

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.

" ____ " _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "бакалавр"

на тему: Розробка веб-сервісу поштового відділення.

Виконав: студента групи

_____ *Кошулько В.В.*
(підпис)

Керівник роботи:

(посада, науковий ступінь вчене звання)
_____ *Беркунський Є.Ю.*
(підпис)

м. Миколаїв – 2022 р.

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 122 "Комп'ютерні науки"
Освітня програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми

к.т.н, доц. Гайда А.Ю.

" ____ " _____ 2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття ступеня вищої освіти "бакалавр"

Студента Кошулько Владислава Валерійовича _____
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка веб-сервісу поштового відділення _____
Керівник роботи: Беркунський Євген Юрієвич _____
Затверджені наказом ректора № 269-уч від "18" травня 2022 року.
2. Термін подання роботи: 20 червня 2022 р.
3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем.
4. Перелік питань, що належать до розробки (найменування розділів): розробка концепції автоматизованого робочого місця оператора поштового відділення, розробка технічного проекту арм, реалізація проекту арм, охорона праці та техніка безпеки
5. Перелік презентаційних матеріалів: Розробка автоматизованого робочого місця оператора поштового відділення, робота за посилками, концепція інформаційної системи відділення, модель АРМ оператора поштового відділення, концептуальна модель БД, логічна модель БД та фізична модель БД, використані інструменти та технології, реалізований проект.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Беркунський Є.Ю.	Лютий 2022	Лютий 2022
2	Беркунський Є.Ю.	Березень 2022	Березень 2022
3	Беркунський Є.Ю.	Квітень 2022	Квітень 2022
4	Гурець Н.В.	Травень 2022	Травень 2022

7. Дата видачі завдання 14 лютого 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	Лютий 2022	
2	Розробка концепції автоматизованого робочого місця оператора поштового відділення	Лютий 2022	
3	Розробка технічного проекту арм	Березень 2022	
4	Реалізація проекту арм	Квітень 2022	
5	Охорона праці та техніка безпеки	Травень 2022	
6	Затвердження теми дипломної роботи	18 травня 2022	
7	Подача роботи на анти плагіат	13 червня 2022	
8	Затвердження дипломної роботи керівником	18 червня 2022	
9	Доповідь та презентація	19 червня 2022	
10	Захист дипломної роботи	27 червня 2022	

Студент: Кошулько В.В.

(підпис)

(ПІБ)

Керівник роботи: Беркунський Є.Ю.

(підпис)

(ПІБ)

АНОТАЦІЯ

Дипломний проект виконується для покращення навичок проектування, аналізу а також розробки інформаційних систем. Розробляється автоматизоване робоче місце для оператора поштового відділення, який займається прийомом, видачою, зберіганням та відправленням посилок у інші відділення.

Робота викладена на 116 сторінках друкованого тексту, містить 4 розділі, 17 рисунки, 9 таблиць, 3 додатки та список використаної літератури.

Робота виконана українською мовою.

ABSTRACT

The diploma project is performed to improve the skills of design, analysis and development of information systems. An automated workstation is being developed for the post office operator, which receives, dispenses, stores and sends parcels to other offices.

The work is presented on 116 pages of printed text, contains 4 chapters, 17 figures, 9 tables, 3 appendices and a list of references.

The work is performed in Ukrainian.

ЗМІСТ

ВСТУП.....	7
КЛЮЧОВІ СЛОВА.....	8
РОЗДІЛ 1. РОЗРОБКА КОНЦЕПЦІЇ АВТОМАТИЗОВАНОГО РОБОЧОГО МІСЦЯ ОПЕРАТОРА ПОШТОВОГО ВІДДІЛЕННЯ	10
1.1.Об’єкт дослідження і аналіз інформаційних процесів.....	10
1.2.Формування вимог до АРМ	14
1.3.Розробка концепції АРМ	16
РОЗДІЛ 2. РОЗРОБКА ТЕХНІЧНОГО ПРОЕКТУ АРМ	23
2.1.Загальносистемні рішення.....	23
2.2.Рішення по інформаційному забезпеченню.....	35
2.3.Рішення по програмному забезпеченню.....	45
2.4.Рішення по технічному забезпеченню.....	50
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОЕКТУ АРМ	52
РОЗДІЛ 4. ОХОРОНА ПРАЦІ ТА ТЕХНІКА БЕЗПЕКИ.....	61
4.1.Аналіз небезпечних і шкідливих факторів при роботі з посилками.....	66
4.2.Розробка інструкції для забезпечення сприятливих умов праці при роботі з посилками.....	66
ВИСНОВОК.....	74
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	75
ДОДАТКИ.....	77
Додаток А – Технічне завдання	
Додаток Б – Методика тестування та випробування	
Додаток В – Код програми та створення БД	

ВСТУП

Тема дипломного проекту – “Розробка автоматизованого робочого місця оператора поштового відділення”. Актуальність у розробці АРМ полягає в тому, що буде спроектовано інформаційну систему, яка дозволить підвищити ефективність вирішення задач поставлених перед оператором поштового відділення.

Мета – підвищення ефективності прийому та видачі посилок оператором поштового відділення за рахунок поліпшення та прискорення рутинних операцій оформленням накладних посилок, швидкої видачі, перевезення та зберігання посилок, також за допомогою ефективного зберігання даних, аналіз стану інформаційних процесів на відділеннях, відстежування кожної посилки та доступ до її історії перевезень. Завдання полягає в розробці програмних засобів і створенні інформаційної системи.

Об'єктом дослідження – є поштове відділення у якому працює оператор та обслуговує клієнтів. Предмет дослідження – інформаційні процеси, які реалізуються оператором при видачі, прийому та перевезення посилок.

На основі проведеного аналізу стану діючої системи для оператора поштового відділення, сформовано вимоги до проекту автоматизованого робочого місця оператора поштового відділення.

Виходячи з запропонованих покращень та сформованих вимог, які можна впровадити у поштовому відділенні, розроблено проект системи для оператора поштового відділення. Розроблено концепцію АРМ, технічний проект, проведено дослідження покращення умов праці та техніки безпеки. Для реалізації системи, базуючись на розробленому технічному проекті, були визначені сучасні програмні та технічні засоби. Реалізовано проект АРМ, який дозволяє вирішити поставлені задачі, такі як: формування накладної посилки,

відстеження перевезень посилки, облік посилок на складі, видача посилок клієнту та формування звітів.

