне забезпечення або які-небудь технічні засоби, взаємодіючі з розроблювальним програмним забезпеченням. Варіант використання – деяка очевидна для діючої особи процедура, що вирішує його конкретне завдання. Всі варіанти використання, так чи інакше, пов'язані з функціональними вимогами до розроблювальної системи й можуть сильно

відрізнятися за обсягом виконуваної роботи. Зв'язок – взаємодія діючих осіб і відповідних варіантів використання. Діаграма використання для проходу до об'єкту зображена (рисунок 2.1)

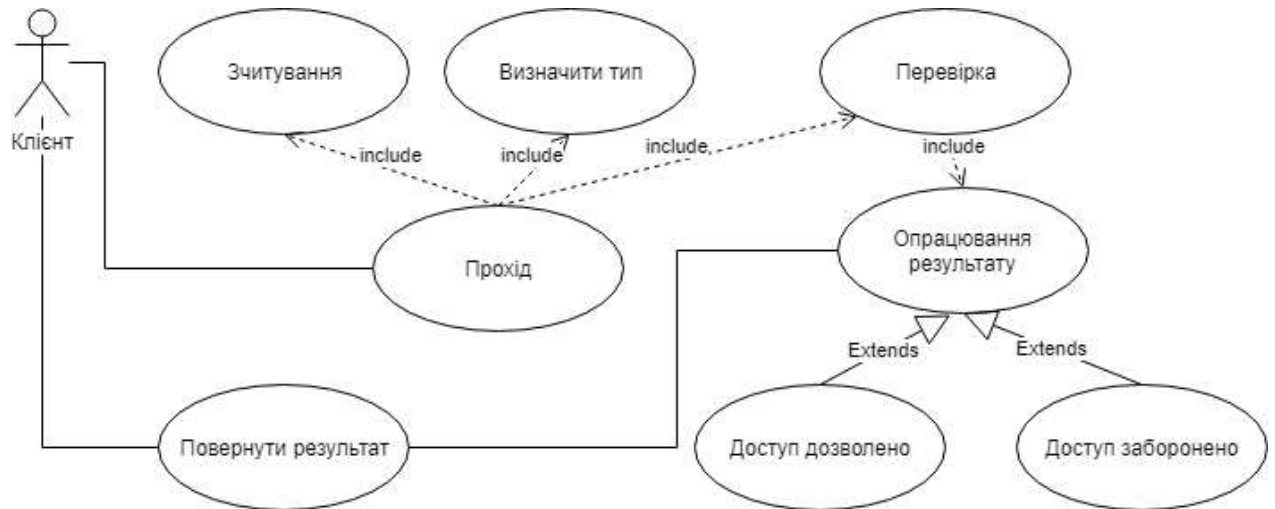


Рисунок 2.1- Діаграма використань події «Прохід на об'єкт»

2.1.2 Розробка діаграми послідовності

На діаграмі послідовності зображуються тільки ті об'єкти, які безпосередньо беруть участь у взаємодії. Ключовим моментом для діаграм послідовності є динаміка взаємодії об'єктів у часі.

У UML діаграма послідовності має як би два виміри. Перше зліва направо у вигляді вертикальних ліній, кожна з яких зображає лінію життя окремого об'єкта, що бере участь у взаємодії. Крайнім зліва на діаграмі зображується об'єкт, який є ініціатором взаємодії. Правіше зображується інший об'єкт, який безпосередньо взаємодіє з першим. Таким чином, всі об'єкти на діаграмі послідовності утворюють деякий порядок, який визначається черговістю або ступенем активності об'єктів при взаємодії один з одним.

Графічно кожен об'єкт зображується прямокутником і розташовується у верхній частині своєї лінії життя. В середині прямокутника записуються ім'я об'єкту і ім'я класу розділені двокрапкою. При цьому вся запис підкреслюється, що є ознакою об'єкта.

Другим виміром діаграми послідовності є вертикальна часова вісь, спрямована зверху вниз. Початкового моменту часу відповідає сама верхня частина діаграми. Взаємодії об'єктів реалізуються за допомогою повідомлень, які надсилаються одними об'єктами іншим. Повідомлення зображуються у вигляді горизонтальних стрілок з ім'ям повідомлення, а їх порядок визначається часом виникнення. Тобто, повідомлення, розташовані на діаграмі послідовності вище, ініціюються раніше тих, що розташовані нижче. Масштаб на осі часу не вказується, оскільки діаграма послідовності моделює лише тимчасову впорядкованість взаємодій типу «раніше-пізніше». Діаграма послідовності процесу верифікації коду зображена (рисунок 2.2)

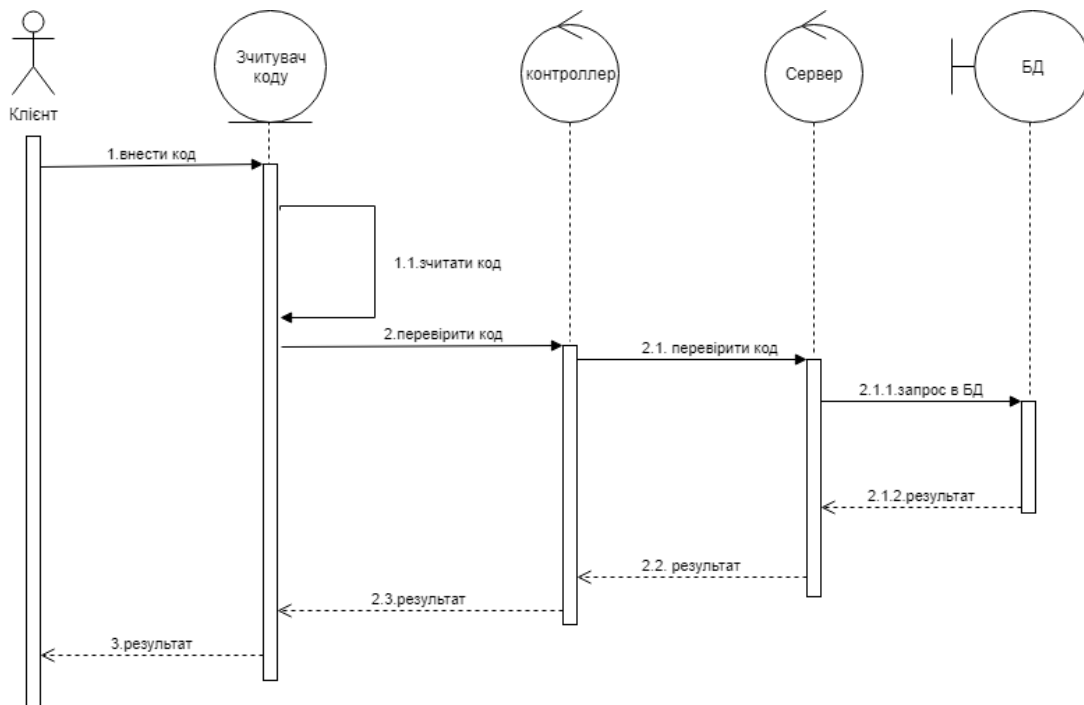


Рисунок 2.2. - Діаграма послідовності процесу верифікації коду

Все відбувається автоматично. Але на випадок якщо сканер штрих-коду якимось чином вийде з ладу і не зможе передати код в систему, передбачена можливість ручного вводу.

2.1.3 Розробка діаграми діяльності

Діаграма діяльності — в UML, візуальне представлення графу діяльностей. Граф діяльностей є різновидом станів скінченного, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.

Дія (англ. *action*) є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів. Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності. Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма діяльності процесу перевірки коду приведена (рисунок 2.3)

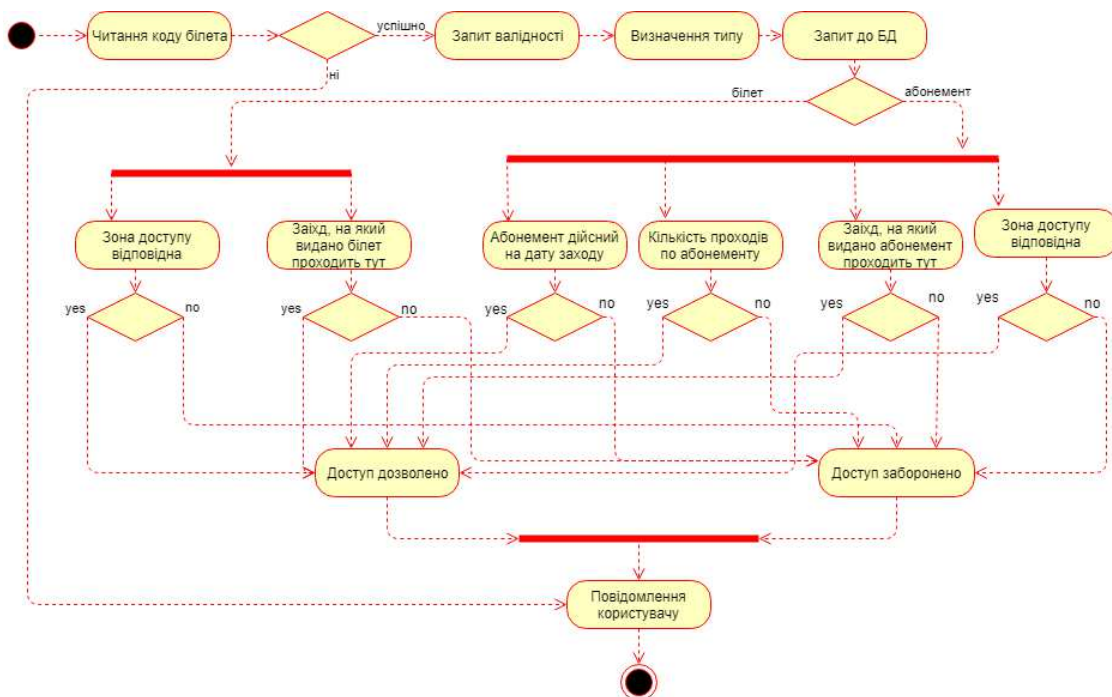


Рисунок 2.3 - Діаграма діяльності процесу перевірки коду

Однією з головних функцій системи контролю доступу є функція «антипасбек» - заборона повторного проходу. Перевірка виконання такої умови забезпечується наступним чином.

Після того як код був успішно зчитаний, відбувається перевірка належності турнікета, через який здійснює прохід особа. Так як зона доступу має зв'язок «один-до-багатьох» з сутністю «Турнікет» з одного боку, та зв'язок «багато-до-багатьох» з сутністю «Сектор» з іншого боку, це можливість по сектору білету знайти відповідну зону доступу. Маючи конкретну зону доступу можна знайти всі її турнікети, і вже серед знайдених турнікетів здійснити пошук конкретного турнікету. Якщо турнікет знайдений, йде наступна перевірка на те, чи був уже зафіксований вхід в дану зону по даному коду. Якщо входу не було – прохід дозволений, в іншому разі – навпаки.

Діаграма активності процесу перевірки коду на повторний прохід зображена (рисунок 2.4)

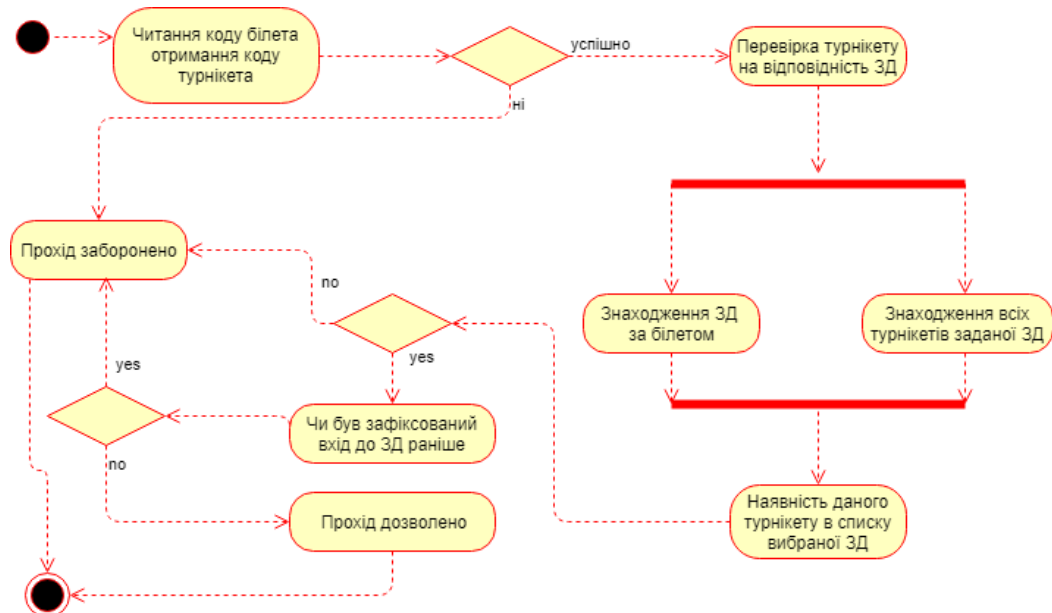


Рисунок 2.4. - Діаграма діяльності процесу перевірки коду на повторний прохід.

2.1.4 Модель даних системи контролю доступу

Будь-яке створення системи починається з визначення об'єктів, які будуть приймати участь у її функціонуванні, з визначення даних та їх взаємозв'язків між собою. Для цього використовують модель даних.

Модель даних – це поняття, яке включає в себе визначення об'єктів системи та їх взаємозв'язок між собою.

Одним з інструментів представлення моделі даних є модель "сутність-зв'язок". Основною перевагою даної моделі є те що вона уніфікована, тобто не залежить від засобів, що будуть її реалізовувати. Модель "сутність-зв'язок" (ER-модель) (англ. Entity-relationship model) ґрунтується на певній важливій семантичній інформації про реальний світ і призначена для логічного представлення даних. Вона визначає значення даних в контексті їх взаємозв'язку з іншими даними. Будь-який фрагмент предметної області може бути представлений як безліч сутностей, між якими існує деяка множина зв'язків. ER-модель - це одна із самих простих візуальних моделей. Вона дозволяє зрозуміти структуру об'єкта «крупними мазками», в загальних рисах. Такий загальний опис структури називається ER-діаграмою або онтологією вибраної предметної області.

Модель даних системи контролю доступу буде складатись з наступних елементів:

- споруда;
- масовий захід;
- сектор;
- білет;
- зона доступу;
- турнікет;
- тип проходу;

- тип коду;
- право;
- роль;
- користувач.

Всі сутності системи можна умовно поділити на чотири групи: системні, сутності фізичного розташування та зонального розподілу, сутності користувача та сутності, які відносяться до масового заходу, що проводиться на стадіоні. Опишемо більш детально кожний об'єкт.

Споруда. Дана сутність слугує для представлення об'єкту, на якому буде встановлена система управління доступом. Даний елемент моделі даних не прив'язаний до жодної з сутностей системи. В конкретній програмі відображає стадіон.

Масовий захід - презентує собою захід, який буде проводитись на конкретній споруді, що визначається по зовнішньому ключу, який зв'язує масовий захід та споруду за допомогою зв'язку «багато-до-одного». Для даного випадку масовим заходом є футбольний матч. Сутність містить в собі необхідні для системи характеристики події, такі як назва, дата початку та часовий діапазон дозволу пропуску відвідувача на матч – дата початку доступу та дата завершення доступу.

Зона доступу – сутність, що представляє собою певний обмежений простір, в який потрібно надати чи заборонити прохід. Зону доступу складають набір секторів. При чому зв'язуються дані об'єкти за допомогою зв'язку «багато-до-багатьох», що означає можливість визначати під зони. Сутність залежить від елементу Споруда, залежність визначається зв'язком «багато-до-одного», тобто одна Споруда може вміщати багато зон доступу.

Турнікет – представляє конкретний турнікет чи інший загороджуючий пристрій в системі. Кожен турнікет має своє ім'я та унікальний ідентифікатор, за яким програмний комплекс визначає чи належить він відповідній зоні доступу, в яку намагається пройти користувач. На кожну зону доступу може бути встановлено декілька турнікетів, наприклад, для регулювання напряму

переміщення відвідувачів на стадіону. Тому турнікет також володіє такою характеристикою як тип проходу.

Тип проходу. Системна сутність, яка використовується для визначення напрямку проходу в зоні доступу та для типу проходу через турнікет. Має три константні значення: вхід, вихід, вхід/вихід.

Сектор – використовується для зберігання інформації про фізичне місце людини на матчі та для логічного поділу стадіону на зони доступу. Має зв'язок «багато-до-багатьох» з зонами доступу та зв'язок «один-до-багатьох» з сутністю Білет.

Білет – сутність для представлення білету в системі. Зберігає в собі таку інформацію як фізичне розміщення місця відвідувача – сектор, ряд, місце, інформацію про захід, на який він був куплений та тип коду. Має зв'язок «багато-до-одного» з масовим заходом, що визначає очевидне поняття: для одного матчу може бути багато білетів.

Тип коду – системна сутність, яка дозволяє визначити особливості коду, що пришов в систему. Має три статичні значення: білет, абонемент та службовий. Останній перетворює білет в картку пропуску людини на стадіон, не перевіряючи параметрів матчу. Мається на увазі, що людина є адміністратором чи іншою визначеною в системі особою.

Користувач – зберігає в собі дані про кожного користувача системи, такі як: ПІБ, логін, пароль, локалізаційні побажання користувача, роль. Через зв'язок «багато-до багатьох» з сутністю Тип коду визначає чи має право даний користувач проходити на відповідні коду зони доступу.

Роль – ідентифікатор певної обмеженої множини прав, що задана конкретному користувачу. Має зв'язок «один-до-багатьох» з сутністю Користувач, тобто роль «Адміністратор» наприклад, може мати декілька осіб. Також має зв'язок «багато-до-багатьох» з сутністю Право, це означає, що декілька ролей можуть мати однакові елементи прав. Наприклад «Адміністратор» та «Менеджер» можуть переглядати зони доступу в плагіні «Адміністратор» програмного комплексу.

Відношення між сутностями системи представленні на діаграмі моделі даних (рисунок 2.5).