КЛЮЧОВІ СЛОВА

БД - База даних, СКБД - система керування/управління базами даних,
АРМ - автоматизоване робоче місце, Hibernate, Spring Security, MVC - Модель-
вигляд-контролер.

РОЗДІЛ 1. РОЗРОБКА КОНЦЕПЦІЇ АВТОМАТИЗОВАНОГО РОБОЧОГО МІСЦЯ ОПЕРАТОРА ПОШТОВОГО ВІДДІЛЕННЯ

1.1. Об'єкт дослідження і аналіз інформаційних процесів

Сьогодні у світі доставка товарів у пункти видачі або відділення, які знаходяться неподалеку від місця проживання людей, або навіть додому, є одною з необхідних логістичних умов населеного пункту, розвитку бізнеса, підприємництва та необхідністю для звичайних жителів.

Основними критеріями для пришвидшення оформлення та отримання посилок є час на формування накладної та видачу посилки, опрацювання відправлень та приїзду машини з посилками. Посилки бувають різного розміру, властивостей та структури, тому найшвидше формування накладних, видача посилок та відправлення і приймання машини з посилками є основою для масштабування кількості відділень та розростання інфраструктури.

Оптимізація оформлення та доставки поділяється на декілька пунктів:

- Оформлення накладної посилки;
- Формування накладної доставки посилок до іншого відділення;
- Формування звіту за прибуттям вантажівки з посилками;
- Звіт про отримання посилки клієнтом;

Оформлення накладних на посилки може відбуватись двома способами:

- Оформлення накладних на посилки онлайн
- Оформлення накладних на посилки оператором і відділенні

За нашою предметною областю нас більше цікавить пункт формування накладних на посилки оператором відділення. Щоб це зробити оператор запросить такі данні про відправника як: ім'я, прізвище, по-батькові, номер

телефону; дані про отримувача: ім'я, прізвище, по-батькові, номер телефону, адреса та номер відділення отримувача; дані про посилку: тип посилки, кількість однотипних посилок за одною накладною; та виміряє габарити посилки: вага та об'єм посилок.

Формування накладної доставки посилок до іншого відділення виконується тільки оператором, йому навіть не потрібно вказувати які посилки відправляти, так як вони вже є у базі, оператору потрібно лише сформувати накладний лист відправлення та переконатись що усі посилки були згружені у вантажівку, після цього вантажівка відбуває а у системі відправленні посилки маркіруються відправленими до наступного відділення.

Формування звіту за прибуттям вантажівки з посилками також відбувається тільки оператором, він формує звіт про прибуття вантажівки та у кожної посилки відбувається оновлення історії їх маршруту між відділеннями, після цього оператор може вивантажити посилки з вантажівки. Реєстрація пошкоджень посилок буде передбачено у наступних версіях АРМ оператора відділення.

Звіт про отримання посилки клієнтом. Клієнт приходить у відділення та показує номер накладної або/та з документом який підтверджує отримувача, після цього оператор відділення може віддати посилку якщо клієнт згоден забрати її та оператор може зафіксувати отримання посилки і накладна буде закрита, якщо клієнт не згоден забрати посилку, то оператор відпускає клієнта і одразу за цією накладною у системі може сформувати зворотну доставку. У наступних версіях розробки буде передбачено встановлення сплати за доставку та страхування посилок.

На кожному з цих етапів відбувається оновлення історії посилок, оновлюється час та дата у якому відділенні посилка знаходиться, оновлюється адреса та номер відділення, якщо вантажівка перевезла посилки у інше

відділення, ці дані також оновлюються після закриття зміни та відкриття зміни оператором. Тобто, відстежування посилок відбувається постійно і якщо посилка десь загубиться то можна чітко побачити на якому етапі вона загубилась та знайти її, таким чином зменшується вірогідність загублення посилки та вірогідність відшукати загублену посилку збільшуються.

Проведемо дослідження робіт якими займається оператор поштового відділення. Основний тип його зайнятості – це оформлення посилок, видача їх, формування маршрутного листа на відправлення та прибуття вантажівки з посилками. Усі роботи виконуються лише оператором. В роботі оператор виконує такі функції:

- Оформляє накладні посилок на відправлення і інші відділення, спілкується з клієнтом та отримання від нього даних необхідних для формування накладної;
- Займається відправленням посилок до інших відділень та формуванням маршрутного листа для відправлення;
- Виміряє розміри посилки, довжину, висоту та ширину далі виводить об'єм посилки, а також зважує її щоб дізнатись вагу посилки та ввести до накладної.
- Отримує від клієнта данні про адресу кінцевого відділення.
- Оформлює прибуття вантажівки з посилками.

При формуванні накладної на посилку використовуються представленні нижче документи і форми.

- Заказ клієнта. Містить в собі: дані клієнта відправника та його номер телефону, дату та час прийняття посилки, данні відділення у якому прийняли

посилку, дані отримувача та його номер телефону, кінцеве відділення отримувача та габарити посилки такі як: об'єм та вага. Також при формуванні накладної, накладна - це гарантійний лист що посилка прийнята та буде доставлена до вказаного відділення.

Номер накладной: 79873912836321	
Отправитель:	Получатель:
ФИО: Иванов Иван Иванович	ФИО: Виктор Виктор Викторович
Номер тел.: +380123456789	Номер тел.: +380987654321
Адресс отделения: г. Киев, 56 отделение	Адресс отделения: г. Львов, 65 отделение
Посилка:	
Вес: 1,56 кг	Тип: Электроника
Объем: 0,4 л	Калличество: 2

Рисунок. 1.4. – Приклад накладної посилки.

Після всього, визначили інформаційні потоки, сильні та слабкі сторони доставки.

Оператор потребує автоматизованого робочого місця для вирішення задач, котрі на даний момент не всі автоматизовані та потребують багато часу. Впровадження АРМ поліпшить документообіг, автоматизує звіти та буде вести облік та історії перевезень посилок, їх надходжень у відділення та відбуття до інших відділень. Це сприятиме меншим витратам часу роботи оператора.

1.2. Формування вимог АРМ

Проектоване АРМ призначене для оператора відділення доставки посилок. АРМ повинно вирішувати досить вузьке коло завдань. Основним завданням, що стоїть перед проектованим АРМ є облік та відстежування посилок. Для спрощення роботи з базою даних, формування звіту, працівник бажає використовувати веб-додаток.

В результаті формування вимог були виділені наступні можливості якими повинна володіти проектоване автоматизоване робоче місце:

- Формування накладних;
- Відстеження посилки, звідки вона прибули та де вона зберігається;
- Видача посилок клієнту;
- Облік посилок, зі збереженням усієї історії;
- Зберігати інформацію о клієнті в системі;
- Зберігати інформацію о працівниках;
- Зберігати інформацію о водіях вантажівок;
- Зберігати інформацію о посылках;
- Зберігати інформацію о кількості однотипних посилок на одну накладну;
- Зберігати інформацію о перевезеннях посилок;
- Можливість змінити або уточнити дані про посилку;
- Зберігати інформацію о розташуванні поштових відділень;
- Подивитись поточні посилки у відділенні;
- Облік посилок які відправлено назад відправнику;
- Безпека даних;
- Зручний user – friendly інтерфейс АРМ;
- Помилко–стійкість;

User – friendly – визначення інтерфейсу, дружнього до користувача в теорії юзабіліті. Цим словосполученням зазвичай позначають середу (в тому числі, сайту та сервісу), продуману із врахуванням зручності для користувача. Ця поняття має наступні ознаки: швидке завантаження, привабливість, мінімалізм, інтуїтивність, демонстрація всіх потрібних функцій відразу, вміння спілкуватися з користувачем та передбачуваність [3].

Безпека даних – це захист даних від ненавмисного або навмисного порушення секретності, руйнування або спотворення даних.

До технічних засобів припадають всі апаратні способи які необхідні для звичайного функціонування АРМ. До них належить:

1. Апаратне забезпечення клієнтського робочого місця;
2. Апаратне забезпечення серверного елементу.

В наявності у оператора є достатньо потужний комп'ютер, але він застосовує його виключно в цілях роботи з посилками та відділенням. В вимогах є розроблення веб – додатку, то для використання АРМ відділення може застосовувати будь – який новітній пристрій з підключенням до мережі INTERNET. Для системної частини проєктованого АРМ можна буде закупити обладнання, або потрібно користуватися хмарними технологіями на кшталт AWS.

1.3. Розробка концепції

На основі аналізу предметної області, вимог, інформаційних потреб, виявлення інформаційних об'єктів і зв'язків між ними, визначили існуючі технічні та програмні засоби, сформулювали вимоги до АРМ оператора відділення. В результаті була розроблена концепція інформаційної системи, яка представлена на рисунку 1.1.

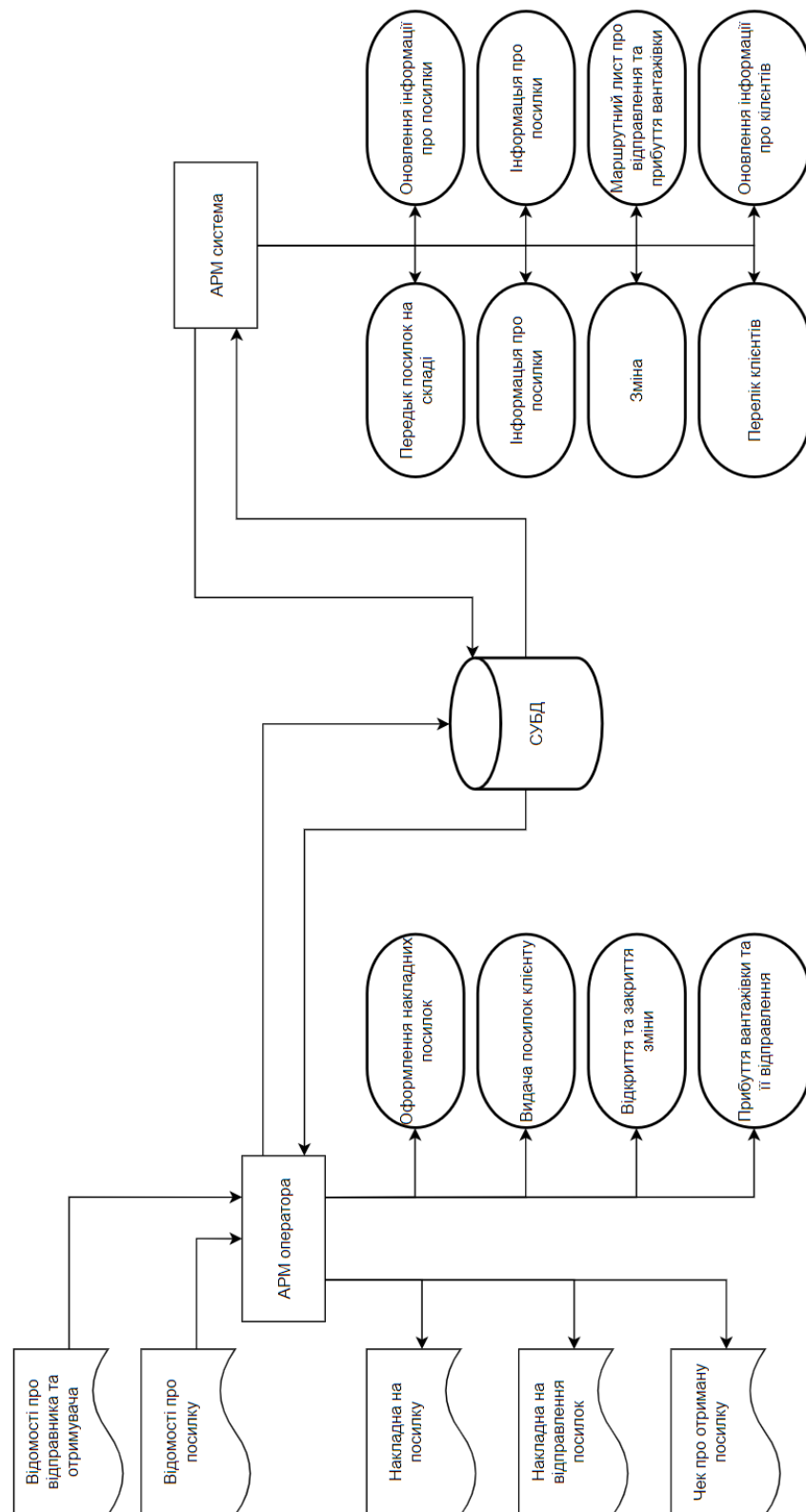


Рисунок. 1.1. – Концепція інформаційної системи відділення.