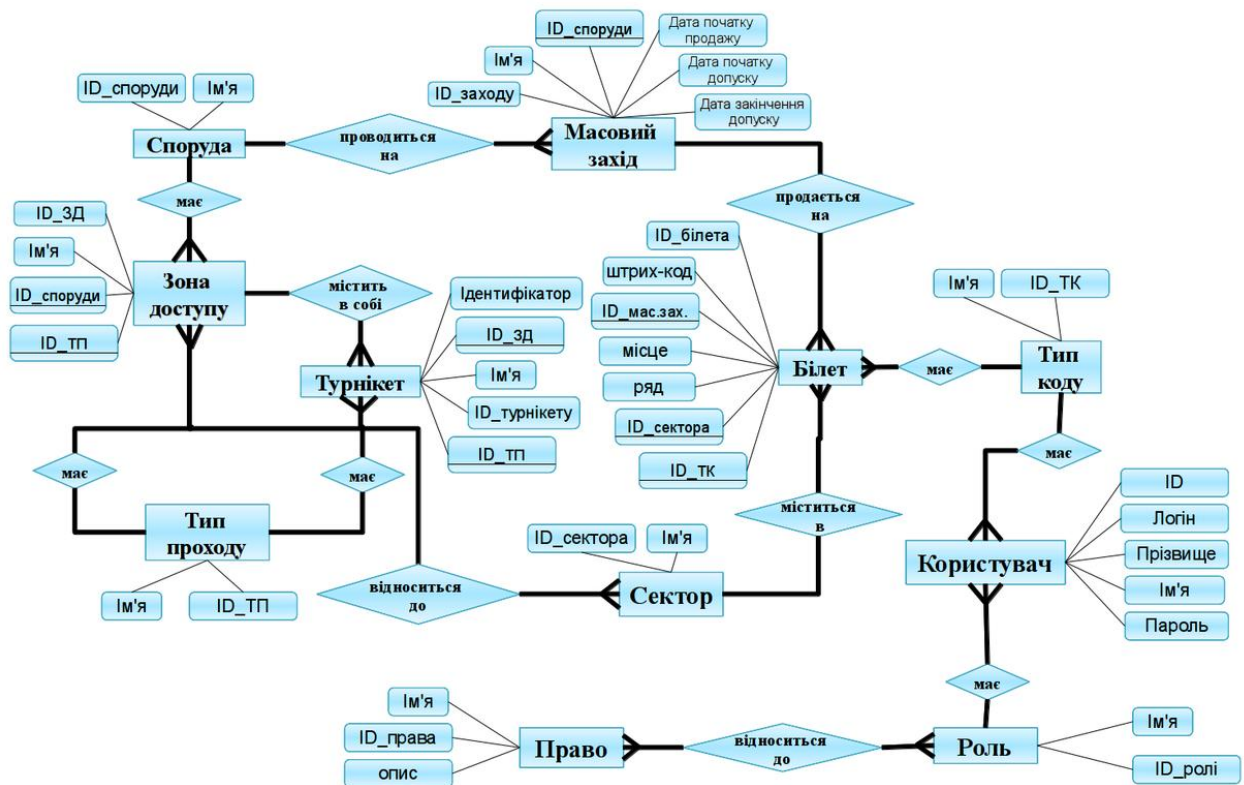


Рисунок 2.5. - Діаграма "Сутність-зв'язок"

2.1.5 Фізична модель

Фізична модель даних – це модель, що описує спосіб зберігання даних в комп'ютері, представляючи інформацію про структуру записів, їх впорядкованість та шляхи доступу. Тобто, дана модель описує реалізацію об'єктів логічної моделі на рівні об'єктів конкретної бази даних. В фізичній моделі у вигляді опису міститься інформація про всі об'єкти БД. В описі об'єктів БД визначається фізичний формат даних, реалізуються обмеження предметної області, бізнес-логіка, що автоматизується, рівні доступу користувачів. Опис створюється на мові

визначення даних обраної СУБД, до якої входять оператори, що дозволяють створити або видалити об'єкт БД, модифікувати його структуру і т. ін.

Як уже зазначалось раніше, в якості сервера реляційної бази даних був використаний PostgreSQL версії 9.1. Так як використана база даних є реляційною, це дозволяє контролювати цілісність на рівні бази за допомогою цілісності посилань та цілісності сутностей.

Об'єкту або сутності реального світу в реляційних БД відповідають кортежі відношень. Конкретно вимога полягає в тому, що будь-який кортеж будь-якого відношення відрізняється від будь-якого іншого кортежу цього відношення, тобто іншими словами, будь-яке відношення має володіти первинним ключем.

Для дотримання цілісності сутності досить гарантувати відсутність в будь-якому відношенні кортежів з одним і тим же значенням первинного ключа.

Стосовно посилань маємо наступні особливості. При оновленні посилається відношень (вставці нових кортежів або модифікації значення зовнішнього ключа в існуючих кортежах) достатньо стежити за тим, щоб не з'являлися некоректні значення зовнішнього ключа. Але як бути при видаленні кортежу з відношень, на якого веде посилання?

Тут існують три підходи, кожен з яких підтримує цілісність за посиланнями.

- Перший підхід полягає в тому, що забороняється проводити видалення кортежу, на який існують посилання (тобто спочатку потрібно або видалити посилання на кортеж з відповідним ключем, або відповідним чином змінити значення їх зовнішнього ключа).

- При другому підході при видаленні кортежу, на який є посилання, всі значення зовнішнього ключа автоматично стають невизначеними.

- Третій підхід (каскадне видалення) полягає в тому, що при видаленні кортежу з відношень, на який веде посилання, з відношення, яке посилається на кортеж, що видаляється, автоматично видаляються всі кортежі, які мають посилання.

Фізична структура бази даних, з визначеними зовнішніми ключами та типами даних зображена (рисунок 2.6)

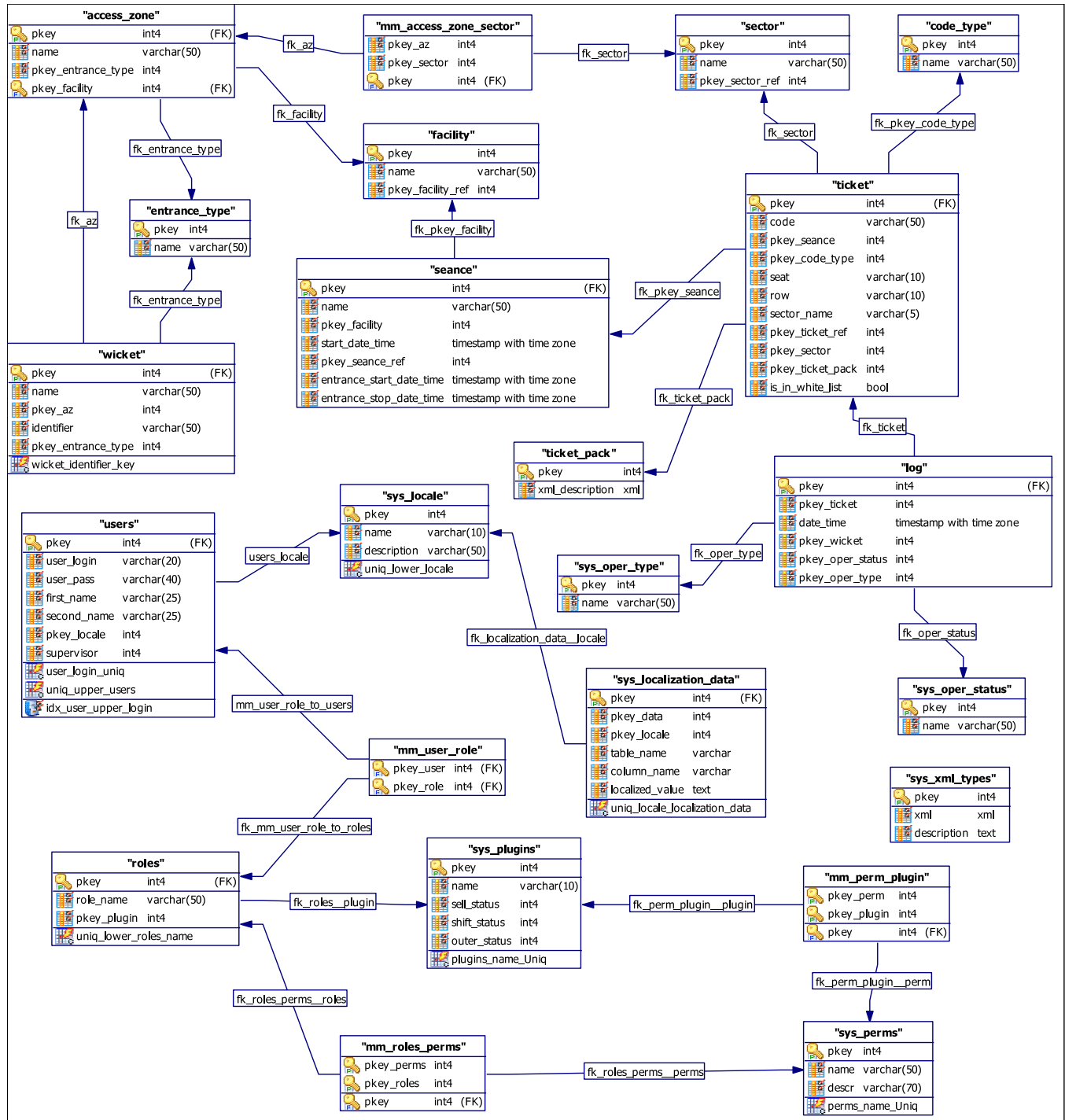


Рисунок 2.6. - Фізична структура бази даних

2.2 Проект технічного забезпечення

Вся система складається з двох компонентів - компоненти взаємодії з користувачами та взаємодії зі сховищами даних.

WebServer. Компонент, що з одного боку взаємодіє зі сховищами даних, а з іншого - з клієнтами, що робить його універсальним посередником між кінцевим користувачем даними та місцем їх зберігання. З реляційної базою даних вся комунікація відбувається через інтерфейс Java DataBase Connectivity (JDBC) – промисловий, платформно-незалежний стандарт, що у видку потреби забезпечить легкий перехід на будь який інший сервер реляційних баз даних. З клієнтською частиною зв'язок відбувається за допомогою HTTP/HTTPS повідомлень, що засновані на передачі даних у вигляді XML/JSON структури.

Завдяки тому, що компонент використовує систему веб-сервісів, то його легко можна буде використовувати для інших платформ. Так однією з вимог було надання можливості використання мережевих контролерів, які в собі вміщують і зчитувач.

Для того щоб, використовувати даний сервер для інших клієнтів потрібно лише конфігурувати контролер на під'єднання до даного серверу та написати адаптер, який би конвертував відповіді серверу в необхідні контейнери чи взагалі примітивні типи даних, що задовольнили б клієнта.

Приведені вище операції були зроблені для пристрою CipherLab 9300. Даний пристрій представляє собою термінал збору даних під керуванням операційної системи Windows CE 6.0 з вбудованим скануючим пристроєм.

2.2.1 Розробка діаграми класів

Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини. WebServer складається з двох основних модулів. Модуль «Admin» за допомогою якого додаються та редагуються дані в СУБД. Другий модуль «Validation» - це основний модуль який перевіряє квитки.

Діаграма класів компонента «WebServer» зображена (рисунок 2.7)

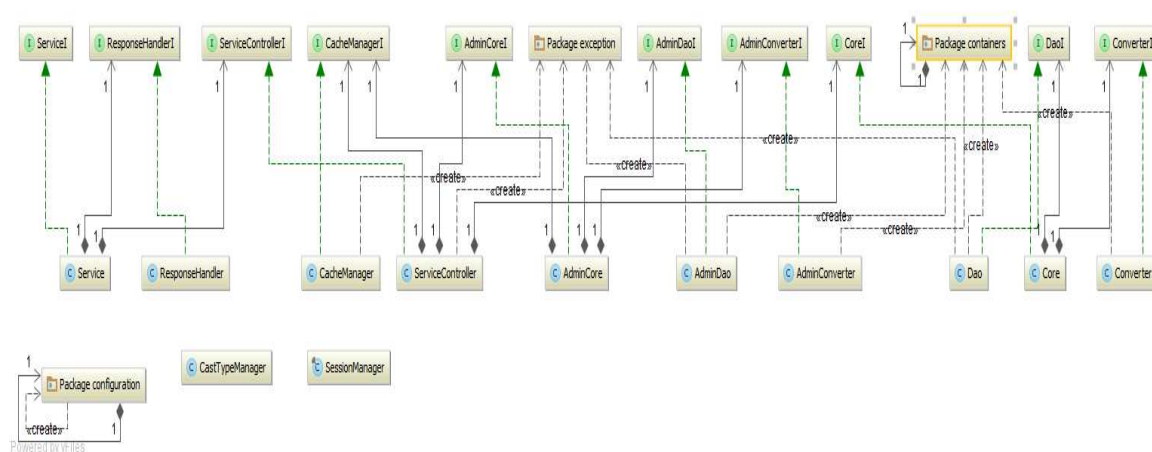


Рисунок 2.7. - Діаграма класів компонента «WebServer»

2.2.2 Розробка діаграми пакетів

Діаграма пакетів уніфікованої мови моделювання (UML) відображають залежності між пакетами, складовими модель.

На додаток до стандартних відносинам залежностей у UML є два спеціальних виду залежностей, визначених між пакетами:

- Імпортування пакету
- Злиття пакету

Імпортування пакету - це відношення між імпортують простором імен і пакетом, що вказує на те, що імпортує простір імен додає імена членів пакету в їх власний простір імен. Типово, Непомічені залежність між двома пакетами інтерпретується як відношення «імпорт пакету» ..

Злиття пакета - спрямоване відношення між двома пакетами, яке вказує, що вміст двох пакетів має бути об'єднано. Це дуже схоже на «Узагальнення» в тому сенсі, що вихідний елемент як би додає характеристики цільового елемента до своїх власних характеристикам, в результаті чого виходить елемент, який поєднує в собі характеристики обох елементів.

Для розуміння того, як упорядковані та згруповані класи в системі, була розроблена діаграма пакетів (рисунок 2.8).

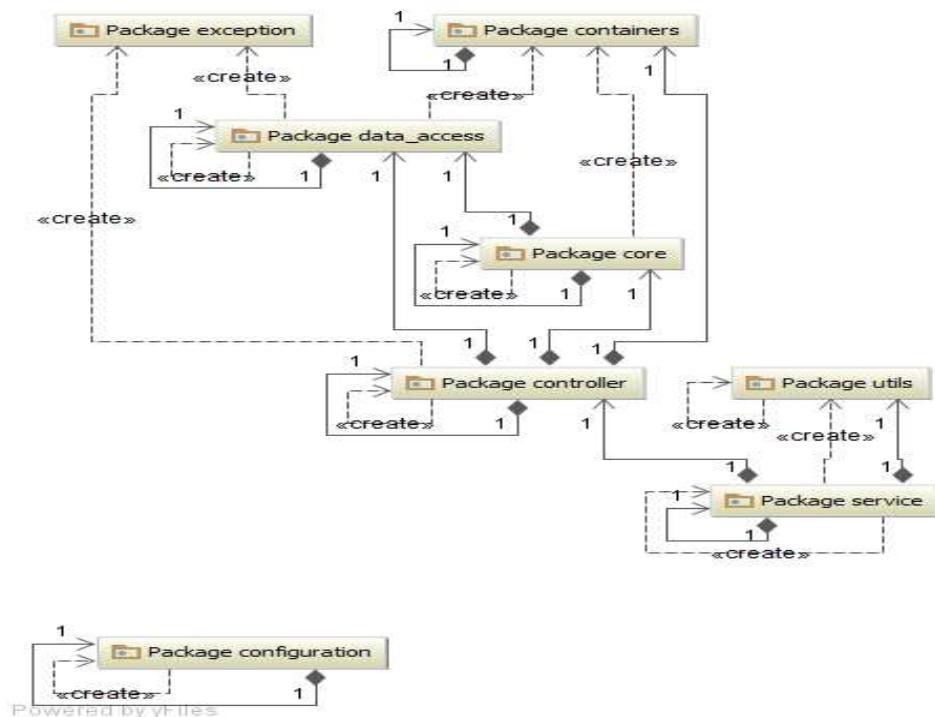


Рисунок 2.8. - Діаграма пакетів

Service – Пакет, який містить в собі всі класи за допомогою, яких приймаються дані від клієнта. Класи в цьому пакеті можуть приймати дані у любых доступних форматів та протоколів.

Core – Пакет, який містить основні класи які управляють системою. Їх основна ціль прийняти дані та в залежності від них викликати той чи інший модуль.

DataAccess – Пакет, який містить класи для взаємозв'язку с базою даних. Формування запитів, та обробка результатів від баз даних.

2.2.3 Діаграма розгортання

Діаграма розгортання призначена для візуалізації елементів і компонентів програми, існуючих лише на етапі її виконання (Runtime). При цьому представляються тільки компоненти-екземпляри програми, що є виконуваними файлами або динамічними бібліотеками. Ті компоненти, які не використовуються на етапі виконання, на діаграмі розгортання не показуються.

Діаграма розгортання зображена (рисунок 2.9)

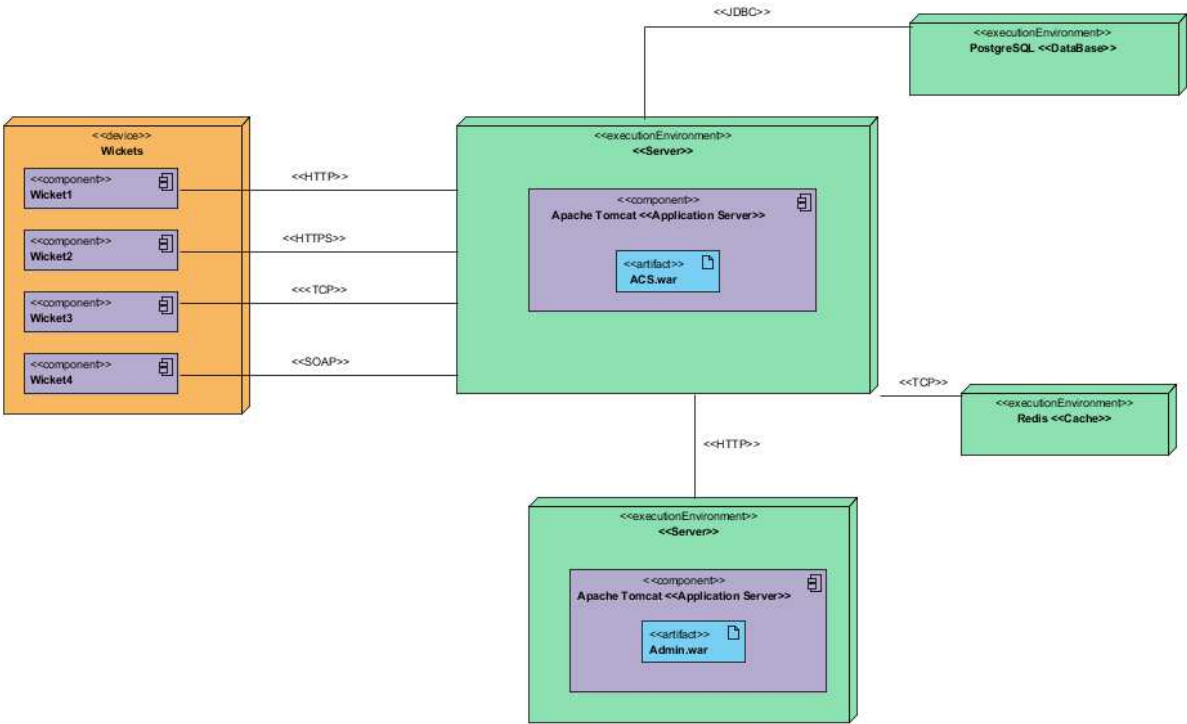


Рисунок 2.9. – Діаграма розгортання

2.3 Проект програмного забезпечення

2.3.1 Мова програмування

Сьогодні створення програмного забезпечення являє собою надзвичайно важке заняття. Труднощі пов'язані з різноманітністю архітектури машин, операційних систем, графічних оболонок і т. д.

Сучасні програмні продукти повинні бути безпечні, високопродуктивні, працювати в розподіленому середовищі, бути нейтральними до архітектури. Всі ці чинники привели до необхідності нового погляду на сам процес створення і розподілу програмних додатків на безлічі комп'ютерів різної архітектури. Вимоги до платформи незалежності примусили відмовитися від традиційного способу створення бінарних файлів, що містять машинні коди і, отже, прив'язані до певної платформи.

Однією з основних вимог до програмного комплексу було саме платформи незалежність, тому що невідомо яку операційну систему може використовувати кінцевий користувач.

Створена компанією Sun Microsystems мова програмування Java задовольняє всі ці вимогам. Java - об'єктно-орієнтована мова програмування, зручна і надійна в експлуатації завдяки таким своїм перевагам, як багатозадачність, підтримка протоколів Internet і кросплатформенність. Java - це мова, що інтерпретується, і кожна Java-програма компілюється для гіпотетичної машини - Віртуальної Машини Java (Java Virtual Machine). Результатом такої компіляції є байт-код Java, який в свою чергу може виконуватися на будь-якій операційній системі за умови наявності там системи часу виконання Java (Java Runtime Executer), яка інтерпретує байт-код в реальний машинний код конкретної системи. JRE складається з віртуальної машини Java (JVM), базових класів платформи Java і допоміжних бібліотек.

Ще однією з важливих переваг технології Java – є її безпека. Вона визначається тим, що Java-машина повністю контролює, як виконується програма. Якщо програма поводить ся несподіваним чином, наприклад, починає намагатися отримати доступ до даних користувача, її робота тут же переривається. Таким чином, користувачі, завантажуючи на свій комп'ютер java-програми, можуть не хвилюватися, що це приведе до зараження вірусами.

Однак, така універсальність даної технології породжує недолік - вимогливість до ресурсів комп'ютера. Так як Java-програми не містять машинного коду і при їх запуску включається в роботу система часу виконання Java, їх продуктивність дещо нижче, ніж у звичайних програм, написаних, наприклад, мовою програмування C++. Проте, даний недолік стає з часом все менш відчутний, внаслідок зростання обчислювальної потужності комп'ютерних систем.

Мова Java є об'єктно-орієнтованою і поставляється з досить об'ємною бібліотекою класів. Бібліотеки класів Java значно спрощують розробку програмних додатків, надаючи в розпорядження програміста потужні засоби вирішення поширених завдань. Тому програміст може більше уваги приділити розв'язання прикладних задач, а не таких, як, наприклад, організація динамічних масивів, взаємодія з операційною системою або реалізація елементів користувацького інтерфейсу.

2.3.2 База даних

PostgreSQL - це об'єктно-реляційна система управління базами даних (СУБД), найбільш розвинута з відкритих СУБД у світі, яка є реальною альтернативою комерційних баз даних.

Може виникнути питання - чому саме PostgreSQL? Найкраще відповісти на нього, порівнявши цей продукт з іншим, не менш популярним і, мабуть, більш відомим - MySQL.

Перш за все, слід відзначити відмінності в організації даних. Для більшості типів баз даних (БД) в тому числі і для MySQL кожна таблиця заноситься у

власний файл, то PostgreSQL організовує єдину файлову структуру, в якій окремі файли не відповідають безпосередньо таблицями чи іншим об'єктам бази даних. Тобто в MySQL можливо створити резервну копію лише частини БД, зберігши відповідні файли. PostgreSQL має більш широкі можливості, як для обробки даних, так і для адміністрування, але це, в свою чергу, дещо підвищує складність роботи з цією СУБД.

На відміну від MySQL, в PostgreSQL присутні транзакції – неподільна послідовність дій маніпулювання даних, результати якої або повністю відображаються в БД, або повністю відсутні. Транзакції забезпечують цілісність БД. Для забезпечення контролю цілісності кожна транзакція повинна починатися при цілісному стані БД і повинна зберегти цей стан цілісним після свого завершення. Якщо оператори, об'єднані в транзакцію, виконуються, то відбувається нормальне завершення транзакції, і БД переходить в оновлений (цілісний) стан. Якщо ж відбувається збій при виконанні транзакції, то відбувається так званий відкат до вихідного стану БД.

Ще однією перевагою PostgreSQL є підтримка роботи з представленнями. Представленням називається SQL-запит на вибірку, якому привласнюється ім'я і який зберігається в базі даних. Представлення дозволяє користувачеві побачити результати збереженого запиту, тому що він може звертатися до них як до звичайних таблиць.

Представлення використовуються з декількох причин:

- дозволяють приховати від користувачів структуру бази даних;
- з їх допомогою можна обмежити доступ до даних, дозволяючи користувачам бачити тільки деякі з рядків і стовпців таблиці ;
- вони спрощують доступ до бази даних, показуючи кожному користувачеві структуру збережених даних в найбільш зручному для нього вигляді.

PostgreSQL підтримує роботу с функціями та типами даних користувача. Функції користувача - це блоки коду SQL, призначені для повторного використання самими різними користувачами: збереженими процедурами,

додатками і навіть іншими функціями. Це процедури, які можуть приймати параметри і повертати значення. Функції користувача можуть повертати скалярні значення (наприклад, єдине значення) або результуючий набір даних (що містить рядки і стовпці даних).

Таким чином, можна сказати, що місце PostgreSQL в тих розробках, які вимагають високого ступеня надійності зберігання інформації, потребують підвищеної перевірки всіх змін, мають необхідність в автоматичному коректуванні великого числа даних при зміні інформації в одній з таблиць, а також завданнях, де потрібно можливість розробки нетривіальних рішень, використання нестандартних операторів і т.д.

2.3.3 Аспектно-орієнтоване програмування (АОП)

АОП - парадигма програмування, в основі якої лежить ідея виділення наскрізної функціональності в окремі сутності - аспекти (aspects). Наскрізною називають функціональність, розосереджену по різних частинах програми. В якості прикладів наскрізної функціональності можна привести завдання логування, трасування, обробки виключень, перевірки прав доступу. АОП може так само використовуватися для вирішення завдань захисту і багатопоточності.

Основні поняття АОП:

- Аспект (англ. aspect) - модуль або клас, який реалізує наскрізну функціональність. Аспект змінює поведінку решти коду, застосовуючи пораду в точках з'єднання, визначених деякими зрізом.

- Порада (англ. advice) - спосіб оформлення коду, який повинен бути викликаний з точки з'єднання. Порада може бути виконана до, після або замість точки з'єднання.

- Точка з'єднання (англ. join point) - точка в програмі, де слід застосувати пораду. Багато реалізації АОП дозволяють використовувати виклики методів і звернення до полів об'єкта в якості точок з'єднання.

- Зріз (англ. pointcut) - набір точок з'єднання. Зріз визначає, чи підходить дана

точка з'єднання до даної поради. Найзручніші реалізації АОП використовують для визначення зрізів синтаксис основної мови (наприклад, в AspectJ застосовуються Java-сигнатури) і дозволяють їх повторне використання.

– Впровадження (англ. introduction, введення) - зміна структури класу або зміна ієрархії для додавання функціональності аспекту в чужий код.

2.3.4 Середовища розробки програми

Для розробки програмно-технічного комплексу були обрані наступні програмні продукти.

IntelliJ Idea – це кросплатформене інтегроване середовище розробки програмних додатків (IDE) мовами програмування Java, JavaFX, JavaScript, Kotlin та інших.

Підтримують рефакторинг, виділення синтаксичних конструкцій кольором, автодоповнення конструкцій коду, що набираються програмістом на льоту, безліч визначених шаблонів коду та ін.

PgAdmin III Version 1.16.0 - це один із клієнтів для сервера PostgreSQL. PgAdmin є зручним засобом для адміністрування сервера PostgreSQL та створення баз даних, так як має всі необхідні для цього інструменти і інтуїтивно зрозумілий графічний інтерфейс.

Програма створена для задоволення найрізноманітніших потреб користувачів, від написання простих SQL-запитів до створення складних баз даних. Графічний інтерфейс pgAdmin підтримує всі можливості PostgreSQL та значно полегшує адміністрування. Додаток включає SQL-редактор з підсвічуванням синтаксису, редактор збережених процедур, агент планування завдань SQL/командної оболонки, підтримку механізму реплікації Slony-I та багато іншого. Підключення до сервера можна встановити, використовуючи TCP/IP або Unix-сокети, і застосувати SSL-шифрування для безпеки. При цьому ніякі додаткові драйвери для з'єднання з сервером баз даних не вимагаються.

Visual Paradigm for UML 8.0 Enterprise Edition - це професійний інструмент роботи з UML, який підтримує весь робочий цикл програми - аналіз орієнтований на об'єкти, дизайн, конструкцію, тестування та розробку програмного продукту. Це програма UML моделювання, яка може будувати всі типи діаграм класів, переглядати в зворотному порядку код, генерувати код з діаграм і генерувати документацію. Visual Paradigm for UML здійснює весь цикл розробки проектів: об'єктно-орієнтований аналіз, фіксацію вимог, планування, моделювання, тестування, проектування баз даних і т.п. VP також забезпечує підтримку версій *проекту і одночасну роботи команди користувачів над одним проектом.*

Subversion, SVN - це безкоштовна система керування версіями з відкритим програмним кодом. Subversion дозволяє керувати файлами і каталогами, а також зберігає історію змін в проекті, які здійснені різними користувачами. Це дозволяє відновити ранні версії даних, дає можливість проаналізувати історію всіх змін. Аналогом даної системи є CVS (Control Version System), яка також є безкоштовною системою управління версіями.

На відміну від CVS (Control Version System) дана система має надає ряд можливостей:

- SVN дає можливість відстежити історію змін в файлах, навіть після їх переміщення чи зміни назви;
- SVN в порівнянні з CVS набагато швидше працює;
- атомарна фіксація змін;
- клієнт-серверний протокол системи розроблений таким чином, що дозволяє пересилати тільки різницю між об'єктами, а не весь об'єкт.

3 РЕЗУЛЬТАТИ РОЗРОБКИ

В результаті виконання дипломної роботи метою є дослідження технологій ідентифікації осіб з селекцією доступу на основі використання розподілених систем.

Основними функціями системи контролю доступу є:

- розмежований доступ в приміщення особам різної ієрархії повноважень. Тобто, кожен співробітник чи глядач стадіону може пройти в певну зону тільки відповідно до його повноважень.

- заборона повторного проходу на об'єкт по одному й тому ж ідентифікатору.

- автоматизація контролю доступу на об'єкт, де встановлена система, таким чином зменшення людського фактору.

- віддалене управління об'єктом, який потребує контроль санкціонованого доступу. Оператор може відкрити чи заблокувати будь-який прохід знаходячись вдалині від об'єкту.

- управління пожежними та сигналізаційними системами.

- збір статистики певної зони.

- функція встановлення місцезнаходження особи з певним ідентифікатором.

Мінімальними вимогами до функціональності систем контролю доступу, які потім були узгоджені з замовником для написання результуючого програмного комплексу, є наступні:

- використання специфічного обладнання, такого як сканер штрих-коду та термінал збору даних, в якості структурних елементів системи;

- модульність для можливості організації системи різного виду;

- швидкодія обробки даних (для того щоб не створювалась черга);

- бюджетність системи, що забезпечується набором мінімальних функцій.

Всі вимоги були виконані в повному обсязі. Додатково було реалізовано систему розмежування прав доступу між різними групами користувачів. Таким чином, у керівництва стадіону є можливість назначити відповідальних осіб для виконання тих чи інших функцій.

Програмний комплекс був впроваджений на тестову платформу стадіону «Лівий Берег» для проведення тестування в реальних умовах. Сервер проходить тестування на швидкість обробки даних, від чого залежить швидкодія всієї системи, та тестується поведінка програми при нестандартних ситуаціях типу відключення живлення, обрив зв'язку каналів передачі даних і т.ін.

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ ВПРОВАДЖЕННЯ ПЗ

4.1 Вступ

На даний момент сучасні обчислювальні засоби широко використовуються в повсякденній діяльності людини. Сьогодні обчислювальна техніка дозволяє вирішувати найрізноманітніші задачі: від звільнення людини від ручної праці до розробки складних технічних розрахунків, що не під силу виконати одній людині. Сучасні засоби програмування дозволяють вирішувати задачі, що стояли в нещодавно перед цілими колективами наукових співробітників.

Розвиток нових технологій і ріст застосування обчислювальної техніки привели до широкого застосування ЕОМ для вирішення проблем, пов'язаних з обробкою інформації, її контролем і аналізом. Сьогодні на основі засобів обчислювальної техніки розробляються і впроваджуються різні автоматизовані інформаційні системи (проектування, технологічної підготовки виробництва).

Використання даного програмного продукту приведе до значної економії, що виражається в наступних показниках:

- звільнення касирів
- прискорення пропуску клієнтів
- виявлення шахраїв та крадіїв
- звільнення персоналу який перевіряє квитки
- прискорення обробки інформації;
- економія часу на обробку інформації.

Найбільш важливим моментом для розробника, з економічної точки зору, є процес формування вартості програмного продукту. Очевидно, що вона являє собою дуже специфічний товар з безліччю особливостей.

Створення інформаційної СКУД (системи контролю та управління доступу) вимагає одноразових витрат на її розробку, придбання необхідних

технічних засобів і поточних витрат на функціонування системи. Економія від функціонування СКУД визначається з урахуванням витрат на її експлуатацію. Відношення цієї економії до витрат на створення системи характеризує економічну ефективність капітальних вкладень. Економічні показники визначаються по діючим на момент розрахунку оптовим цінам, тарифам і ставкам заробітної плати.

4.2 Розрахунок витрат на створення та експлуатацію інформаційної системи

Витрати на розробку системи складаються з витрат на зарплату розробника, на амортизацію ЕОМ, на якій виконується розробка, на експлуатацію цієї ЕОМ, на засоби розробки та витрат на матеріали і комплектуючі.

Розробка програмного забезпечення виконується програмістом, місячний оклад якого складає 8000 грн . Додаткова заробітна плата складає 20% від основної. Виходячи з цього, основна і додаткова заробітна плата розроблювача системи 9600 грн/міс , а вартість сучасної ПК складає 15000 грн. При вартості кіловат-години електроенергії рівної 1.42 грн., розраховується вартість розробки програми. Витрати на допоміжні матеріали приведені в табл. 4.1

Таблиця 4.1 - Витрати на допоміжні матеріали

Пункти витрат	Сума грн.
Зчитувач	1000.00

Вартість програми розрахуємо по формулі:

$$C_{пр} = (З_{зп} + З_{сз} + З_{зг} + З_{е}) * T + З_{м} \quad (4.1)$$

де T – тривалість розробки, міс.

З_{зп} - основна і додаткова заробітна плата обслуговуючого персоналу, грн.

Зсз - відрахування на соціальні заходи (38% від основної і додаткової заробітної плати), грн.

Ззг - загальногосподарські витрати (10% від основної заробітної плати), грн.

Зе - витрати на електроенергію.

Зм - витрати на основні і допоміжні матеріали.

Зе при споживанні потужності 0.5 кВт, тривалості роботи на місяць, рівної $22 \cdot 6 = 132$ годин, вартості кіловат-години електроенергії 2.15 грн.,

$$З_e = 132 \cdot 2,15 \cdot 0.5 = 141,9 \text{ грн.}$$

Таблиця 4.2 - Витрати на розробку системи.

Найменування витрат	Одиниця	Кількість
Тривалість розробки	Мес	3
Основна і додаткова заробітна	Грн.	9600
Відрахування на соціальні заходи	Грн.	3 648
Загальногосподарські витрати	Грн.	960
Витрати на допоміжні матеріали	Грн.	1000
Витрати на електроенергію	Грн.	141,9

Відповідно до формули 4.1 вартість програми:

$$С_{пр} = (9600 + 3648 + 960 + 141,9) \cdot 3 + 1000 = 44\,049,70 \text{ грн.}$$

Амортизаційні відрахування на устаткування складають 70% балансової вартості в рік:

$$А_{об} = 15000 \cdot 0,70 = 10\,500 \text{ грн.}$$

Річний обсяг робіт ПК у годинах визначається в такий спосіб:

$$\Phi_M = 264 \cdot T_z \quad (5.2)$$

де T_z - це середнє місячне завантаження устаткування (близько 8 годин)

264 - середня кількість робочих днів у році.

Отже, річний обсяг роботи ПЕОМ складе:

$$\Phi_M = 264 \cdot 8 = 2\,112 \text{ годин}$$

Витрати на електроенергію Z_e при 2112 годинах роботи устаткування в рік складуть

$$Z_e = 2112 * 2,15 * 0,5 = 2270,40 \text{ грн.}$$

Експлуатаційні витрати для ПК за рік складуть:

$$Z_{zp} = 10500 + 2270,4 = 12770,40 \text{ грн.}$$

Отже, у перший рік витрати на створення й експлуатацію програми складуть:

$$Z_{se} = 44\,049,70 + 12770,40 = 56820,10 \text{ грн.}$$

4.4 Економічна ефективність розробки і впровадження системи

Основним показником економічної ефективності функціонування системи є підвищення ефективності керування інформацією у вигляді зниження витрат на керування при одночасному збільшенні швидкості і якості одержання потрібного результату.