Проаналізувавши розроблену концепцію можна виокремити деякі елементи:

- Виконане АРМ – операторський блок;
- Проектоване АРМ – системний блок;
- Система керування базами даних;
- Програмні засоби.
- Технічні засоби;

Автоматизоване робоче місце оператора поштового відділення

Інженерний блок містить програмне забезпечення, яке дозволяє повністю здійснити формування накладної, формування маршрутного листа, зареєструвати прибуття вантажівки з посилками, та зафіксувати видачу посилки клієнту. З отриманих маршрутних листів – створити історії перевезень, які формують маршрут переміщення кожної посилки. Це дозволяє мінімізувати вірогідність загублення посилки або її неврахування.

Проектований АРМ містить системний блок завдань. Цей блок дозволить оператору відділення не гаяти час на облік, який можна автоматизувати за допомогою машини.

- Сформувати накладну – дозволяє отримати документ “Накладна посилки”.
- Перелік посилок, їх наявність на складі, їх історію та клієнтів, – виводи на екран наявну інформацію у БД на кожну накладну, наприклад поточні посилки на складі.
- Зміни відкриття відділення та закриття його, тим самим ми фіксуємо поточні посилки на складі.

- Зберігання даних про клієнтів, що дозволить прискорити формування накладних.
- Оновлення інформації про посилки та клієнтів.

Програмні засоби

На теперішній час мова Java є одним з найбільш популярних та розповсюджених мов програмування. Java – універсальна мова програмування, яка застосовується для вирішення багатоманітного походження завдань. За весь період існування переросла із простої універсальної мови у повну екосистему та платформу, яка єднає різні технології, які застосовуються для широкого спектру задач: від написання великих веб-порталів і сервісів до створення десктопних додатків. Ще мова Java завзято використовується для створення програмного забезпечення для неймовірної кількості різних пристроїв: побутової техніки, ПК, смартфонів, планшетів.

На поточний час величезні програмні продукти розроблюються за допомогою IDE(Integrated Development Environment) – інтегрованих серед розробки. Ці програми облегшують та пришвидшують створення програмних продуктів. На цей час найуніверсальнішою, найпопулярнішою і найкращою IDE для Java є IntelliJ IDEA від компанії JetBrains. Так як оператору потрібен web – додаток, нам потребується IntelliJ IDEA Ultimate, яка підтримує Spring Boot/MVC або JAVA EE.

Spring MVC – це веб – фреймворк Spring. Він дає змогу створювати RESTful сервіси (наприклад, JSON / XML) або інші веб – продукти і добре інтегрується в екосистему Spring, та підтримує контролери і REST контролери в Spring Boot додатках.

Spring Boot – це дуже корисний проект, ціллю якого є полегшення створення додатків на базі Spring та дозволяє простим способом створити web – продукти та web – додатки, потребуючи від розробника найменших зусиль за його налаштуванням та написання коду. Властивість Spring Boot, яка буде потрібна, це інтегрований Tomcat сервер.

Система керування базами даних

Обрання системи керування базами даних (СКБД) являє собою складну багато-параметричну задачу та є одним з необхідних стадій при розробці проекту, що працює з базою даних.

Для проектного АРМ оператора поштового відділення, ґрунтуючись на викладених вище вимогах, в якості СКБД нашої розроблюваної системи вибраний PostgreSQL – це вольна об'єктно-реляційна СКБД. PostgreSQL ґрунтується на мові SQL та підтримує доволі багато можливостей та стандартів SQL: 2011. На міркування більшості це надзвичайно стабільна СКБД, у ній практично неможливо зламати таблиці і знищити дані та неможливо покласти. Це також грамотна і стара СКБД. Вона приблизно як MySQL та тільки краще, але потрібно уміти її налаштувати. Важливими чинниками при обранні PostgreSQL – це гнучкість у використанні, надійність, вкрай структуровані дані та легке масштабування і розширення. За допомогою сторонніх засобів та бібліотек дуже зручно та легко розширювати та працювати з БД.

Технічні засоби

Для віддаленого застосовування системного блоку АРМ слід буде купити обладнання для сервера з щонайменшими технічними характеристиками не менше за вказані:

1. Процесор – Intel Core i3 3220 і більше;
2. Оперативна пам'ять – 8 Gb;
3. Мережева карта – не менше ніж 100Мбіт;
4. SSD диск з місцем не менше ніж 120 Гбайт;
5. Монітор з роздільною здатністю 1280*720 і більше.

Робоче місце варто обладнати чорно-білим або кольоровим принтером для друкування звітів.

На базі розробленої концепції буде реалізовано технічний проект системи АРМ оператора поштового відділення, який дозволить збільшити ефективність прийому та видачі посилок, загалом системної частини, за рахунок доцільного використання ресурсів та часу.

РОЗДІЛ 2. РОЗРОБКА ТЕХНІЧНОГО ПРОЕКТУ АРМ ОПЕРАТОРА ПОШТОВОГО ВІДДІЛЕННЯ

2.1. Загальносистемні рішення

Автоматизоване робоче місце оператора поштового відділення репрезентує собою “Системний блок” призначений для застосовування оператором відділення, котрий забезпечує формування накладних посилки та накладних маршруту, облік посилок, облік змін оператора, перелік наявних посилок на складі, облік клієнтів, звітність про отримані та видані посилки.

Згідно з сформованою концепцією та вимогами, розроблена модель використання АРМ оператора поштового відділення. Оператор застосовує web – інтерфейс браузера для роботи з логікою та базою даних. АРМ застосовує методики браузера для відображення інформації на монітор користувача.

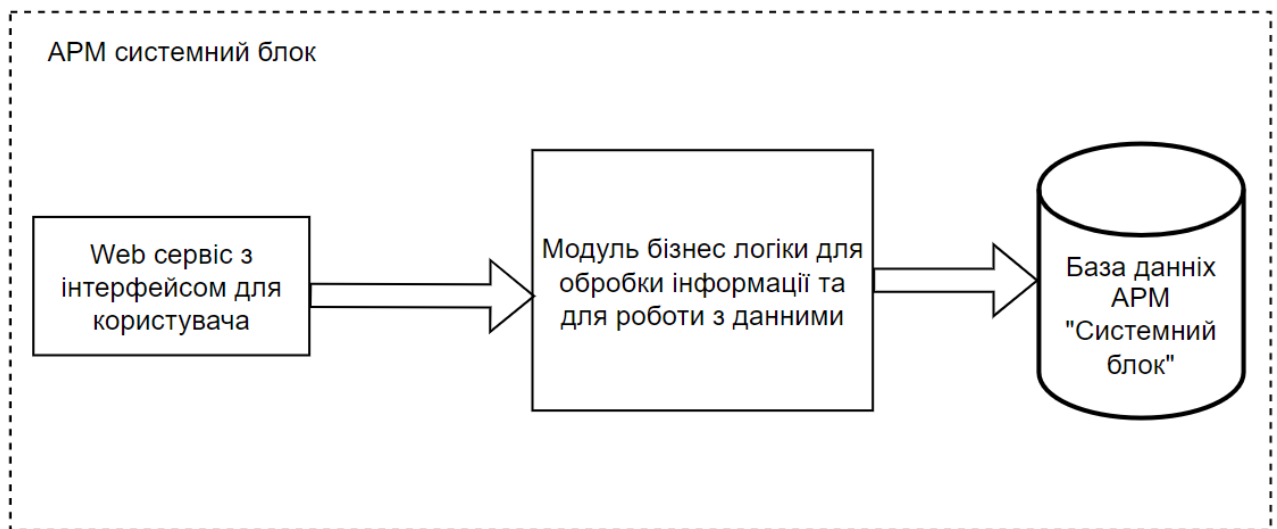


Рисунок 2.1. Модель АРМ оператора поштового відділення.

АРМ оператора виконує наступні функції:

- ❖ Формує накладну на посилку;
- ❖ Відстежує посилку по відділенням;
- ❖ Відстежує фурнітуру;
- ❖ Веде облік посилок на складі;

- ❖ Формує звіти про отримання посилки відділенням, про отримання посилки клієнтом, звіти про відправлення при прибуття вантажівки з посилками;

- ❖ Зберігає інформацію о клієнтах в системі;
- ❖ Зберігає інформацію о працівниках;
- ❖ Зберігає інформацію про прибуття вантажівки;
- ❖ Зберігає інформацію про відправлення вантажівки;
- ❖ Зберігає інформацію об отриманні клієнтом посилки;
- ❖ Зберігає інформацію об отриманні поштовим відділенням посилки від клієнта;

- ❖ Зберігає інформацію о адресу розташування поштових відділень;

Система керування базами даних у цьому АРМ застосовується для сполучення бізнес логіки та БД. СКБД приймає запити такі як: додавання, видаленню або редагуванню записів, відправляє відповіді на запити, займається функціонуванням по формуванню таблиць, а також контролює правильність даних. АРМ оператора для роботи з даними реалізує CRUD операції.

CRUD – акронім, який розшифровується чотирма базовими функціями, які застосовуються при роботі з базаю даних: створення, зчитування, модифікація та видалення. Введено Мартіном Джеймсом (англ. Martin James) в 1983 році як стандартизовану класифікацію функцій з маніпуляцією даними [11]. Ці операції є стандартними і виконуються на базі СКБД. У нашому випадку ми використовуємо PostgreSQL та вона теж підтримує дані операції.

Для роботи із системою варто розробити графічний інтерфейс користувача, котрий застосовує браузер для демонстрації операцій. Сторінки зобов'язані надавати графічне зображення даних які зберігаються у БД. Для CRUD операцій реалізувати поля. Мати динамічне меню у верхньому фрагменті екрану та основну панель після входу до системи.

Головна панель:

1. Вхід у систему оператором;
2. Відкриття або закриття зміни
 - 2.1. Відкриття зміни
 - 2.2. Закриття зміни
3. Посилки на складі;
 - 3.1. Відобразити посилки на складі;
 - 3.2. Видати посилку;
 - 3.3. Відправити посилку назад;
4. Відправлення та прибуття вантажівки;
 - 4.1. Прибуття вантажівки;
 - 4.2. Відправлення вантажівки;
5. Вихід з системи.

UML – мова графічного опису для об'єктного моделювання в області розроблення програмного забезпечення. UML є мовою широкого профілю, який являє собою відкритий стандарт, котрий застосовує графічні позначення для створення загальних моделей системи, так зване UML-моделлю. UML створений для визначення, проектування, документування та візуалізації, в основному для програмних систем [10]. З його допомогою даного засобу дозволено створювати різні діаграми. Для нашої задачі на стадії розробки проектного документання буде застосовуватись один з видів – це діаграма варіантів використання (Use – case diagram). Модель варіантів використання розроблена на базі стандартизованої мови UML та представлена на рисунку 2.2.



Рисунок 2.2. Діаграма прецедентів.

Приклади використання варіантів АРМ приведені у таблицях 2.1 – 2.6.

Таблиця 2.1. Сформувати накладну посилки.

Для чого використовується	Використовується для формування накладної посилки.
Діюча особа	Оператор відділення
Передумова	Клієнт виражає бажання відправити посилку.

На що реагує	Натискання на пункт сформувати накладну.
Сценарій	Оператор натискає сформувати накладну, заповнює ПІБ клієнта відправника та його номер телефону, заповнює ПІБ клієнта отримувача, його номер телефону та адресу відділення отримувача, а також номер відділення, вводить вагу та об'єм посилки. Натискає сформувати.
Що відбудеться	Формування накладної посилки, в її історію автоматично записується дата та час створення, відділення у якому прийняли, та ставиться статус «Посилка готова до перевезення» та усі данні зберігаються у БД.

Таблиця 2.2. Сформувати накладну відправлення вантажівки з посилками.

Для чого використовується	Використовується для формування маршрутного листа на відправлення вантажівки з посилками у інші відділення з поточного відділення.
Діюча особа	Оператор відділення
Передумова	Необхідно завантажити вантажівку посилками для відправлення у інше відділення
На що реагує	Натискання на пункт відправити вантажівку.
Сценарій	Оператор натискає на пункт відправити вантажівку, виводиться панель з переліком посилок які необхідно завантажити у вантажівку, після

	завантаження оператор перевіряє чи всі посилки завантажені і натискає сформувати.
Що відбудеться	Формування маршрутного листу відправлення вантажівки з посилками, система оновлює історію посилок автоматично де у цю історію записується дата та час відправлення, відділення з якого відправили, та ставиться статус «Посилка в дорозі» та усі данні збережуться у БД.

Таблиця 2.3. Сформувати накладну прибуття вантажівки з посилками.

Для чого використовується	Використовується для формування маршрутного листа о прибутті вантажівки з посилками з іншого відділення у поточного відділення.
Діюча особа	Оператор відділення
Передумова	Необхідно вивантажити вантажівку посилками для прийняття вантажівки з посилками з іншого відділення
На що реагує	Натискання на пункт прибуття вантажівки.
Сценарій	Оператор натискає на пункт прибуття вантажівки, виводиться панель з переліком посилок які необхідно вивантажити з вантажівки, після вивантаження оператор перевіряє чи всі посилки вивантажені і натискає сформувати.
Що відбудеться	Формування маршрутного листу прибуття вантажівки з посилками, система оновлює історію

	посилок автоматично де у цю історію записується дата та час прибуття, відділення у яке прибула вантажівка, та ставиться статус «Посилка у відділенні» та усі данні збережуться у БД.
--	--

Таблиця 2.4. Відкриття зміни оператором відділення.