Крім багатьох інших негативних ефектів, ручна обробка інформації спричиняє наступні негативні економічні ефекти:

- високі витрати на складування паперових документів (сейфи, шафи, папки й ін.);
- підвищені витрати на канцтовари;
- витрати, пов'язані з роботою виявлення раніше допущених помилок (людський фактор).

До числа основних факторів, що визначають приріст прибутку в зв'язку з упровадженням програми, відносяться:

- підвищення продуктивності праці;
- вивільнення робочого часу.

Крім того, не піддається прямій грошовій оцінці підвищення оперативності керування, якість одержуваних результатів, поліпшення організації праці і т.д.

Обов'язковою умовою визначення економічної ефективності програми є порівнянність усіх показників у часі, за цінами й іншими нормами, використовуваними для визначення показників, за змістом і колом елементів витрат.

Визначимо пряму економічну ефективність, ґрунтуючись на тому, що впровадження програмного продукту вивільняє 2,0 працівника (за експертною оцінкою фахівців підприємства).

Зарплата 2,0 працівника в рік складає:

$$(3000 * 12) * 2 = 72000 \text{ грн.}$$

Річний економічний ефект розраховується по формулі:

$$\mathcal{E}_{\text{год}} = \Delta C_n - E_n \cdot k,$$

де ΔC_n – вивільнені кошти після впровадження програмного продукту (72000 грн.) мінус експлуатаційні витрати (10 943,52 грн.) = 61056,47 грн.

E_n – коефіцієнт ефективності (дорівнює коефіцієнту амортизації (0,70));

k – одноразові витрати на впровадження продукту (56820.1)

$$\mathcal{E}_{\text{год}} = 61056,47 - 32\,150,68 * 0,7 = 38\,551 \text{ (грн.)}$$

Строк окупності системи розраховується по формулі:

$$T = \frac{k}{\Delta C} = \frac{56820.1}{38551} = 1,47 \text{ року}$$

Отже строк окупності системи складає приблизно 18 місяців.

5 ОХОРОНА ПРАЦІ

5.1 Загальні визначення та актуальність питання охорони праці.

Охорона праці є системою правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних та лікувально-профілактичних заходів та засобів, спрямованих на збереження здоров'я і працездатності людей в процесі праці.

Як відомо, цілком безпечних і нешкідливих виробництв не існує, тому основною задачею охорони праці є зведення до мінімуму імовірність поразки чи захворювання працюючого з одночасним забезпеченням комфорту при максимальній продуктивності праці. Основний принцип у галузі охорони праці – це пріоритет життя і здоров'я працівника по відношенню до результатів виробничої діяльності. При цьому роботодавці зобов'язані забезпечувати здорові і безпечні умови праці для своїх працівників, вживати всіх заходів щодо запобігання ризиків на робочих місцях, навчати працівників правилам і нормам охорони праці і виконувати інші вимоги законодавства.

Турбота про охорону праці є справою державної важливості. 14 жовтня 1992 року був прийнятий Закон України "Про охорону праці", у якому закріплені найважливіші положення в області охорони праці, і який поширюється на всіх суб'єктів підприємницької діяльності незалежно від форм власності.

Ст.4 Закону "Про охорону праці" формулює принципи державної політики у сфері охорони праці. Ці принципи в силу вказаного їх визначення, слід кваліфікувати і як принципи інституту охорони праці, які підлягають застосуванню при відсутності відповідних конкретних правових норм, і як такі, що визначають напрямки подальшого розвитку інституту охорони праці, і обов'язки держави, які вона взяла на себе і має додержувати при здійсненні своїй політики у майбутньому. З урахуванням цього принципи, зазначені ст. 4 Закону "Про охорону праці" мають певне регулятивне значення. В статті закріплені такі принципи:

– пріоритет життя і здоров'я працівників відносно результатів виробничої діяльності підприємства, повна відповідальність власника за створення безпечних і нешкідливих умов праці;

– комплексне вирішення завдань охорони праці на засадах національних програм з цих питань, а також з урахуванням інших напрямків економічної і соціальної політики, досягнень в галузі науки, техніки охорони здоров'я;

– соціальний захист працівників, повне відшкодування шкоди особам потерпілим від нещасних випадків на виробництві та професійних захворювань;

– установлення єдиних нормативів з охорони праці для всіх підприємств незалежно від форм власності і видів їх діяльності;

– використання економічних методів управління охороною праці. проведення політики пільгового оподаткування, то сприяє створенню безпечних та нешкідливих умов праці, участь держави в фінансуванні заходів з охорони праці;

– здійснення навчання населення, професійної підготовки та підвищення кваліфікації працівників з питань охорони праці;

– забезпечення координації діяльності державних органів, установ, організацій та об'єднань громадян, що вирішують різні проблеми охорони здоров'я, гігієни та безпеки праці, а також співробітництво.

Державне керування органами охорони праці здійснюють: Кабінет Міністрів України, державний комітет України по нагляду за охороною праці, міністерство й інші централізовані органи, місцеві ради народних депутатів і їхні виконавчі комітети. Державний нагляд за дотриманням законодавства й інших нормативних актів по охороні праці здійснюють: державний комітет України по нагляду за охороною праці, державний комітет України по атомній і радіаційній безпеці, органи державного пожежного нагляду, органи і служби санітарно епідемічного нагляду. А трудові колективи в особі своїх уповноважених і профспілки здійснюють суспільний контроль за охороною праці.

Вищий нагляд за точним виконанням законів про працю міністерствами і відомствами, підприємствами, установами, громадянами і посадовими особами покладений на Генерального Прокурора України.

Велике значення в створенні здорових і безпечних умов праці має стандартизація. Вона дозволяє вжити діючих заходів по підвищенню технічного рівня й упорядкуванню розробки нормативно-технічної документації по безпеці праці. З 1972 року створюється Система стандартів по безпеці праці (ССБП), що являє собою комплекс великого числа взаємозалежних стандартів, спрямованих на забезпечення безпеки праці. Ця система встановлює загальні вимоги і норми по видах небезпечних і шкідливих виробничих факторів, загальні вимоги безпеки до виробничого устаткування і процесів, вимоги до засобів захисту працюючих, методи оцінки безпеки праці.

Охорона праці – один з центральних інститутів трудового права. Він має виключно практичне значення. Недодержання вимог охорони праці створює небезпеку для здоров'я і життя працівників. У свою чергу і ті, кого законодавець називає власником або уповноваженим ним органом, несуть сувору, у тому числі й кримінальну, відповідальність за порушення правил охорони праці.

Однією із характерних особливостей сучасного розвитку суспільства є зростання сфер діяльності людини, в яких використовуються інформаційні технології. Широке розповсюдження отримали персональні комп'ютери. Однак їх використання загострило проблеми збереження власного та суспільного здоров'я, вимагає удосконалення існуючих та розробки нових підходів до організації робочих місць, проведення профілактичних заходів для запобігання розвитку негативних наслідків впливу ПК на здоров'я користувачів. Тому питання охорони праці при роботі з ПК є досить актуальним.

5.2 Аналіз небезпечних і шкідливих факторів при роботі з ПК

З точки зору впливу небезпечних і шкідливих факторів при роботі з ПК можна виділити наступні:

- вплив електромагнітного та електростатичного випромінювань;

- вплив недостатньої освітленості робочого місця;
- вплив електричного струму;
- вплив шуму;
- вплив несприятливих умов мікроклімату;
- вплив нераціонального розташування устаткування і неправильної організації робочого місця.

Тривале перебування людини в зоні комбінованого впливу різних несприятливих факторів може призвести до професійного захворювання. Розглянемо більш детально вплив означених факторів на користувача ПК.

1) Вплив електромагнітного та електростатичного випромінювань.

При роботі ПК виникають два типи випромінювань: електростатичне та електромагнітне. Навколо електростатично-зарядженого монітора підвищується концентрація пилу. Електромагнітне випромінювання створюється магнітними котушками. Результати медичних досліджень показують, що електризований пил викликає запалення шкіри, а електромагнітне випромінювання приводить до порушення біологічних процесів (порушення сну, зниження кров'яного тиску, уповільнення серцевої діяльності, зміни складу крові).

2) Вплив недостатньої освітленості робочого місця.

Напружена зорова робота внаслідок нераціонального освітлення може бути причиною функціональних порушень у зоровому аналізаторі і привести до розладу зору, а у важких випадках – і до повної втрати. Втома органів зору залежить від ступеня напруженості процесів, що супроводжують зорове сприйняття. При роботі оператора ПК одним з основних навантажень на організм є навантаження на зір, тому дуже важливо приділити особливу увагу освітленості робочого місця.

У залежності від джерела світла виробниче висвітлення може бути трьох видів: природним, штучним і сполученим. При роботі з ПК вдень у ясну погоду використовується природне освітлення, яке здійснюється через світлові віконні прорізи. При нестачі природного світла, а також при роботі у вечірній час використовується штучне освітлення, джерелами якого є газорозрядні лампи. При

такому освітленні величезне значення має правильне розташування джерел світла, їхній тип і кількість.

Правильна освітленість робочого місця сприяє високій працездатності користувача ПК і зводить до мінімуму шкідливий вплив на органи зору. Освітленість робочого місця, що рекомендується для користувача ПК складає 400–750 лк.

3) Вплив електричного струму.

Робота комп'ютерів і периферійних пристроїв можлива завдяки електричному струмові, до джерел якого вони підключаються, тому існує небезпека ураження користувача струмом. Щоб уникнути нещасних випадків варто строго дотримуватися правил електробезпеки.

Небезпека електричного струму на відміну від інших небезпек збільшується за рахунок того, що людина не в змозі без спеціальних приладів виявити напругу дистанційно, а також швидкоплинності поразки – небезпека виявляється, коли людина вже уражена. Ураження струмом приводить до різних порушень в організмі, викликаючи як місцеву поразку тканин і органів, так і загальне ураження організму. Людина відчуває дратівну дію перемінного струму промислової частоти силою 0,6–1,5 мА і постійного струму 5–7 мА. Ці струми не представляють серйозної небезпеки для організму людини, тому що при такій силі струму можливе самостійне звільнення людини від контакту зі струмоведучими частинами. Для перемінного струму промислової частоти сила невідпускаючого струму знаходиться в межах 6–20 мА. Постійний струм не викликає невідпускаючого ефекту, але приводить до сильних болючих відчуттів, сила такого струму 15–80 мА і більше.

4) Вплив шуму.

Шумом прийнято називати всякий небажаний для людини звук, що заважає сприйняттю корисних сигналів. Шум являє собою безладне сполучення звуків різної інтенсивності і частоти. Він викликає труднощі з розпізнанням кольірних сигналів, знижує швидкість сприйняття кольору, гостроту зору, зорову адаптацію, порушує сприйняття візуальної інформації, зменшує на 5–12% продуктивність

праці. Тривалий вплив шуму з рівнем звукового тиску 90 дБ знижує продуктивність праці на 30–60%.

Робота ПК може супроводжуватися шумами, викликаними деякими рухливими пристроями усередині комп'ютера, а також підключеними до нього. До таких пристроїв можна віднести вентилятори, пристрої читання дисків, принтери. Виникаючий шум, у залежності від інтенсивності, може привести до підвищення нервової напруги і зниження працездатності.

5) Вплив несприятливих умов мікроклімату.

Мікрокліматичні параметри (температура, відносна вологість і швидкість руху повітря) впливають на самопочуття і здоров'я людини, а також на надійність роботи засобів обчислювальної техніки.

При підвищеній температурі і вологості в приміщенні відбувається перегрівання тіла, що погіршує самопочуття і працездатність. Варто враховувати, що ПК, які знаходяться в приміщенні, сприяють збільшенню температури.

Для комфортної роботи приймається температура $23 (\pm 2)^{\circ}\text{C}$ й $18 (\pm 2)^{\circ}\text{C}$ в теплий і холодний час року відповідно, при відносній вологості $55 (\pm 5)\%$.

6) Вплив нераціонального розташування устаткування і неправильної організації робочого місця.

Робоче місце – це частина простору, у якій користувач ПК здійснює трудову діяльність і проводить велику частину робочого часу. Важливу роль грає планування робочого місця, яке повинне задовольняти вимогам зручності виконання робіт і економії енергії і часу користувача, зручності обслуговувань ЕОМ, дотримання правил охорони праці.

При плануванні робочого місця необхідно враховувати зони досяжності при розташуванні моніторів, клавіатури, маніпулятора «миша». Ці зони встановлюються на підставі антропологічних даних людського тіла. Під час роботи користувач ПК не повинен піддаватися впливові сторони подразників, якими можуть бути неадекватне фарбування пристроїв ЕОМ або приміщення.

5.3 Розрахунок системи загального освітлення приміщення з ПК

Рациональне освітлення приміщень – один з найбільш важливих факторів, від яких залежить ефективність трудової діяльності людини. Гарне освітлення необхідне для виконання більшості задач.

До систем освітлення висувають наступні вимоги:

- відповідність рівня освітленості робочих місць характеру виконуваної зорової роботи;
- досить рівномірний розподіл яскравості на робочих поверхнях і в навколишньому просторі;
- сталість освітленості в часі;
- оптимальна спрямованість випромінюваного освітлювальними приладами світлового потоку;
- довговічність, економічність, електро- та пожежабезпечності, естетичність, зручність і простота експлуатації.

В приміщенні з ПК необхідно організувати змішане освітлення (комбінування природного і штучного освітлення). Природне освітлення створюється бічним освітленням через вікно. Штучне освітлення використовується при недостатньому природному освітленні. У даному приміщенні використовується загальне штучне освітлення. Розрахунок виконаємо за методом світлового потоку з урахування потоку, віддзеркаленого від стін та стелі.

Необхідно зробити розрахунок загального освітлення виробничої приміщення. Довжина $A = 10$ м, ширина $B = 7$ м, висота $H = 3$ м. Висота робочої поверхні $h_p = 1$ м. Для освітлення використовується світильні УПМ-15. Мінімальна освітленість лампи наколювання по нормах $E_{\min} = 100$ лк. Коефіцієнт відображення стелі $S_n = 50\%$, стін $S_c = 30\%$, робочої поверхні $S_p = 10\%$. Напруга мережі 220 В.

Відстань від стелі до робочої поверхні:

$$H_0 = H - h_p = 3 - 1 = 2 \text{ м.}$$

Відстань від стелі до світильника:

$$h_c = 0,2 \cdot H_0 = 0,2 \cdot 2 = 0,4 \text{ м.}$$

Висота підвісу світильника над освітленою поверхнею:

$$h = H_0 - h_c = 2 - 0,4 = 1,6 \text{ м.}$$

Висота підвісу світильника над підлогою:

$$H_n = h + h_p = 1,6 + 1 = 2,6 \text{ м.}$$

Для досягнення найбільшої рівномірності освітлення приймаємо відношення $L/h = 1,5$.

Відстань між центрами світильників:

$$L = 1,5 \cdot h = 1,5 \cdot 1,6 = 2,4 \text{ м.}$$

Необхідна кількість світильників:

$$N = S / L^2 = (10 \cdot 7) / (2,4)^2 \approx 12.$$

Приймаємо кількість світильників: 12 (3 ряди по 4 світильники).

Індекс приміщення:

$$i = (A \cdot B) / (h \cdot (A + B)) = (10 \cdot 7) / (1,6 \cdot (10 + 7)) = 2,57 \text{ (найближче значення в таблиці 2,5).}$$

По таблиці коефіцієнтів використання світлового потоку при $i=2,5$, $S_{\pi}=50\%$, $S_c=30\%$, $S_p=10\%$ для світильника типу «УПМ-15» коефіцієнт використання світлового потоку $\eta=0,59$.

Світловий потік однієї лампи:

$$\Phi = (E_{\min} \cdot S \cdot K_3 \cdot Z) / (N \cdot \eta) = (100 \cdot 70 \cdot 1,5 \cdot 1,1) / (12 \cdot 0,59) = 1631 \text{ лм, де}$$

K_3 – коефіцієнт запасу, який враховує запилення світильників і знос джерел світла в процесі експлуатації; для приміщень, освітлюваних люмінесцентними лампами $K_3=1,5$ за умови чищення світильників не рідше двох разів на рік;

Z – коефіцієнт нерівномірності освітлення ($Z=1,1$);

За знайденим світовим потоком із таблиці «Параметри ламп накаливання загального призначення з розрахунковими напругами 130 і 220 В» обираємо лампу Г215-225-150 накаливання потужністю 150 Вт, яка має світовий потік 2090 лк, що найбільше близький до розрахункового.

Фактична освітленість:

$$E_{\phi} = E_{\min} \cdot \Phi_{\text{л}} / \Phi_{\text{р}} = 100 \cdot 2090 / 1631 = 128,14 \text{ лк.}$$

Загальна потужність:

$$P_{\text{заг}} = P_{\text{л}} \cdot N = 150 \cdot 12 = 1800 \text{ Вт} = 1,8 \text{ кВт.}$$

Таким чином, для оптимальної роботи в приміщенні необхідно встановити 3 ряди по 4 світильники типу Г215-225-150. Загальна потужність усіх світильників складе 1,8 кВт.

5.4 Розробка заходів щодо забезпечення сприятливих умов праці при роботі з персональним комп'ютером

При виконанні робіт на комп'ютерах необхідно дотримуватись вимог загальної та даної інструкції з охорони праці.

До самостійної роботи на комп'ютерах допускаються особи, які пройшли медичний огляд, навчання по професії, вступний інструктаж з охорони праці та первинний інструктаж з охорони праці на робочому місці. В подальшому вони проходять повторні інструктажі з охорони праці на робочому місці один раз на півріччя, періодичні медичні огляди один раз на два роки.

Під час роботи на комп'ютерах можуть діяти такі небезпечні та шкідливі фактори, як:

- фізичні;
- психофізіологічні.

Основним обладнанням робочого місця користувача комп'ютера є монітор, системний блок та клавіатура.

Робочі місця мають бути розташовані на відстані не менше 1,5 м від стіни з вікнами, від інших стін на відстані 1м, між собою на відстані не менше 1,5 м. Відносно вікон робоче місце доцільно розташовувати таким чином, щоб природне світло падало на нього збоку, переважно зліва.