Для чого використовується	Використовується для відкриття зміни оператором відділення.
Діюча особа	Оператор відділення.
Передумова	Оператору відділення необхідно відкрити змінну для початку робочого дня.
На що реагує	Натискання на пункт відкрити зміну.
Сценарій	Оператор відділення входить у систему під власним логіном та паролем, вибирає пункт «Відкрити зміну» далі відкривається панель де потрібно ввести усі посилки на складі, на натиснути відкрити змінну.
Що відбудеться	Система формує відкриття зміни та оновлює історію посилки де у цю історію записується дата та час відкриття зміни, відділення у якому відкрили зміну, та ставиться статус «Змінна відкрита: Посилка на складі» та усі данні збережуться у БД.

Таблиця 2.5. Відкриття зміни оператором відділення.

Для чого використовується	Використовується для відкриття зміни оператором відділення.
---------------------------	---

Діюча особа	Оператор відділення.
Передумова	Оператору відділення необхідно відкрити змінну для початку робочого дня.
На що реагує	Натискання на пункт відкрити зміну.
Сценарій	Оператор відділення входить у систему під власним логіном та паролем, вибирає пункт «Відкрити зміну» далі відкривається панель де потрібно ввести усі посилки на складі, на натиснути відкрити змінну.
Що відбудеться	Система формує відкриття зміни та оновлює історію посилки де у цю історію записується дата та час відкриття зміни, відділення у якому відкрили зміну, та ставиться статус «Змінна відкрита: Посилка на складі» та усі данні зберігаються у БД.

Таблиця 2.6. Закриття зміни оператором відділення.

Для чого використовується	Використовується для закриття зміни оператором відділення.
Діюча особа	Оператор відділення.
Передумова	Оператору відділення необхідно закриття змінну для закінчення робочого дня.
На що реагує	Натискання на пункт закриття зміну.
Сценарій	Оператор відділення вибирає пункт «Закрити зміну» далі відкривається панель де потрібно ввести усі посилки на складі, на натиснути закрити змінну.

Що відбудеться	Система формує закриття зміни та оновлює історію посилок де у цю історію записується дата та час закриття зміни, відділення у якому закрили зміну, та ставиться статус «Змінна закрита: Посилка на складі» та усі данні зберігаються у БД.
-------------------	--

Таблиця 2.7. Отримання посилки клієнтом та закриття накладної.

Для чого використовується	Використовується для фіксування отримання клієнтом посилки.
Діюча особа	Оператор відділення.
Передумова	Клієнт виражає бажання отримати посилку у відділенні.
На що реагує	Закрити накладну.
Сценарій	Оператор відділення перевіряє чи є така посилка з таким номером накладної на складі та вибирає пункт «Закрити накладну» та віддає посилку клієнту отримувачу.
Що відбудеться	Система формує закриття накладної та оновлює історію посилок де у цю історію записується дата та час закриття зміни, відділення у якому закрили зміну, та ставиться статус «Змінна закрита: Посилка на складі» та усі данні зберігаються у БД.

Таблиця 2.8. Зворотне відправлення посилки клієнтом отримувачем
якому посилка не підійшла.

Для чого використовується	Використовується для формування зворотної накладної посилки.
Діюча особа	Оператор відділення.
Передумова	Клієнт виражає бажання відмовитись отримати посилку.
На що реагує	Сформувати зворотну накладну.
Сценарій	Оператор відділення натискає на пункт сформувати зворотну накладну на посилку та перевіряє чи не пошкодив клієнт її при огляді а далі формується зворотна накладна на посилку.
Що відбудеться	Система формує зворотну накладну на посилку де на основі історії цієї посилки формується нова накладна у якій відділення відправлення буде поточне відділення а відділення отримувача буде відділення відправлення з накладної посилки, далі отримувачем стає відправник а відправником залишається відправник з накладної, усі інші дані переносяться у зворотну накладну та на накладну ставиться статус «Зворотна доставка» та усі данні заносяться у історію нової накладної та зберігаються у БД.

Таблиця 2.9. Показати поточні посилки на складі відділення

Для чого використовується	Використовується для відображення інформації о посилках у відділенні.
---------------------------	---

Діюча особа	Оператор відділення.
Передумова	Клієнт виражає бажання дізнатись інформацію про прибуття його посилки або оператор виражає бажання переглянути наявні на складі посилки.
На що реагує	Сформувати зворотну накладну.
Сценарій	Оператор відділення натискає на пункт посилки на складі.
Що відбудеться	Система формує список поточних на складі посилок та виводить їх на інтерфейс користувача.

В системі слід реалізувати безпеку інформації. Для даних, які розташовані в осередку АРМ доступ зобов'язаний мати тільки оператор поштового відділення. Для запобігання витоку даних та забезпечення інформаційної безпеки повинен бути запроваджений засіб захисту даних. Дані котрі циркулюють та зберігаються у середині АРМ призначені тільки для оператора та несуть деяку цінність, як предмет інформаційної власності. Попри витоку даних, даною інформацією зможуть використати зловмисники та конкуренти.

Щоб безпека інформації залишалась на високому рівні, розроблено норми застосування АРМ та реалізація таких правил лежить загалом на операторах.

Засади, котрі допоможуть запобігти небезпекам: необхідно поставити антивірусну програму, використовувати ліцензійні продукти, своєчасно встановлювати оновлення сигнатури безпеки, ставити всі наявні оновлення системи та не запускати і не встановлювати інакші СКБД та інші прикладні програми, комп'ютер на якому встановлено АРМ сервер, зобов'язаний бути у

надійній місдині і не може бути розсекреченим. Доступ до такого приміщення повинен бути не легким.

2.2. Рішення по інформаційному забезпеченню

АРМ буде застосовувати тільки оператор відділення, інформація до системи надходить від одного користувача – оператора відділення, проте з різних джерел.