Робочі місця слід розташовувати так, щоб уникнути попадання в очі прямого світла. Джерела освітлення рекомендується розташовувати з обох боків екрану

паралельно напрямку погляду. Для уникнення світлових відблисків екрану, клавіатури в напрямку очей користувача, від світильників загального освітлення або сонячних променів, необхідно використовувати антиполюскові сітки, спеціальні фільтри для екранів, захисні козирки, на вікнах - жалюзі.

Фільтри з металевої або нейлонової сітки використовувати не рекомендується, тому що сітка спотворює зображення через інтерференцію світла. Найкращу якість зображення забезпечують скляні поляризаційні фільтри. Вони усувають практично всі відблиски, роблять зображення чітким і контрастним.

При роботі з текстовою інформацією (в режимі введення даних та редагування тексту, читання з екрану) найбільш фізіологічним правильним є зображення чорних знаків на світлому (чорному) фоні.

Монітор повинен бути розташований на робочому місці так, щоб поверхня екрана знаходилася в центрі поля зору на відстані 400-700 мм від очей користувача. Рекомендується розміщувати елементи робочого місця так, щоб витримувалася однакова відстань очей від екрана, клавіатури, тексту.

Зручна робоча поза при роботі з комп'ютером забезпечується регулюванням висоти робочого столу, крісла та підставки для ніг. Раціональною робочою позою може вважатися таке положення, при якому ступні працівника розташовані горизонтально на підлозі або підставці для ніг, стегна зорієнтовані у горизонтальній площині, верхні частини рук - вертикальні. Кут ліктьового суглоба коливається в межах 70-90°, зап'ястя зігнуті під кутом не більше ніж 20°, нахил голови 15-20°.

Для нейтралізації зарядів статичної електрики в приміщенні, де виконується робота на комп'ютерах, в тому числі на лазерних та світлодіодних принтерах, рекомендується збільшувати вологість повітря за допомогою кімнатних зволожувачів. Не рекомендується носити одяг з синтетичних матеріалів.

Вимоги безпеки перед початком роботи:

- увімкнути систему кондиціонування в приміщенні;
- перевірити надійність встановлення апаратури на робочому столі.

Повернути монітор так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) і трохи зверху вниз, при цьому екран має бути трохи нахиленим, нижній його край ближче до оператора;

- перевірити загальний стан апаратури, перевірити справність електропроводки, з'єднувальних шнурів, штепсельних вилок, розеток, заземлення захисного екрана;

- відрегулювати освітленість робочого місця;
- відрегулювати та зафіксувати висоту крісла, зручний для користувача нахил його спинки;

- приєднати до системного блоку необхідну апаратуру. Усі кабелі, що з'єднують системний блок з іншими пристроями, слід вставляти та виймати при вимкненому комп'ютері;

- ввімкнути апаратуру комп'ютера вимикачами на корпусах в послідовності: монітор, системний блок, принтер (якщо передбачається друкування);

- відрегулювати яскравість свічення монітора, мінімальний розмір світної точки, фокусування, контрастність. Не слід робити зображення надто яскравим, щоб не втомлювати очей.

Рекомендується:

- яскравість свічення екрана - не менше 100K_g/M²;
- відношення яскравості монітора до яскравості оточуючих його поверхонь в робочій зоні - не більше 3:1;

- мінімальний розмір точки свічення не більше 0,4 мм для монохромного монітора і не менше 0,6 мм для кольорового, контрастність зображення знаку – не менше 0,8.

При виявленні будь-яких несправностей роботу не розпочинати, повідомити про це керівника.

Вимоги безпеки під час виконання роботи:

- необхідно стійко розташовувати клавіатуру на робочому столі, не опускати її хитання. Під час роботи на клавіатурі сидіти прямо, не напружуватися;

- для забезпечення несприятливого впливу на користувача пристроїв типу

”миша” належить забезпечувати вільну велику поверхню столу для переміщення ”миші” і зручного упору ліктьового суглоба;

- не дозволяються по сторонні розмови, подразнюючі шуми;
- періодично при вимкненому комп'ютері прибирати ледь змоченою мильним розчином бавовняною ганчіркою порох з поверхонь апаратури. Екран ВДТ та захисний екран протирають ганчіркою, змоченою у спирті. Не дозволяється використовувати рідинні або аерозольні засоби чищення поверхонь комп'ютера.

Забороняється:

- самостійно ремонтувати апаратуру. Ремонт апаратури здійснюється спеціалістами з технічного обслуговування комп'ютера;
- класти будь-яку предмети на апаратуру комп'ютера;
- закривати будь-чим вентиляційні отвори апаратури, що може призвести до її перегрівання і виходу з ладу.

Для зняття статичної електрики рекомендується час від часу доторкатися до металевих поверхонь.

Розташувати принтер необхідно поруч з системним блоком таким чином, щоб з'єднувальний шнур не був натягнутий. Забороняється ставити принтери на системний блок.

Згідно з інструкцією фірми-виробника потрібно дотримуватися правил зберігання картриджа.

Забороняється:

- зберігати картриджі без упаковки;
- ставити картриджі вертикально;
- перевертати картридж етикеткою донизу;
- відкривати кришку валика і доторкатися до нього;
- самому заповнювати використаний картридж.

Вимоги безпеки після закінчення роботи:

- закінчити та записати у пам'ять комп'ютера файл, що знаходиться в роботі;
- вимкнути принтер та інші периферійні пристрої. Штепсельні вилки

витягнути з розеток;

- ретельно вимити руки теплою водою з милом;
- вимкнути кондиціонер, освітлення і загальне електроживлення;
- пройти в спеціально обладнаному приміщенні сеанс психофізіологічного розвантаження і зняття втоми з виконанням спеціальних вправ аутогенного тренування.

6 ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

6.1 Вступна частина

У наш час захист навколишнього середовища висувається на перший план. Наслідки недостатньої уваги до проблеми можуть бути катастрофічними. Мова йде не тільки про добробут людства, а про його виживання. Особливо тривожно те, що деградація природного середовища і може виявитися необоротною.

Під охороною навколишнього середовища розуміють сукупність міжнародних, державних і регіональних правових актів, інструкцій і стандартів, які доводять загальні юридичні вимоги до кожного конкретного забруднювача і забезпечують його зацікавленість у виконанні цих вимог, конкретних природоохоронних заходів по втіленню в життя цих вимог.

Тільки якщо всі ці складові частини відповідають один одному за змістом і темпами розвитку, тобто складаються в єдину систему охорони навколишнього природного середовища, можна розраховувати на успіх.

Оскільки не була вирішена вчасно завдання охорони природи від негативного впливу людини, тепер все частіше постає завдання захисту людини від впливу змінилася природного середовища. Обидва ці поняття інтегруються в терміні «охорона навколишнього (людини) природного середовища».

Охорона навколишнього природного середовища складається з:

- правової охорони, що формулює наукові екологічні принципи у вигляді юридичних законів, обов'язкових для виконання;
- матеріального стимулювання природоохоронної діяльності з ціллю зробити її економічно вигідною для підприємств;
- інженерної охорони, що розробляє природоохоронну та ресурсозберігаючу технологію і техніку.

Основними принципами охорони навколишнього природного середовища є:

- пріоритет забезпечення сприятливих екологічних умов для життя, праці та відпочинку населення;
- науково обґрунтоване поєднання екологічних і економічних інтересів суспільства;

- облік законів природи і можливостей самовідновлення і самоочищення її ресурсів;
- недопущення незворотних наслідків для охорони природного середовища і здоров'я людини;
- право населення та громадських організацій на своєчасну і достовірну інформацію про стан навколишнього середовища та негативному впливі на неї і на здоров'я людей різних виробничих об'єктів;
- невідворотність відповідальності за порушення вимог природоохоронного законодавства.

7.2 Забруднення навколишнього середовища від підприємства виготовлення паперу

Забруднення навколишнього середовища – надходження в природне середовище будь-яких твердих, газоподібних речовин, мікроорганізмів або енергій (у вигляді звуків, шумів, електромагнітного й радіоактивного випромінювання) у кількостях, що викликають зміну складу й властивостей компонентів природи й (або) чинять шкідливий вплив на людину, флору й фауну.

Види забруднень навколишнього середовища різні й різноманітні: викиди в атмосферу хімічних сполук і сумішей; надходження у водне середовище всіляких виробничих і комунально-побутових відходів, влучення в неї нафтових продуктів; засмічування ландшафтів сміттям і пакувальними матеріалами; забруднення полів, луків, лісів і водних об'єктів пестицидами; підвищення рівня іонізуючої радіації, виробничих і побутових шумів, вібрацій, а також нагромадження тепла в атмосфері.

Папір, винайдений в Китаї приблизно 2000 років тому, є екологічно безпечним матеріалом, оскільки він може біологічно розкладатися та піддаватися утилізації. На сьогоднішній день виробляють більше 3000 різних видів паперу. На прикладі одержання екологічно чистої продукції варто оцінити ступінь небезпеки такого виробництва для довкілля.

Процес виробництва ґрунтується на використанні деревини та включає змішування рослинних волокон з водою, вологе очищення, вибілювання, пресування та висушування.

Незважаючи на існування різних видів паперу та паперових продуктів, які виробляються у різні способи, основні методи виробництва залишаються такими самими протягом багатьох віків.

Процес виробництва паперу має негативні наслідки для довкілля. Головним джерелом утворення забруднених стічних вод при виробництві паперу є процес одержання целюлози, яке базується на сульфатному та сульфітному способах виварювання деревини та відбілюванні напівфабрикату хлором. При цьому в поверхневі води попадають сульфати, хлориди, органічні сірчисті сполуки (диметилсульфід і диметилдісульфід), в повітря викидаються меркаптани, які мають дуже неприємний запах. Використання хлору для відбілювання призводить до забруднення стічних вод поліхлорованими сполуками, серед яких дуже небезпечні ПХБ.

Для виробництва паперу вирубаються ліси, що призводить до зменшення кількості рідкісних тварин та рослин, ерозії ґрунту та зменшення кількості ґрунтових вод.



Рисунок 6.1 – Життєвий цикл паперу.

6.3 Заходи збереження природних ресурсів

Природоохоронною є будь-яка діяльність, спрямована на збереження якості навколишнього середовища на рівні, що забезпечує стійкість біосфери. До неї відноситься як великомасштабна, здійснювана на загальнодержавному рівні діяльність, так і діяльність окремих підприємств.

В усьому світі щоденно виробляється приблизно 650 000,00 тонн паперу, з них викидається 500 000,00 тонн, які могли б успішно перероблятися. Способи подолання проблеми:

1. Багаторазове використання паперу — користування бібліотеками, написання з обох боків листка паперу, використання підручників для навчання багатьох поколінь учнів та студентів, а також повторне використання картонних коробок для упаковки різних речей. Ці та інші способи допоможуть скоротити потребу у виробництві нових паперових виробів.
2. Обмеження обсягів споживання — комп'ютери (впровадження електронного обігу документів) можуть використовуватися як потужний засіб зменшення обсягів споживання паперу.
3. Використовувати сучасні технології, які не потребують використання паперу.
4. Зберегти ліси — утилізація бавовняних відходів та повторне використання картонних та паперових виробів. Для виробництва утилізованого паперу вимагається в 2-3 рази менше енергії, що допомагає зберегти приблизно п'ять гектарів лісів, які використовуються для виробництва однієї тонни паперу, і таким чином зменшити забруднення повітря на 75%, а води — на 60%. Немає сумнівів у тому, що утилізація паперу — надійний спосіб охорони довкілля.
5. Для вторинної переробки паперу потрібно майже в два рази менше енергії, ніж для її первинного виробництва з дерева, при цьому шкідливі викиди зменшуються більш, ніж на 70%.

ВИСНОВКИ

В результаті виконання магістерської роботи було досліджено засобів розподілених систем та виконано розробку інформаційного комплексу ідентифікації осіб з селекцією доступу на основі використання розподілених систем.

Комплекс складається з:

- Проект інформаційного забезпечення
 - Розробка діаграми варіантів використання
 - Розробка діаграми послідовності
 - Розробка діаграми діяльності
 - Модель даних системи контролю доступу
 - Фізична модель
- Проект технічного забезпечення
 - Розробка діаграми класів
 - Розробка діаграми пакетів
 - Розробка діаграми розгортання
- Проект програмного забезпечення
 - Мова програмування
 - База даних
 - Аспектно-орієнтоване програмування (АОП)
 - Середовища розробки програми

У подальшому планується впровадження комплексу забезпечення системи контролю та управління доступу та ідентифікації осіб.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Системи контролю доступу, СКУД - що це таке? [електронний ресурс]. — Режим доступу: URL: http://www.video-lux.org/articles_skud/1. — Загол. з екрану;
2. Ключові особливості Nedap AEOS [електронний ресурс]. — Режим доступу: URL: http://www.7person.com/nedap_aeos_security_system/nedapaeos_features/. — Загол. з екрану;
3. Full-height access controls for more security [електронний ресурс]. — Режим доступу: URL: <http://www.skidata.com/en/fairs-attractions-arenas/access-readers-turnstiles/drehkreuz-variogate0.html>. — Загол. з екрану;
4. PERCo-S-600 економічна СКД для невеликих і середніх підприємств [електронний ресурс]. — Режим доступу: URL: http://www.aktivsb.ua/cat217_713_714.html. — Загол. з екрану;
5. GATE-4000 [електронний ресурс]. — Режим доступу: <http://www.skdgate.ua/index.php?nomer=249>. — Загл. с екрана;
6. Система контролю доступа Keri Systems [електронний ресурс]. — Режим доступу: URL: http://www.building-m.com/kontrol_dostupa/skud_keri_systems/skud_keri_systems.html. — Загл. с екрана;
7. Depend upon Reliable Keri Products for Access and Security [електронний ресурс]. — Режим доступу: URL: <http://www.kerisys.com/pages/products/#doorsnet>. — Загол. з екрану;
8. Системи контролю доступу - основні поняття та особенности [електронний ресурс]. — Режим доступу: URL: <http://www.cleper.org/articles/description.php?n=175>. — Загол. з екрану;
9. Розподілена, централізована та змішана архітектура [електронний ресурс]. — Режим доступу: URL: http://www.sec-control.org/sec_27.htm. — Загол. з екрану;

10. Централізована архітектура СКУД [електронний ресурс]. — Режим доступу: URL: <http://skdgate.ua/stati/tsentralizovannaya-arhitektura-skud.html>. — Загол. з екрану;
11. Класифікація систем контролю доступу [електронний ресурс]. — Режим доступу : URL : <http://www.agoragroup.com.ua/tech/skd.html>. — Загол. з екрану;
12. Автономні СКУД [електронний ресурс]. — Режим доступу : URL : <http://www.hid.su/news/56/>. — Загол. з екрану;
13. Вибір та застосування систем контролю та управління доступом [електронний ресурс]. — Режим доступу: URL : <http://datasolution.ua/vybor-i-primenenie-sistem-kontrolya-i-upravleniya-dostupom/>. — Загол. з екрану;
14. Модель "Сутність-Зв'язок" | ER-модель [електронний ресурс]. — Режим доступу : URL : http://life-prog.ua/view_programmer.php?id=123&page=13/. — Загол. з екрану;
15. Мова програмування Java. Часть 1 [електронний ресурс]. — Режим доступу : URL : <http://club.shelek.org/viewart.php?id=141/>. — Загол. з екрану;
16. Что такое технология Java и зачем она мне нужна? [електронний ресурс]. — Режим доступа : URL : http://java.com/ru/download/faq/whatis_java.xml/. — Загл. с экрана;
17. Языки программирования [електронний ресурс]. — Режим доступа : URL : <http://edu.cps.tver.ru/archives/1175/>. — Загл. с экрана;
18. PostgreSQL. Самая продвинутая из открытых СУБД в мире [електронний ресурс]. — Режим доступа : URL : <http://www.sai.msu.su/~megera/postgres/talks/pgsql-short.html/>. — Загл. с экрана;
19. PostgreSQL Vs MySQL: Comparative Review [електронний ресурс]. — Режим доступа : URL : <http://tech.gaeatimes.com/index.php/archive/postgresql-vs-mysql-comparative-review/>. — Загол. з екрану;
20. Создание представлений [електронний ресурс]. — Режим доступа : URL : http://citforum.ru/database/sql_any/sql_059.shtml/. — Загл. с экрана;

ДОДАТОК А - ТЕХНІЧНЕ ЗАВДАННЯ

Підстави для розробки

Підставою для розробки є завдання на дослідження інформаційного комплексу ідентифікації осіб з селекцією доступу на основі використання розподілених систем.

Функціональне призначення системи

- заведення всіх необхідних параметрів системи (споруди, зони доступу, точки проходу);
- реєстрація ідентифікаторів відвідувачів, які будуть представляти собою штрих-коди проданих білетів та абонементів;
- прийом системою ідентифікаторів від відвідувачів;
- верифікація отриманого ідентифікатору в системі;
- формування результату.

Вимоги до надійності системи:

- Безперебійна робота сервера
- Повинне відбуватися резервне копіювання бази даних раз на місяць;
- Надійне з'єднання між сервером та клієнтами.

Вимоги до програмної документації:

В програмну документацію повинні входити:

- Текст програми;
- Опис програми;
- Програма та методика випробувань;
- Керівництво системного адміністратора.

Вимоги до складу та параметрів технічних засобів

Технічне забезпечення серверу

На сервері буде зберігатися існуюча база даних, яка містить інформацію про заходи, сектора, білети та інше...

Сервер повинен уміти обслуговувати багатьох клієнтів одночасно. В зв'язку з цим рекомендовані технічні характеристики сервера наступні:

- процесор - Intel® Core™ i3-2310M Processor (2.10 GHz), або краще;
- оперативна пам'ять - 2Гб або більше;
- вільне місце на жорсткому диску – 30Гб або більше;
- надійне з'єднання з мережею Інтернет, або підключення до локальної мережі компанії;
- монітор;
- маніпулятори – клавіатура та миша.

ДОДАТОК Б – ТЕХНІЧНИЙ ОПИС

Загальні відомості

Назва: Комплексу ідентифікації осіб з селекцією доступу на основі використання розподілених систем.

Основним засобом розробки програмного забезпечення системи є програмне середовище IntelliJ IDEA. Програмне забезпечення виконане мовою програмування Java. Інструмент для ведення бази даних – PostgreSQL

Функціональне призначення

Дана система автоматизує процес ідентифікації осіб для проходження на масовий захід. Вирішує проблему ідентифікації осіб, забезпечує безпеку повторного входу до заходу, швидкість ідентифікації а також формування звітів.

Опис логічної структури

При розробці програмного забезпечення системи використані методи об'єктно-орієнтованого проектування і розробки. Основними частинами програмного забезпечення є клієнтські додатки та серверна частина.

ДОДАТОК Г - ТЕКСТ ПРОГРАМНИХ МОДУЛІВ

Модуль WebServer

```
package com.pb.irbis_dept.acs.server.rest.service;

import com.pb.irbis_dept.acs.common.generate.*;
import com.pb.irbis_dept.acs.server.rest.controller.ServiceControllerI;
import com.pb.irbis_dept.acs.server.rest.utils.ResponseHandlerI;

import javax.ws.rs.core.Response;

public class Service implements ServiceI {

    private ServiceControllerI acsController;
    private ResponseHandlerI handler;

    public void setAcsController(ServiceControllerI acsController) {
        this.acsController = acsController;
    }

    public void setHandler(ResponseHandlerI handler) {
        this.handler = handler;
    }

    @Override
    public Response login(JAXBLoginReq loginReq) {
        return handler.handle4Answer(acsController.login(loginReq));
    }

    @Override
    public Response validate(JAXBValidateTicketReq ticketReq) {
        return handler.handle4Answer(acsController.validate(ticketReq));
    }

    @Override
    public Response addOuterTicket(JXBAddOuterTicketReq addTicketReq) {
        return handler.handle4Answer(acsController.addOuterTicket(addTicketReq));
    }

    @Override
    public Response addTurnstile(JXBAddTurnstileReq addTurnstileReq) {
        return handler.handle4Answer(acsController.addTurnstile(addTurnstileReq));
    }

    @Override
```



```

    public Response updateTurnstile(JAXBUpdateTurnstileReq jaxbUpdateTurnstileReq)
    {
        return
handler.handle4Answer(acsController.updateTurnstile(jaxbUpdateTurnstileReq));
    }

    @Override
    public void deleteTurnstile(JAXBDeleteTurnstileReq jaxbDeleteTurnstileReq) {
        acsController.deleteTurnstile(jaxbDeleteTurnstileReq);
    }

    @Override
    public Response accessZones(JAXBAccessZoneReq accessZoneReq) {
        return handler.handle4Answer(acsController.getAccessZones(accessZoneReq));
    }

    @Override
    public Response structures(JAXBStructureReq jaxbStructureReq) {
        return
handler.handle4Answer(acsController.getStructures(jaxbStructureReq));
    }

    @Override
    public Response facilities(JAXBFacilityReq jaxbFacilityReq) {
        return
handler.handle4Answer(acsController.getFacilities(jaxbFacilityReq));
    }

    @Override
    public Response seances(JAXBSeanceReq jaxbSeanceReq) {
        return handler.handle4Answer(acsController.getSeances(jaxbSeanceReq));
    }

    @Override
    public Response tickets(JAXBTicketReq jaxbTicketReq) {
        return handler.handle4Answer(acsController.getTickets(jaxbTicketReq));
    }

    @Override
    public Response turnstiles(JAXBTurnstileReq jaxbTurnstileReq) {
        return
handler.handle4Answer(acsController.getTurnstiles(jaxbTurnstileReq));
    }

    @Override
    public Response sectors(JAXBSectorReq jaxbSectorReq) {
        return handler.handle4Answer(acsController.getSectors(jaxbSectorReq));
    }

    @Override
    public Response restrictsAZT(JAXBRTAccessZoneTurnstileReq
jaxbrtAccessZoneTurnstileReq) {

```

```

        return
handler.handle4Answer(acsController.getRestrictsAZT(jaxbrtAccessZoneTurnstileReq))
;
    }

    @Override
    public Response addAccessZone(JAXBAddAccessZoneReq addAccessZoneReq) {
        return
handler.handle4Answer(acsController.addAccessZone(addAccessZoneReq));
    }

    @Override
    public Response updateAccessZone(JAXBUpdateAccessZoneReq updateAccessZoneReq)
{
        return
handler.handle4Answer(acsController.updateAccessZone(updateAccessZoneReq));
    }

    @Override
    public void deleteAccessZone(JAXBDeleteAccessZoneReq deleteAccessZoneReq) {
        acsController.deleteAccessZone(deleteAccessZoneReq);
    }

    @Override
    public Response addStructure(JAXBAddStructureReq jaxbAddStructureReq) {
        return
handler.handle4Answer(acsController.addStructure(jaxbAddStructureReq));
    }

    @Override
    public Response updateStructure(JAXBUpdateStructureReq jaxbUpdateStructureReq)
{
        return
handler.handle4Answer(acsController.updateStructure(jaxbUpdateStructureReq));
    }

    @Override
    public void deleteStructure(JAXBDeleteStructureReq jaxbDeleteStructureReq) {
        acsController.deleteStructure(jaxbDeleteStructureReq);
    }

    @Override
    public Response addFacility(JAXBAddFacilityReq jaxbAddFacilityReq) {
        return
handler.handle4Answer(acsController.addFacility(jaxbAddFacilityReq));
    }

    @Override
    public Response updateFacility(JAXBUpdateFacilityReq jaxbUpdateFacilityReq) {
        return
handler.handle4Answer(acsController.updateFacility(jaxbUpdateFacilityReq));
    }

```

```

@Override
public void deleteFacility(JAXBDeleteFacilityReq jaxbDeleteFacilityReq) {
    acsController.deleteFacility(jaxbDeleteFacilityReq);
}

@Override
public Response addSeance(JAXBAddSeanceReq jaxbAddSeanceReq) {
    return handler.handle4Answer(acsController.addSeance(jaxbAddSeanceReq));
}

@Override
public Response updateSeance(JAXBUpdateSeanceReq jaxbUpdateSeanceReq) {
    return
handler.handle4Answer(acsController.updateSeance(jaxbUpdateSeanceReq));
}

@Override
public void deleteSeance(JAXBDeleteSeanceReq jaxbDeleteSeanceReq) {
    acsController.deleteSeance(jaxbDeleteSeanceReq);
}

@Override
public Response addSector(JAXBAddSectorReq jaxbAddSectorReq) {
    return handler.handle4Answer(acsController.addSector(jaxbAddSectorReq));
}

@Override
public Response updateSector(JAXBUpdateSectorReq jaxbUpdateSectorReq) {
    return
handler.handle4Answer(acsController.updateSector(jaxbUpdateSectorReq));
}

@Override
public void deleteSector(JAXBDeleteSectorReq jaxbDeleteSectorReq) {
    acsController.deleteSector(jaxbDeleteSectorReq);
}

@Override
public Response addTicket(JAXBAddTicketReq jaxbAddTicketReq) {
    return handler.handle4Answer(acsController.addTicket(jaxbAddTicketReq));
}

@Override
public Response updateTicket(JAXBUpdateTicketReq jaxbUpdateTicketReq) {
    return
handler.handle4Answer(acsController.updateTicket(jaxbUpdateTicketReq));
}

@Override
public void deleteTicket(JAXBDeleteTicketReq jaxbDeleteTicketReq) {
    acsController.deleteTicket(jaxbDeleteTicketReq);
}

```

```

    }

    @Override
    public void addRTAccessZoneTurnstile(JAXBAddRTAccessZoneTurnstileReq
jaxbAddRTAccessZoneTurnstileReq) {
        acsController.addRestrictAZT(jaxbAddRTAccessZoneTurnstileReq);
    }

    @Override
    public void deleteRTAccessZoneTurnstile(JAXBDeleteRTAccessZoneTurnstileReq
jaxbDeleteRTAccessZoneTurnstileReq) {
        acsController.deleteRestrictAZT(jaxbDeleteRTAccessZoneTurnstileReq);
    }
}

package com.pb.irbis_dept.acs.server.rest.core;

import com.pb.irbis_dept.acs.common.generate.JAXBValidateTicketReq;
import com.pb.irbis_dept.acs.common.generate.JAXBValidateTicketResp;
import com.pb.irbis_dept.acs.server.rest.containers.ValidateTicket;
import com.pb.irbis_dept.acs.server.rest.containers.ValidateTicketInfo;
import com.pb.irbis_dept.acs.server.rest.data_access.DaoI;
import com.pb.irbis_dept.acs.server.rest.exception.ACSEException;

public class Core implements CoreI {

    private DaoI db;
    private ConverterI converter;

    public void setConverter(ConverterI converter) {
        this.converter = converter;
    }

    public void setDb(DaoI db) {
        this.db = db;
    }

    @Override
    public JAXBValidateTicketResp validate(JAXBValidateTicketReq ticketReq) {
        ValidateTicket validateTicket = converter.getValidateTicket(ticketReq);
        ValidateTicketInfo validate = db.validate(validateTicket);
        return converter.getValidateTicketResp(validate);
    }
}

package com.pb.irbis_dept.acs.server.rest.core;

import com.pb.irbis_dept.acs.common.generate.*;
import com.pb.irbis_dept.acs.server.rest.containers.*;
import com.pb.irbis_dept.acs.server.rest.data_access.AdminDaoI;
import com.pb.irbis_dept.acs.server.rest.data_access.CacheManagerI;
import com.pb.irbis_dept.acs.server.rest.utils.SessionManager;

public class AdminCore implements AdminCoreI {

```

```

private AdminConverterI adminConverter;
private AdminDaoI adminDao;
private CacheManagerI cacheManager;

public void setCacheManager(CacheManagerI cacheManager) {
    this.cacheManager = cacheManager;
}

public void setAdminConverter(AdminConverterI adminConverter) {
    this.adminConverter = adminConverter;
}

public void setAdminDao(AdminDaoI adminDao) {
    this.adminDao = adminDao;
}

@Override
public JAXBAddOuterTicketResp addOuterTicket(JAXBAddOuterTicketReq
addTicketReq) {
    AddOuterTicket addOuterTicket =
adminConverter.getAddOuterTicket(addTicketReq);
    return
adminConverter.getAddOuterTicketResp(adminDao.addOuterTicket(addOuterTicket));
}

@Override
public JAXBAddTurnstileResp addTurnstile(JAXBAddTurnstileReq addTurnstileReq)
{
    Turnstile turnstile = adminConverter.getAddTurnstile(addTurnstileReq);
    return
adminConverter.getAddTurnstileResp(adminDao.addOrUpdateTurnstile(turnstile));
}

@Override
public JAXBAccessZoneResp getAccessZones(JAXBAccessZoneReq accessZoneReq) {
    AccessZoneFilter accessZoneFilter =
adminConverter.getAccessZoneFilter(accessZoneReq);
    return
adminConverter.getAccessZones(adminDao.getAccessZones(accessZoneFilter));
}

@Override
public JAXBAddAccessZoneResp addAccessZone(JAXBAddAccessZoneReq
addAccessZoneReq) {
    AccessZone accessZone = adminConverter.getAddAccessZone(addAccessZoneReq);
    return
adminConverter.getAddAccessZoneResp(adminDao.addOrUpdateAccessZone(accessZone));
}

@Override
public JAXBUpdateAccessZoneResp updateAccessZone(JAXBUpdateAccessZoneReq
updateAccessZoneReq) {

```

```

        AccessZone                accessZone                =
adminConverter.getUpdateAccessZone(updateAccessZoneReq);
        return
adminConverter.getUpdateAccessZoneResp(adminDao.addOrUpdateAccessZone(accessZone))
;
    }

    @Override
    public void deleteAccessZone(JAXBDeleteAccessZoneReq deleteAccessZoneReq) {
        AccessZone                accessZone                =
adminConverter.getDeleteAccessZone(deleteAccessZoneReq);
        adminDao.deleteAccessZone(accessZone);
    }

    @Override
    public JAXBUpdateTurnstileResp updateTurnstile(JAXBUpdateTurnstileReq
jaxbUpdateTurnstileReq) {
        Turnstile                turnstile                =
adminConverter.getUpdateTurnstile(jaxbUpdateTurnstileReq);
        return
adminConverter.getUpdateTurnstileResp(adminDao.addOrUpdateTurnstile(turnstile));
    }

    @Override
    public void deleteTurnstile(JAXBDeleteTurnstileReq jaxbDeleteTurnstileReq) {
        Turnstile                turnstile                =
adminConverter.getDeleteTurnstile(jaxbDeleteTurnstileReq);
        adminDao.deleteTurnstile(turnstile);
    }

    @Override
    public JAXBAddStructureResp addStructure(JAXBAddStructureReq
jaxbAddStructureReq) {
        Structure structure = adminConverter.getAddStructure(jaxbAddStructureReq);
        return
adminConverter.getAddStructureResp(adminDao.addOrUpdateStructure(structure));
    }

    @Override
    public JAXBUpdateStructureResp updateStructure(JAXBUpdateStructureReq
jaxbUpdateStructureReq) {
        Structure                structure                =
adminConverter.getUpdateStructure(jaxbUpdateStructureReq);
        return
adminConverter.getUpdateStructureResp(adminDao.addOrUpdateStructure(structure));
    }

    @Override
    public void deleteStructure(JAXBDeleteStructureReq jaxbDeleteStructureReq) {
        Structure                structure                =
adminConverter.getDeleteStructure(jaxbDeleteStructureReq);
        adminDao.deleteStructure(structure);
    }

```

```

    }

    @Override
    public JAXBAddFacilityResp addFacility(JAXBAddFacilityReq jaxbAddFacilityReq)
    {
        Facility facility = adminConverter.getAddFacility(jaxbAddFacilityReq);
        return
        adminConverter.getAddFacilityResp(adminDao.addOrUpdateFacility(facility));
    }

    @Override
    public JAXBUpdateFacilityResp updateFacility(JAXBUpdateFacilityReq
    jaxbUpdateFacilityReq) {
        Facility facility =
        adminConverter.getUpdateFacility(jaxbUpdateFacilityReq);
        return
        adminConverter.getUpdateFacilityResp(adminDao.addOrUpdateFacility(facility));
    }

    @Override
    public void deleteFacility(JAXBDeleteFacilityReq jaxbDeleteFacilityReq) {
        Facility facility =
        adminConverter.getDeleteFacility(jaxbDeleteFacilityReq);
        adminDao.deleteFacility(facility);
    }

    @Override
    public JAXBAddSeanceResp addSeance(JAXBAddSeanceReq jaxbAddSeanceReq) {
        Seance seance = adminConverter.getAddSeance(jaxbAddSeanceReq);
        return
        adminConverter.getAddSeanceResp(adminDao.addOrUpdateSeance(seance));
    }

    @Override
    public JAXBUpdateSeanceResp updateSeance(JAXBUpdateSeanceReq
    jaxbUpdateSeanceReq) {
        Seance seance = adminConverter.getUpdateSeance(jaxbUpdateSeanceReq);
        return
        adminConverter.getUpdateSeanceResp(adminDao.addOrUpdateSeance(seance));
    }

    @Override
    public void deleteSeance(JAXBDeleteSeanceReq jaxbDeleteSeanceReq) {
        Seance seance = adminConverter.getDeleteSeance(jaxbDeleteSeanceReq);
        adminDao.deleteSeance(seance);
    }

    @Override
    public JAXBAddSectorResp addSector(JAXBAddSectorReq jaxbAddSectorReq) {
        Sector sector = adminConverter.getAddSector(jaxbAddSectorReq);
        return
        adminConverter.getAddSectorResp(adminDao.addOrUpdateSector(sector));
    }

```

```

    }

    @Override
    public JAXBUpdateSectorResp updateSector(JAXBUpdateSectorReq
jaxbUpdateSectorReq) {
        Sector sector = adminConverter.getUpdateSector(jaxbUpdateSectorReq);
        return
adminConverter.getUpdateSectorResp(adminDao.addOrUpdateSector(sector));
    }

    @Override
    public void deleteSector(JAXBDeleteSectorReq jaxbDeleteSectorReq) {
        Sector sector = adminConverter.getDeleteSector(jaxbDeleteSectorReq);
        adminDao.deleteSector(sector);
    }

    @Override
    public JAXBAddTicketResp addTicket(JAXBAddTicketReq jaxbAddTicketReq) {
        Ticket ticket = adminConverter.getAddTicket(jaxbAddTicketReq);
        return
adminConverter.getAddTicketResp(adminDao.addOrUpdateTicket(ticket));
    }

    @Override
    public JAXBUpdateTicketResp updateTicket(JAXBUpdateTicketReq
jaxbUpdateTicketReq) {
        Ticket ticket = adminConverter.getUpdateTicket(jaxbUpdateTicketReq);
        return
adminConverter.getUpdateTicketResp(adminDao.addOrUpdateTicket(ticket));
    }

    @Override
    public void deleteTicket(JAXBDeleteTicketReq jaxbDeleteTicketReq) {
        Ticket ticket = adminConverter.getDeleteTicket(jaxbDeleteTicketReq);
        adminDao.deleteTicket(ticket);
    }

    @Override
    public void addRestrictAZT(JAXBAddRTAccessZoneTurnstileReq
jaxbAddRTAccessZoneTurnstileReq) {
        RestrictAZTurnstile restrictAzTurnstile =
adminConverter.getAddRestrictAzTurnstile(jaxbAddRTAccessZoneTurnstileReq);
        adminDao.addRestrictAZT(restrictAzTurnstile);
    }

    @Override
    public void deleteRestrictAZT(JAXBDeleteRTAccessZoneTurnstileReq
jaxbDeleteRTAccessZoneTurnstileReq) {
        RestrictAZTurnstile restrictAzTurnstile =
adminConverter.getDeleteRestrictAzTurnstile(jaxbDeleteRTAccessZoneTurnstileReq);
        adminDao.deleteRestrictAZT(restrictAzTurnstile);
    }