Оператор вводить до системи такі дані:

- Для реєстрування клієнта відправника: прізвище, ім'я, по батькові, телефон;
- Для реєстрування клієнта отримувача: прізвище, ім'я, по батькові, телефон, регіон, місто отримувача, та номер поштового відділення;
- Для формування накладної на посилку: вага, об'єм, та тип посилки;
- Для формування маршрутного листа на відправлення вантажівки: номери усіх посилок які потрібно відправити;
- Для формування маршрутного листа на прибуття вантажівки: номери усіх посилок які потрібно прийняти у відділення;
- Для закриття зміни у відділенні оператором: номери усіх посилок які є у відділенні на момент закриття зміни;
- Для відкриття зміни у відділенні оператором: номери усіх посилок які є у відділенні на момент відкриття зміни;
- Для входу у систему: логін та пароль;
- Для формування накладної на зворотне відправлення: потрібно лише натиснути на відповідну клавішу на панелі;
- Для закриття накладної: потрібно лише натиснути на відповідну клавішу на панелі;

Усі вхідні дані вимагають збереження у БД, це забезпечує функціональність СКБД.

Вихідні дані системи теж одержує тільки користувач та уже він може розпоряджатися ними як йому потрібно. З АРМ надходять такі дані:

- Сформована накладна на посилку;
- Закрита накладна на посилку;
- Маршрутний лист на прибуття вантажівки;
- Маршрутний лист на відправлення вантажівки;
- Фіксація закритої зміни;
- Фіксація відкритої зміни;
- Формування накладної зворотної доставки;
- Перелік клієнтів і їх дані;
- Перелік співробітників і їх дані;
- Перелік посилок на складі;
- Перелік наявних відділень;
- Різноманітні звіти пов'язані з доставкою.

Потрібно розробити структуру бази даних для ефективного зберігання інформації та її обробки.

Основні частини концептуальної моделі даних - це відносини та об'єкти. Об'єкти репрезентують собою абиякий реальний (конкретний) об'єкт у наданій області.

Сутність - деяка абстракція реального світу у визначеній постановці задачі, явища або процесу, про котрий потрібно зберегти у системі інформацію. Синонімом терміну «сутність» застосовується теж один з термінів як «інформаційний об'єкт».

Сутність - об'єкт природи або реального світу, дані про котрий зберігаються в відношенні (таблиці, в якій містяться дані).

Об'єкти у будь-який мить часу характеризуються спеціальним станом, який описується декількома зв'язками, відносин та властивостей з іншими об'єктами.

Характеристика, яка окреслює якусь ознаку сутності, яку потрібно сформулювати та записати, зветься атрибутом. Атрибут, котрий характеризує сутність, зветься ідентифікатором.

Кожний об'єкт предметної області характеризується набором атрибутів, які відображають певні особливості об'єкта. Атрибути застосовуються за для визначення, інформація яких зобов'язана бути зібраною про об'єкт. Зразками атрибутів об'єкта «Клієнт-відправник» є Прізвище, Ім'я, по-Батькові, моб. телефон і т.д.

У предметної області також можна виокремити такі сутності:

- 1) Клієнт,
- 2) Оператор,
- 3) Посилка,
- 4) Накладна,
- 5) Відділення,
- 6) Маршрутна накладна,
- 7) Історія перевезень,
- 8) Аккаунт,
- 9) Адреса.

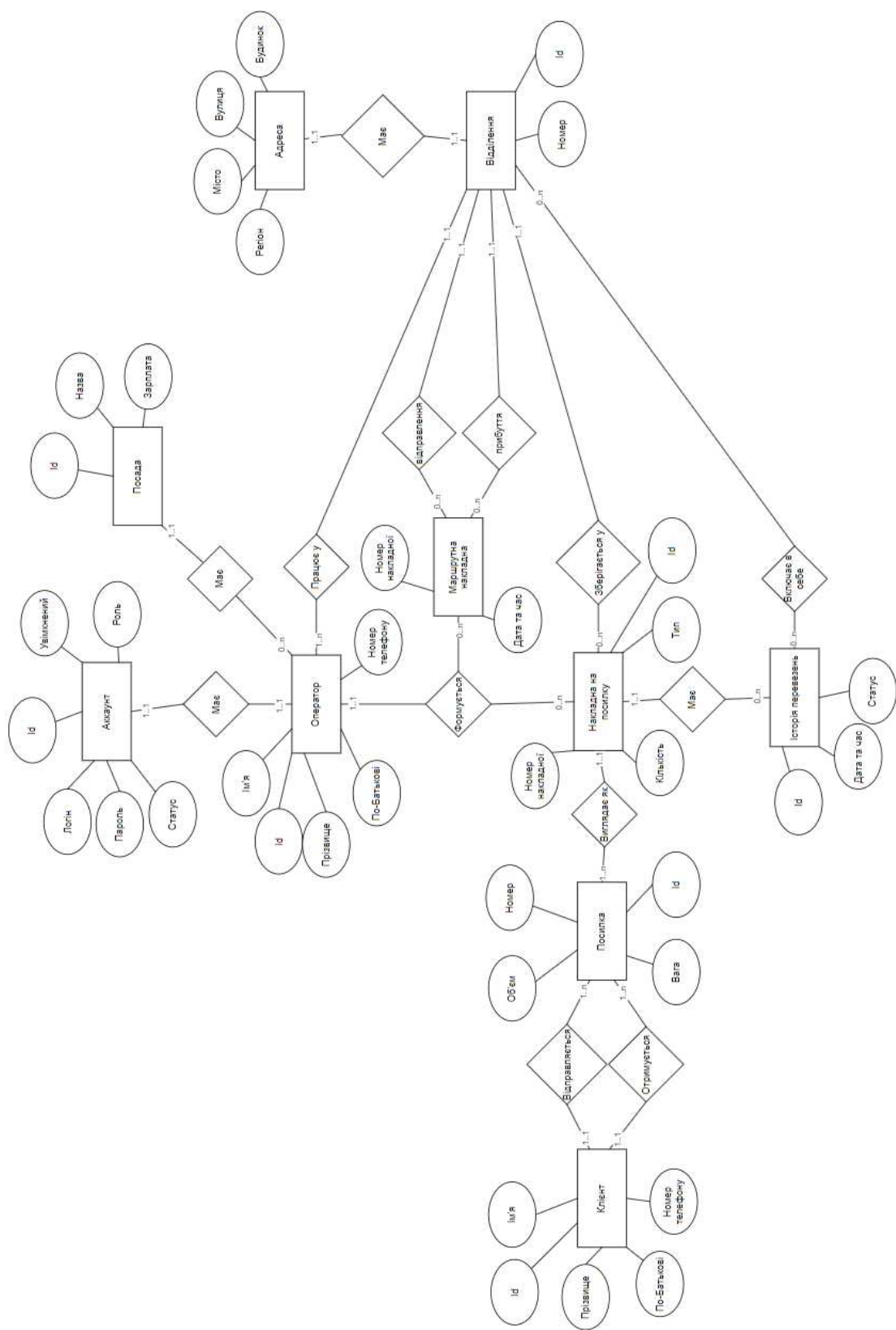


Рисунок 2.3. Концептуальна модель БД.