```



```

@Override
public JAXBStructureResp getStructures(JAXBStructureReq jaxbStructureReq) {
    return adminConverter.getStructures(adminDao.getStructures());
}

@Override
public JAXBFacilityResp getFacilities(JAXBFacilityReq jaxbFacilityReq) {
    FacilityFilter facilityFilter =
adminConverter.getFacilityFilter(jaxbFacilityReq);
    return
adminConverter.getFacilities(adminDao.getFacilities(facilityFilter));
}

@Override
public JAXBTurnstileResp getTurnstiles(JAXBTurnstileReq jaxbTurnstileReq) {
    TurnstileFilter turnstileFilter =
adminConverter.getTurnstileFilter(jaxbTurnstileReq);
    return
adminConverter.getTurnstiles(adminDao.getTurnstiles(turnstileFilter));
}

@Override
public JAXBSeanceResp getSeances(JAXBSeanceReq jaxbSeanceReq) {
    SeanceFilter seanceFilter = adminConverter.getSeanceFilter(jaxbSeanceReq);
    return adminConverter.getSeances(adminDao.getSeances(seanceFilter));
}

@Override
public JAXBSectorResp getSector(JAXBSectorReq jaxbSectorReq) {
    SectorFilter sectorFilter = adminConverter.getSectorFilter(jaxbSectorReq);
    return adminConverter.getSectors(adminDao.getSectors(sectorFilter));
}

@Override
public JAXBRTAccessZoneTurnstileResp
getRestrictsAZT(JAXBRTAccessZoneTurnstileReq jaxbrtAccessZoneTurnstileReq) {
    RestrictAZTFilter restrictAZTFilter =
adminConverter.getRestrictAZTFilter(jaxbrtAccessZoneTurnstileReq);
    return
adminConverter.getRestrictAZT(adminDao.getRestrictsAZT(restrictAZTFilter));
}

@Override
public JAXBTicketResp getTickets(JAXBTicketReq jaxbTicketReq) {
    TicketFilter ticketFilter = adminConverter.getTicketFilter(jaxbTicketReq);
    return adminConverter.getTickets(adminDao.getTickets(ticketFilter));
}

@Override
public JAXBLoginResp login(JAXBLoginReq loginReq) {
    Login login = adminConverter.getLogin(loginReq);
}

```

```

        UserInfo userInfo = adminDao.login(login);
        String session = SessionManager.generateSession();
        cacheManager.storeUser(session, userInfo);
        return adminConverter.getLoginResp(session, userInfo);
    }
}

package com.pb.irbis_dept.acs.server.rest.data_access;

import com.assembla.nosan.sqb.SQB;
import com.assembla.nosan.sqb.element.ComplexElement;
import com.pb.common.db.manager.DBManagerFactoryI;
import com.pb.common.db.manager.DBManagerI;
import com.pb.irbis_dept.acs.server.rest.containers.*;
import com.pb.irbis_dept.acs.server.rest.exception.ACSEException;
import com.pb.irbis_dept.acs.server.rest.utils.CastTypeManager;
import com.pb.irbisdept.common.utils.money.Money;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class AdminDao implements AdminDaoI {

    private DBManagerFactoryI dbManagerFactory;
    private SQB sqb;
    final static Logger LOGGER = LoggerFactory.getLogger(AdminDao.class);

    public void setSqb(SQB sqb) {
        this.sqb = sqb;
    }

    public void setDbManagerFactory(DBManagerFactoryI dbManagerFactory) {
        this.dbManagerFactory = dbManagerFactory;
    }

    @Override
    public UserInfo login(Login login) {
        String query = sqb.setFunctionName("acs.authenticate_user").build(new
ComplexElement(login, Login.class));
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet rs = null;
        UserInfo userInfo = new UserInfo();
        try {
            rs = dbManager.executeQuery(query, LOGGER);
            while (rs.next()) {
                userInfo.setId(rs.getInt("r_pkey"));
                userInfo.setLogin(rs.getString("r_login"));
                userInfo.setFirstName(rs.getString("r_first_name"));
            }
        }
    }
}

```

```

        userInfo.setSecondName(rs.getString("r_second_name"));
    }
} catch (SQLException e) {
    throw new ACSEException(e.getMessage(), e);
} finally {
    dbManager.close(rs, LOGGER);
}
return userInfo;
}

@Override
public AddTicketInfo addOuterTicket(AddOuterTicket addOuterTicket) {
    String query = sqb.setFunctionName("acs.write_tickets").build(new
ComplexElement(addOuterTicket, AddOuterTicket.class));
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet rs = null;
    AddTicketInfo addTicketInfo = new AddTicketInfo();
    try {
        rs = dbManager.executeQuery(query, LOGGER);
        while (rs.next()) {
            addTicketInfo.setTicketNew(rs.getInt("r_count_new"));
            addTicketInfo.setTicketUpdate(rs.getInt("r_count_exists"));
        }
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(rs, LOGGER);
    }
    return addTicketInfo;
}

@Override
public AccessZone addOrUpdateAccessZone(AccessZone accessZone) {
    String query = new StringBuilder(sqb.setCastType("acs.access_zone").
setFunctionName("acs.add_or_update_access_zone").build(new
ComplexElement(accessZone, AccessZone.class))).append(" AS r_pkey").toString();
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet rs = null;
    AccessZone addAccessZoneInfo = new AccessZone();
    try {
        rs = dbManager.executeQuery(query, LOGGER);
        while (rs.next()) {
            addAccessZoneInfo.setId(rs.getInt("r_pkey"));
        }
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(rs, LOGGER);
    }
    return addAccessZoneInfo;
}

@Override

```

```

        public Turnstile addOrUpdateTurnstile(Turnstile turnstile) {
            String query = new
StringBuilder(sqb.setFunctionName("acs.add_or_update_turnstile").
                setCastType("acs.turnstile").build(new ComplexElement(turnstile,
Turnstile.class))).append(" AS r_pkey").toString();
            DBManagerI dbManager = dbManagerFactory.getDBManager();
            ResultSet rs = null;
            Turnstile addTurnstileInfo = new Turnstile();
            try {
                rs = dbManager.executeQuery(query, LOGGER);
                while (rs.next()) {
                    addTurnstileInfo.setId(rs.getInt("r_pkey"));
                }
            } catch (SQLException e) {
                throw new ACSEException(e.getMessage(), e);
            } finally {
                dbManager.close(rs, LOGGER);
            }
            return addTurnstileInfo;
        }

        @Override
        public List<AccessZoneInfo> getAccessZones(AccessZoneFilter accessZoneFilter)
        {
            String query = sqb.setFunctionName("acs.get_access_zones").build(new
ComplexElement(accessZoneFilter, AccessZoneFilter.class));
            DBManagerI dbManager = dbManagerFactory.getDBManager();
            ResultSet rs = null;
            List<AccessZoneInfo> accessZoneList = new ArrayList<>();
            try {
                rs = dbManager.executeQuery(query, LOGGER);
                while (rs.next()) {
                    AccessZoneInfo accessZoneInfo = new AccessZoneInfo();
                    AccessZone accessZone = new AccessZone();
                    accessZone.setActive(rs.getBoolean("r_active"));
                    accessZone.setId(rs.getInt("r_pkey_access_zone"));
                    accessZone.setName(rs.getString("r_description_access_zone"));
                    AccessZone parentZone = new AccessZone();
                    parentZone.setName(rs.getString("r_description_parent"));

                    parentZone.setId(CastTypeManager.getIntValue(rs.getString("r_pkey_parent")));
                    Facility facility = new Facility();
                    facility.setName(rs.getString("r_description_facility"));

                    facility.setId(CastTypeManager.getIntValue(rs.getString("r_pkey_facility")));
                    Structure structure = new Structure();
                    structure.setName(rs.getString("r_description_structure"));
                    structure.setId(rs.getInt("r_pkey_structure"));
                    accessZoneInfo.setAccessZone(accessZone);
                    accessZoneInfo.setParentAccessZone(parentZone);
                    accessZoneInfo.setFacility(facility);
                    accessZoneInfo.setStructure(structure);
                    accessZoneList.add(accessZoneInfo);
                }
            }
        }

```

```

        }
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(rs, LOGGER);
    }
    return accessZoneList;
}

@Override
public void deleteAccessZone(AccessZone accessZone) {
    String query = sqb.setFunctionName("acs.delete_access_zone").build(new
ComplexElement(accessZone, AccessZone.class).
        setExclude("name", "zoneType", "parentZoneId", "active",
"facilityId", "structureId"));
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet resultSet = null;
    try {
        resultSet = dbManager.executeQuery(query, LOGGER);
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(resultSet, LOGGER);
    }
}

@Override
public void deleteTurnstile(Turnstile deleteTurnstile) {
    String query = sqb.setFunctionName("acs.delete_turnstile").build(new
ComplexElement(deleteTurnstile, Turnstile.class).
        setExclude("identifier", "name", "passageType", "active",
"facilityId", "structureId"));
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet resultSet = null;
    try {
        resultSet = dbManager.executeQuery(query, LOGGER);
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(resultSet, LOGGER);
    }
}

@Override
public Structure addOrUpdateStructure(Structure structure) {
    Structure result = new Structure();
    String query =
String
Builder(sqb.setFunctionName("acs.add_or_update_structure").setCastType("acs.
structure").

```

```

        build(new ComplexElement(structure, Structure.class))).
        append(" AS r_pkey").toString();
DBManagerI dbManager = dbManagerFactory.getDBManager();
ResultSet resultSet = null;
try {
    resultSet = dbManager.executeQuery(query, LOGGER);
    while (resultSet.next()) {
        result.setId(resultSet.getInt("r_pkey"));
    }
} catch (SQLException e) {
    throw new ACSEException(e.getMessage(), e);
} finally {
    dbManager.close(resultSet, LOGGER);
}
return result;
}

@Override
public void deleteStructure(Structure structure) {
    String query = sqb.setFunctionName("acs.delete_structure").build(new
ComplexElement(structure, Structure.class).setExclude("name", "refId",
"jaxbTicketSystem"));
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet resultSet = null;
    try {
        resultSet = dbManager.executeQuery(query, LOGGER);
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(resultSet, LOGGER);
    }
}

@Override
public Facility addOrUpdateFacility(Facility facility) {
    Facility result = new Facility();
    String query = new
StringBuilder(sqb.setFunctionName("acs.add_or_update_facility").setCastType("acs.f
acility").build(new ComplexElement(facility, Facility.class))).
        append(" AS r_pkey").toString();
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet resultSet = null;
    try {
        resultSet = dbManager.executeQuery(query, LOGGER);
        while (resultSet.next()) {
            result.setId(resultSet.getInt("r_pkey"));
        }
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(resultSet, LOGGER);
    }
}

```

```

        return result;
    }

    @Override
    public void deleteFacility(Facility facility) {
        String query = sqb.setFunctionName("acs.delete_facility").build(new
ComplexElement(facility, Facility.class).setExclude("name", "refId",
"structureId"));
        ResultSet resultSet = null;
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
    }

    @Override
    public Seance addOrUpdateSeance(Seance seance) {
        String query = sqb.setFunctionName("acs.add_or_update_seance").setCastType("acs.sea
n ce").build(new ComplexElement(seance, Seance.class)).
append(" AS r_pkey").toString();
        Seance result = new Seance();
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
            while (resultSet.next()) {
                result.setId(resultSet.getInt("r_pkey"));
            }
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
        return result;
    }

    @Override
    public void deleteSeance(Seance seance) {
        String query = sqb.setFunctionName("acs.delete_seance").build(new
ComplexElement(seance, Seance.class).setExclude("refId", "name", "facilityId",
"seanceStart", "entranceStart", "entranceEnd"));
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        }
    }

```

```

        } finally {
            dbManager.close(resultSet, LOGGER);
        }
    }

    @Override
    public Sector addOrUpdateSector(Sector sector) {
        String query = new
        StringBuilder(sqb.setFunctionName("acs.add_or_update_sector").setCastType("acs.sec
        tor").build(new ComplexElement(sector, Sector.class))).
            append(" AS r_pkey").toString();
        Sector result = new Sector();
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
            while (resultSet.next()) {
                result.setId(resultSet.getInt("r_pkey"));
            }
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
        return result;
    }

    @Override
    public void deleteSector(Sector sector) {
        String query = sqb.setFunctionName("acs.delete_sector").build(new
        ComplexElement(sector, Sector.class).
            setExclude("refId", "name", "facilityId", "accessZoneId"));
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
    }

    @Override
    public Ticket addOrUpdateTicket(Ticket ticket) {
        String query = new
        StringBuilder(sqb.setFunctionName("acs.add_or_update_ticket").setCastType("acs.tic
        ket").build(new ComplexElement(ticket, Ticket.class))).
            append(" AS r_pkey").toString();
        Ticket result = new Ticket();
        DBManagerI dbManager = dbManagerFactory.getDBManager();
    }

```



```

        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
            while (resultSet.next()) {
                result.setId(resultSet.getInt("r_pkey"));
            }
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
        return result;
    }

    @Override
    public void deleteTicket(Ticket ticket) {
        String query = sqb.setFunctionName("acs.delete_ticket").build(new
        ComplexElement(ticket, Ticket.class).
            setExclude("code", "seatRow", "seatNumber", "price", "sellDate",
            "refId", "type", "seanceId", "sectorId", "state", "delete"));
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
    }

    @Override
    public void addRestrictAZT(RestrictAZTurnstile restrictAzTurnstile) {
        String query = sqb.setFunctionName("acs.add_rt_access_zone_turnstile").build(new
        ComplexElement(restrictAzTurnstile, RestrictAZTurnstile.class));
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
    }

    @Override
    public void deleteRestrictAZT(RestrictAZTurnstile restrictAzTurnstile) {

```

```

        String query =
sqb.setFunctionName("acs.delete_rt_access_zone_turnstile").build(new
ComplexElement(restrictAzTurnstile, RestrictAZTurnstile.class));
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
    }

    @Override
    public List<StructureInfo> getStructures() {
        String query = sqb.setFunctionName("acs.get_all_structure").build();
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        List<StructureInfo> structures = new ArrayList<>();
        ResultSet resultSet = null;
        try {
            resultSet = dbManager.executeQuery(query, LOGGER);
            while (resultSet.next()) {
                StructureInfo structureInfo = new StructureInfo();
                structureInfo.setId(resultSet.getInt("r_pkey_structure"));

structureInfo.setName(resultSet.getString("r_description_structure"));

structureInfo.setReference(CastTypeManager.getIntValue(resultSet.getString("r_pkey
_ref_structure")));

structureInfo.setTicketSystemId(CastTypeManager.getIntValue(resultSet.getString("r
_pkey_ticket_system")));

structureInfo.setTicketSystemName(resultSet.getString("r_description_ticket_system
"));

                structures.add(structureInfo);
            }
        } catch (SQLException e) {
            throw new ACSEException(e.getMessage(), e);
        } finally {
            dbManager.close(resultSet, LOGGER);
        }
        return structures;
    }

    @Override
    public List<FacilityInfo> getFacilities(FacilityFilter facilityFilter) {
        String query = sqb.setFunctionName("acs.get_facilities").build(new
ComplexElement(facilityFilter, FacilityFilter.class));
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet resultSet = null;

```

```

List<FacilityInfo> facilities = new ArrayList<>();
try {
    resultSet = dbManager.executeQuery(query, LOGGER);
    while (resultSet.next()) {
        FacilityInfo facilityInfo = new FacilityInfo();
        Facility facility = new Facility();
        facility.setId(resultSet.getInt("r_pkey_facility"));
        facility.setName(resultSet.getString("r_description_facility"));
        facility.setRefId(CastTypeManager.getIntValue("r_pkey_ref"));
        Structure structure = new Structure();
        structure.setId(resultSet.getInt("r_pkey_structure"));
        structure.setName(resultSet.getString("r_description_structure"));
        facilityInfo.setFacility(facility);
        facilityInfo.setStructure(structure);
        facilities.add(facilityInfo);
    }
} catch (SQLException e) {
    throw new ACSEException(e.getMessage(), e);
} finally {
    dbManager.close(resultSet, LOGGER);
}
return facilities;
}

@Override
public List<TurnstileInfo> getTurnstiles(TurnstileFilter turnstileFilter) {
    String query = sqb.setFunctionName("acs.get_turnstiles").build(new
ComplexElement(turnstileFilter, TurnstileFilter.class));
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet resultSet = null;
    List<TurnstileInfo> turnstiles = new ArrayList<>();
    try {
        resultSet = dbManager.executeQuery(query, LOGGER);
        while (resultSet.next()) {
            TurnstileInfo turnstileInfo = new TurnstileInfo();
            Turnstile turnstile = new Turnstile();
            turnstile.setId(resultSet.getInt("r_pkey_turnstile"));
            turnstile.setName(resultSet.getString("r_description_turnstile"));
            turnstile.setActive(resultSet.getBoolean("r_active"));
            turnstile.setIdentifier(resultSet.getString("r_identifier"));
            Facility facility = new Facility();

facility.setId(CastTypeManager.getIntValue(resultSet.getString("r_pkey_facility"))
);

            facility.setName(resultSet.getString("r_description_facility"));
            Structure structure = new Structure();
            structure.setId(resultSet.getInt("r_pkey_structure"));
            structure.setName(resultSet.getString("r_description_structure"));
            TurnstileInfo.Passage passage = new TurnstileInfo.Passage();
            passage.setId(resultSet.getInt("r_pkey_passage_type"));

passage.setName(resultSet.getString("r_description_passage_type"));

```

```

        turnstileInfo.setFacility(facility);
        turnstileInfo.setPassage(passage);
        turnstileInfo.setStructure(structure);
        turnstileInfo.setTurnstile(turnstile);
        turnstiles.add(turnstileInfo);
    }
} catch (SQLException e) {
    throw new ACSEException(e.getMessage(), e);
} finally {
    dbManager.close(resultSet, LOGGER);
}

return turnstiles;
}

@Override
public List<SeanceInfo> getSeances(SeanceFilter seanceFilter) {
    String query = sqb.setFunctionName("acs.get_seances").build(new
ComplexElement(seanceFilter, SeanceFilter.class));
    List<SeanceInfo> seances = new ArrayList<>();
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet resultSet = null;
    try {
        resultSet = dbManager.executeQuery(query, LOGGER);
        while (resultSet.next()) {
            SeanceInfo seanceInfo = new SeanceInfo();
            Seance seance = new Seance();

seance.setEntranceEnd(resultSet.getTimestamp("r_entrance_end_datetime"));

seance.setEntranceStart(resultSet.getTimestamp("r_entrance_start_datetime"));

seance.setSeanceStart(resultSet.getTimestamp("r_seance_start_datetime"));

seance.setRefId(CastTypeManager.getIntValue(resultSet.getString("r_pkey_ref_seance
"))));

            seance.setName(resultSet.getString("r_description_seance"));
            seance.setId(resultSet.getInt("r_pkey_seance"));
            Facility facility = new Facility();
            facility.setId(resultSet.getInt("r_pkey_facility"));
            facility.setName(resultSet.getString("r_description_facility"));
            seanceInfo.setFacility(facility);
            seanceInfo.setSeance(seance);
            seances.add(seanceInfo);
        }
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(resultSet, LOGGER);
    }

    return seances;
}

```

```

@Override
public List<SectorInfo> getSectors(SectorFilter sectorFilter) {
    String query = sqb.setFunctionName("acs.get_sectors").build(new
ComplexElement(sectorFilter, SectorFilter.class));
    List<SectorInfo> sectors = new ArrayList<>();
    ResultSet resultSet = null;
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    try {
        resultSet = dbManager.executeQuery(query, LOGGER);
        while (resultSet.next()) {
            SectorInfo sectorInfo = new SectorInfo();
            Sector sector = new Sector();
            sector.setId(resultSet.getInt("r_sector_pkey"));
            sector.setName(resultSet.getString("r_sector_description"));

sector.setRefId(CastTypeManager.getIntValue(resultSet.getString("r_pkey_ref")));

            Facility facility = new Facility();
            facility.setId(resultSet.getInt("r_facility_pkey"));
            facility.setName(resultSet.getString("r_facility_description"));

            AccessZone accessZone = new AccessZone();

accessZone.setId(CastTypeManager.getIntValue(resultSet.getString("r_access_zone_pk
ey")));

accessZone.setName(resultSet.getString("r_access_zone_description"));

            sectorInfo.setAccessZone(accessZone);
            sectorInfo.setSector(sector);
            sectorInfo.setFacility(facility);
            sectors.add(sectorInfo);
        }
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(resultSet, LOGGER);
    }

    return sectors;
}

@Override
public List<RestrictAZTInfo> getRestrictsAZT(RestrictAZTFilter
restrictAZTFilter) {
    String query =
sqb.setFunctionName("acs.get_rt_access_zone_turnstile").build(new
ComplexElement(restrictAZTFilter, RestrictAZTFilter.class));
    List<RestrictAZTInfo> restricts = new ArrayList<>();
    ResultSet resultSet = null;
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    try {

```

```

        resultSet = dbManager.executeQuery(query, LOGGER);
        while (resultSet.next()) {
            RestrictAZTInfo restrict = new RestrictAZTInfo();
            AccessZone accessZone = new AccessZone();
            accessZone.setId(resultSet.getInt("r_pkey_access_zone"));

            accessZone.setName(resultSet.getString("r_description_access_zone"));
            Turnstile turnstile = new Turnstile();
            turnstile.setId(resultSet.getInt("r_pkey_turnstile"));
            turnstile.setName(resultSet.getString("r_description_turnstile"));
            restrict.setAccessZone(accessZone);
            restrict.setTurnstile(turnstile);
            restricts.add(restrict);
        }
    } catch (SQLException e) {
        throw new ACSEException(e.getMessage(), e);
    } finally {
        dbManager.close(resultSet, LOGGER);
    }
    return restricts;
}

@Override
public List<TicketInfo> getTickets(TicketFilter ticketFilter) {
    String query = sqb.setFunctionName("acs.get_tickets").build(new
ComplexElement(ticketFilter, TicketFilter.class));
    List<TicketInfo> tickets = new ArrayList<>();
    DBManagerI dbManager = dbManagerFactory.getDBManager();
    ResultSet resultSet = null;
    try {
        resultSet = dbManager.executeQuery(query, LOGGER);
        while (resultSet.next()) {
            TicketInfo ticketInfo = new TicketInfo();
            Ticket ticket = new Ticket();
            ticket.setId(resultSet.getInt("r_pkey_ticket"));
            ticket.setCode(resultSet.getString("r_code"));
            ticket.setSeatRow(resultSet.getString("r_seat_row"));
            ticket.setSeatNumber(resultSet.getString("r_seat_number"));
            ticket.setPrice(Money.valueOf(resultSet.getLong("r_price")));

            ticket.setSellDate(resultSet.getTimestamp("r_ticket_sell_datetime"));

            ticket.setRefId(CastTypeManager.getIntValue(resultSet.getString("r_pkey_ref")));

            TicketInfo.TicketType ticketType = new TicketInfo.TicketType();
            ticketType.setId(resultSet.getInt("r_pkey_ticket_type"));

            ticketType.setName(resultSet.getString("r_description_ticket_type"));

            Seance seance = new Seance();
            seance.setId(resultSet.getInt("r_pkey_seance"));
            seance.setName(resultSet.getString("r_description_seance"));

```

```

        Sector sector = new Sector();

sector.setId(CastTypeManager.getIntValue(resultSet.getString("r_pkey_sector")));
        sector.setName(resultSet.getString("r_description_sector"));

        TicketInfo.TicketState ticketState = new TicketInfo.TicketState();
        ticketState.setId(resultSet.getInt("r_pkey_ticket_state"));

ticketState.setName(resultSet.getString("r_description_ticket_state"));


        ticketInfo.setSeance(seance);
        ticketInfo.setTicket(ticket);
        ticketInfo.setSector(sector);
        ticketInfo.setTicketState(ticketState);
        ticketInfo.setTicketType(ticketType);

        tickets.add(ticketInfo);

    }
} catch (SQLException e) {
    throw new ACSEException(e.getMessage(), e);
} finally {
    dbManager.close(resultSet, LOGGER);
}
return tickets;
}

```

```

package com.pb.irbis_dept.acs.server.rest.data_access;

```

```

import com.assembla.nosan.sqb.SQB;
import com.assembla.nosan.sqb.element.ComplexElement;
import com.pb.common.db.manager.DBManagerFactoryI;
import com.pb.common.db.manager.DBManagerI;
import com.pb.irbis_dept.acs.server.rest.containers.ValidateTicket;
import com.pb.irbis_dept.acs.server.rest.containers.ValidateTicketInfo;
import com.pb.irbis_dept.acs.server.rest.exception.ACSEException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

```

```

import java.sql.ResultSet;
import java.sql.SQLException;

```

```

public class Dao implements DaoI {

```

```

    private DBManagerFactoryI dbManagerFactory;
    private SQB sqb;
    final static Logger LOGGER = LoggerFactory.getLogger(Dao.class);

```

```

    public void setSqb(SQB sqb) {
        this.sqb = sqb;
    }

```

```

    }

    public void setDbManagerFactory(DBManagerFactoryI dbManagerFactory) {
        this.dbManagerFactory = dbManagerFactory;
    }

    @Override
    public ValidateTicketInfo validate(ValidateTicket validateTicket) {
        String query =
sqb.setCastType("acs.verifiable_data").setFunctionName("acs.validate_document").
        build(new ComplexElement(validateTicket, ValidateTicket.class));
        DBManagerI dbManager = dbManagerFactory.getDBManager();
        ResultSet rs = null;
        ValidateTicketInfo validateTicketInfo = new ValidateTicketInfo();
        try {
            rs = dbManager.executeQuery(query, LOGGER);
            while (rs.next()) {

validateTicketInfo.setFacilityName(rs.getString("r_facility_name"));
                validateTicketInfo.setSeanceName(rs.getString("r_seance_name"));
                validateTicketInfo.setSectorName(rs.getString("r_sector_name"));
                validateTicketInfo.setSeatRow(rs.getString("r_seat_row"));
                validateTicketInfo.setSeatNumber(rs.getString("r_seat_number"));
            }
            return validateTicketInfo;
        } catch (SQLException ex) {
            throw new ACSEException(ex.getMessage(), ex);
        } finally {
            dbManager.close(rs, LOGGER);
        }
    }

}

} package com.pb.irbis_dept.acs.server.rest.data_access;

import com.pb.irbis_dept.acs.server.rest.configuration.ServerConst;
import com.pb.irbis_dept.acs.server.rest.containers.UserInfo;
import com.pb.irbis_dept.acs.server.rest.exception.ACSEException;
import com.pb.irbisdept.common.utils.cache.CacheException;
import com.pb.irbisdept.common.utils.cache.CacheI;
import com.pb.irbisdept.common.utils.cache.CachePool;
import org.slf4j.MDC;

public class CacheManager implements CacheManagerI {

    private CachePool cachePool;
    private Integer sessionTime;

    public void setSessionTime(Integer sessionTime) {
        this.sessionTime = sessionTime;
    }

    public void setCachePool(CachePool cachePool) {

```



```

        this.cachePool = cachePool;
    }

    @Override
    public void storeUser(String session, UserInfo userInfo) {
        try {
            cachePool.getResource().set(session, sessionTime, userInfo);
        } catch (CacheException e) {
            throw new ACSEException(e.getMessage(), e);
        }
    }

    @Override
    public boolean isExistSession(String session) {
        try {
            CacheI resource = cachePool.getResource();
            boolean exist = resource.isExist(session);
            if (exist) {
                resource.expire(session, sessionTime);
            }
            return exist;
        } catch (CacheException e) {
            throw new ACSEException(e.getMessage(), e);
        }
    }

    @Override
    public UserInfo getUser() {
        String session = MDC.get(ServerConst.SESSION);
        try {
            return (UserInfo) cachePool.getResource().get(session);
        } catch (CacheException e) {
            throw new ACSEException(e.getMessage(), e);
        }
    }
}

package com.pb.irbis_dept.acs.server.rest.controller;

import com.pb.irbis_dept.acs.common.generate.*;
import com.pb.irbis_dept.acs.server.rest.configuration.ServerConst;
import com.pb.irbis_dept.acs.server.rest.containers.UserInfo;
import com.pb.irbis_dept.acs.server.rest.core.AdminCoreI;
import com.pb.irbis_dept.acs.server.rest.core.CoreI;
import com.pb.irbis_dept.acs.server.rest.data_access.CacheManagerI;
import com.pb.irbis_dept.acs.server.rest.exception.ACSEException;
import org.slf4j.MDC;

public class ServiceController implements ServiceControllerI {

    private CoreI core;
    private AdminCoreI adminCore;

```

```

private CacheManagerI cacheManager;

public void setCacheManager(CacheManagerI cacheManager) {
    this.cacheManager = cacheManager;
}

public void setCore(CoreI core) {
    this.core = core;
}

public void setAdminCore(AdminCoreI adminCore) {
    this.adminCore = adminCore;
}

@Override
public JAXBLoginResp login(JAXBLoginReq loginReq) {
    return adminCore.login(loginReq);
}

@Override
public JAXBValidateTicketResp validate(JAXBValidateTicketReq ticketReq) {
    validateSession(ticketReq);
    return core.validate(ticketReq);
}

@Override
public JAXBAddOuterTicketResp addOuterTicket(JAXBAddOuterTicketReq
addTicketReq) {
    validateSession(addTicketReq);
    return adminCore.addOuterTicket(addTicketReq);
}

@Override
public JAXBAddTurnstileResp addTurnstile(JAXBAddTurnstileReq addTurnstileReq)
{
    validateSession(addTurnstileReq);
    return adminCore.addTurnstile(addTurnstileReq);
}

@Override
public JAXBAccessZoneResp getAccessZones(JAXBAccessZoneReq accessZoneReq) {
    validateSession(accessZoneReq);
    return adminCore.getAccessZones(accessZoneReq);
}

@Override
public JAXBAddAccessZoneResp addAccessZone(JAXBAddAccessZoneReq
addAccessZoneReq) {
    validateSession(addAccessZoneReq);
    return adminCore.addAccessZone(addAccessZoneReq);
}

@Override

```

```

        public JAXBUpdateAccessZoneResp updateAccessZone(JAXBUpdateAccessZoneReq
updateAccessZoneReq) {
            validateSession(updateAccessZoneReq);
            return adminCore.updateAccessZone(updateAccessZoneReq);
        }

        @Override
        public void deleteAccessZone(JAXBDeleteAccessZoneReq deleteAccessZoneReq) {
            validateSession(deleteAccessZoneReq);
            adminCore.deleteAccessZone(deleteAccessZoneReq);
        }

        @Override
        public JAXBUpdateTurnstileResp updateTurnstile(JAXBUpdateTurnstileReq
jaxbUpdateTurnstileReq) {
            validateSession(jaxbUpdateTurnstileReq);
            return adminCore.updateTurnstile(jaxbUpdateTurnstileReq);
        }

        @Override
        public void deleteTurnstile(JAXBDeleteTurnstileReq jaxbDeleteTurnstileReq) {
            validateSession(jaxbDeleteTurnstileReq);
            adminCore.deleteTurnstile(jaxbDeleteTurnstileReq);
        }

        @Override
        public JAXBAddStructureResp addStructure(JAXBAddStructureReq
jaxbAddStructureReq) {
            validateSession(jaxbAddStructureReq);
            return adminCore.addStructure(jaxbAddStructureReq);
        }

        @Override
        public JAXBUpdateStructureResp updateStructure(JAXBUpdateStructureReq
jaxbUpdateStructureReq) {
            validateSession(jaxbUpdateStructureReq);
            return adminCore.updateStructure(jaxbUpdateStructureReq);
        }

        @Override
        public void deleteStructure(JAXBDeleteStructureReq jaxbDeleteStructureReq) {
            validateSession(jaxbDeleteStructureReq);
            adminCore.deleteStructure(jaxbDeleteStructureReq);
        }

        @Override
        public JAXBAddFacilityResp addFacility(JAXBAddFacilityReq jaxbAddFacilityReq)
{
            validateSession(jaxbAddFacilityReq);
            return adminCore.addFacility(jaxbAddFacilityReq);
        }

```

```

        @Override
        public JAXBUpdateFacilityResp updateFacility(JAXBUpdateFacilityReq
jaxbUpdateFacilityReq) {
            validateSession(jaxbUpdateFacilityReq);
            return adminCore.updateFacility(jaxbUpdateFacilityReq);
        }

        @Override
        public void deleteFacility(JAXBDeleteFacilityReq jaxbDeleteFacilityReq) {
            validateSession(jaxbDeleteFacilityReq);
            adminCore.deleteFacility(jaxbDeleteFacilityReq);
        }

        @Override
        public JAXBAddSeanceResp addSeance(JAXBAddSeanceReq jaxbAddSeanceReq) {
            validateSession(jaxbAddSeanceReq);
            return adminCore.addSeance(jaxbAddSeanceReq);
        }

        @Override
        public JAXBUpdateSeanceResp updateSeance(JAXBUpdateSeanceReq
jaxbUpdateSeanceReq) {
            validateSession(jaxbUpdateSeanceReq);
            return adminCore.updateSeance(jaxbUpdateSeanceReq);
        }

        @Override
        public void deleteSeance(JAXBDeleteSeanceReq jaxbDeleteSeanceReq) {
            validateSession(jaxbDeleteSeanceReq);
            adminCore.deleteSeance(jaxbDeleteSeanceReq);
        }

        @Override
        public JAXBAddSectorResp addSector(JAXBAddSectorReq jaxbAddSectorReq) {
            validateSession(jaxbAddSectorReq);
            return adminCore.addSector(jaxbAddSectorReq);
        }

        @Override
        public JAXBUpdateSectorResp updateSector(JAXBUpdateSectorReq
jaxbUpdateSectorReq) {
            validateSession(jaxbUpdateSectorReq);
            return adminCore.updateSector(jaxbUpdateSectorReq);
        }

        @Override
        public void deleteSector(JAXBDeleteSectorReq jaxbDeleteSectorReq) {
            validateSession(jaxbDeleteSectorReq);
            adminCore.deleteSector(jaxbDeleteSectorReq);
        }
    }

```

```

@Override
public JAXBAddTicketResp addTicket(JAXBAddTicketReq jaxbAddTicketReq) {
    validateSession(jaxbAddTicketReq);
    return adminCore.addTicket(jaxbAddTicketReq);
}

@Override
public JAXBUpdateTicketResp updateTicket(JAXBUpdateTicketReq
jaxbUpdateTicketReq) {
    validateSession(jaxbUpdateTicketReq);
    return adminCore.updateTicket(jaxbUpdateTicketReq);
}

@Override
public void deleteTicket(JAXBDeleteTicketReq jaxbDeleteTicketReq) {
    validateSession(jaxbDeleteTicketReq);
    adminCore.deleteTicket(jaxbDeleteTicketReq);
}

@Override
public void addRestrictAZT(JAXBAddRTAccessZoneTurnstileReq
jaxbAddRTAccessZoneTurnstileReq) {
    validateSession(jaxbAddRTAccessZoneTurnstileReq);
    adminCore.addRestrictAZT(jaxbAddRTAccessZoneTurnstileReq);
}

@Override
public void deleteRestrictAZT(JAXBDeleteRTAccessZoneTurnstileReq
jaxbDeleteRTAccessZoneTurnstileReq) {
    validateSession(jaxbDeleteRTAccessZoneTurnstileReq);
    adminCore.deleteRestrictAZT(jaxbDeleteRTAccessZoneTurnstileReq);
}

@Override
public JAXBStructureResp getStructures(JAXBStructureReq jaxbStructureReq) {
    validateSession(jaxbStructureReq);
    return adminCore.getStructures(jaxbStructureReq);
}

@Override
public JAXBFacilityResp getFacilities(JAXBFacilityReq jaxbFacilityReq) {
    validateSession(jaxbFacilityReq);
    return adminCore.getFacilities(jaxbFacilityReq);
}

@Override
public JAXBTurnstileResp getTurnstiles(JAXBTurnstileReq jaxbTurnstileReq) {
    validateSession(jaxbTurnstileReq);
    return adminCore.getTurnstiles(jaxbTurnstileReq);
}

@Override

```

```

    public JAXBSeanceResp getSeances(JAXBSeanceReq jaxbSeanceReq) {
        validateSession(jaxbSeanceReq);
        return adminCore.getSeances(jaxbSeanceReq);
    }

    @Override
    public JAXBSectorResp getSectors(JAXBSectorReq jaxbSectorReq) {
        validateSession(jaxbSectorReq);
        return adminCore.getSector(jaxbSectorReq);
    }

    @Override
    public JAXBRTAccessZoneTurnstileResp
getRestrictsAZT(JAXBRTAccessZoneTurnstileReq jaxbrtAccessZoneTurnstileReq) {
        validateSession(jaxbrtAccessZoneTurnstileReq);
        return adminCore.getRestrictsAZT(jaxbrtAccessZoneTurnstileReq);
    }

    @Override
    public JAXBTicketResp getTickets(JAXBTicketReq jaxbTicketReq) {
        validateSession(jaxbTicketReq);
        return adminCore.getTickets(jaxbTicketReq);
    }

    private void validateSession(JAXBacsRequest acsRequest) {
        boolean b = cacheManager.isExistSession(acsRequest.getSessionId());
        if (!b) {
            throw new ACSEException("session is not valid");
        }
        initMDCContext(acsRequest);
    }

    private void initMDCContext(JAXBacsRequest acsRequest) {
        MDC.put(ServerConst.SESSION, acsRequest.getSessionId());
        UserInfo user = cacheManager.getUser();
        MDC.put(ServerConst.USER_ID, String.valueOf(user.getId()));
        MDC.put(ServerConst.USER_NAME, user.getLogin());
    }
}

```