На стадії проектування визначаються основи БД, а також обов'язково потрібно визначити логічну структуру бази для даної предметної області. Проект логічних структур баз даних установлює набір об'єктно – реляційних таблиць, а також їх та логічні зв'язки та структуру по між нашими таблицями. На етапі формування структури БД та таблиць виділяється множина стовпців (полів), для яких дається розміру даних, опис типу та інших особливостей. Окрім цього, має бути присутній унікальний первинний ключ у таблиці, котрий може бути з одного поля або бути композитним, тобто складатися з декількох полів.

Інформацію про структуру БД слід зібрати, виявити властивості та характеристики об'єктів, визначити ці об'єкти системи та зв'язки між цими об'єктами, з котрими слід створити складові вихідні таблиці, стовпці та поля, та для усіх об'єктів позначити первинні унікальні ключі та установити зв'язки між ними.

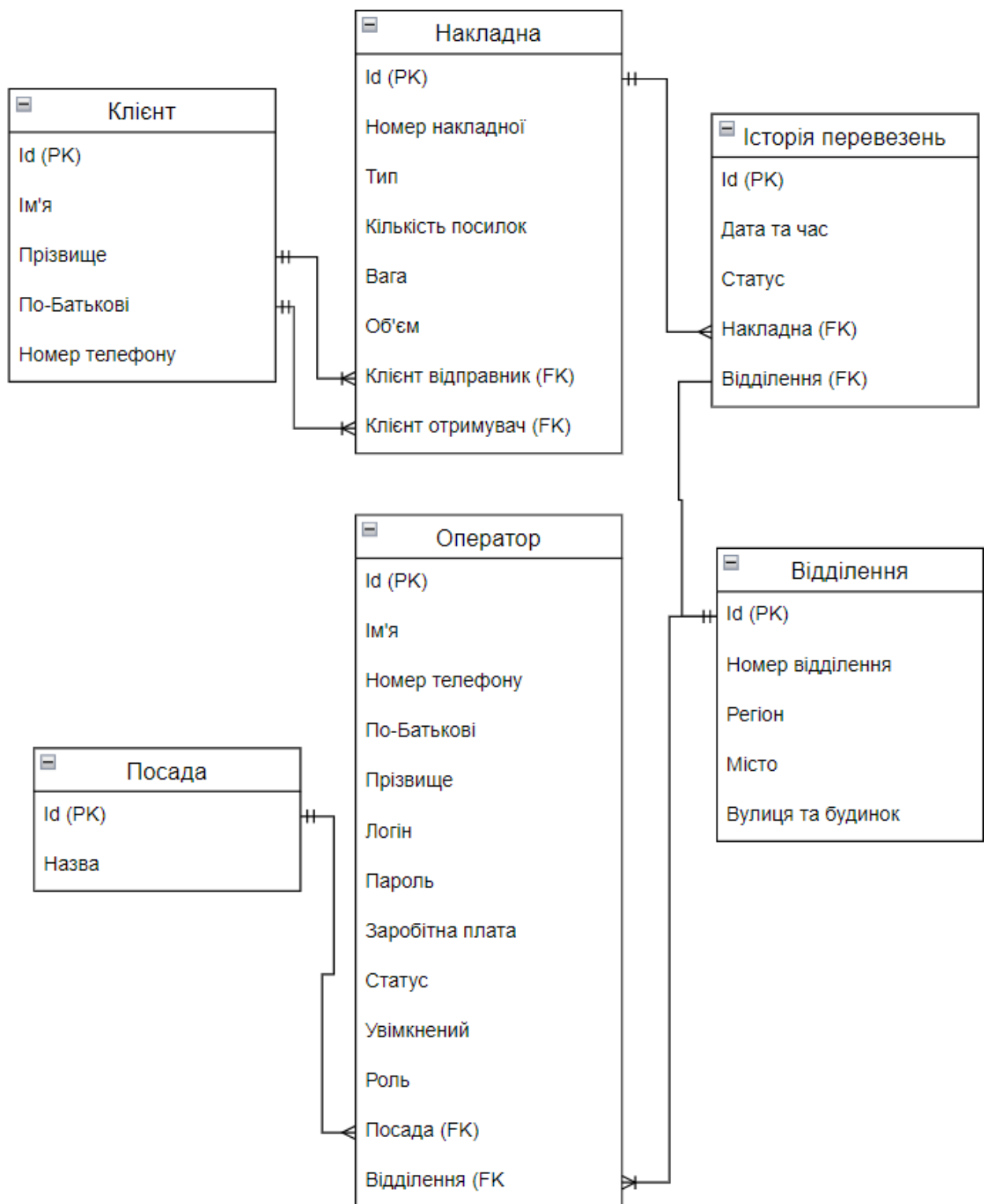


Рисунок 2.4. Логічна модель БД.

Логічна модель та структура БД - це структура або модель для користувача системи, а фізична модель - це структура БД для самого комп'ютера. Фізична модель дозволяє визначити, властивості даних та тип, котрі запишуться у пам'яті комп'ютера.

Норми переходу від логічної структури БД до фізичної структури такі: усі відношення переводиться у файли бази даних, всі стовпеці переводяться у поля файлу, будь-який рядок - у записи файлу. Стадія фізичного конструювання бази даних охоплює в собі формування складу файлів та заповнення їх відповідно до обмежень вихідними даними, а також допущеннями та особливостями даної предметної області.

Фізичне моделювання БД – це конструювання в СКБД таблиць. В ході фізичного проектування бази даних слід зазначити імена кожної з таблиць, обрати типи та дати імена полям таблиць. СКБД PostgreSQL дає для зберігання та використання величезну кількість різних типів даних для створення будь-якого поля у таблиці. Опрацювавши всі визначені варіанти і вибравши підходящі, відповідно до логічної моделі, розроблено фізичну структуру БД.

Зв'язок між таблицями використовуються за допомогою полів `id` у таблицях, значення якому не має ідентичних, на основі цього і здійснюється зсилання та за сприянням цього без переконливо можемо ідентифікувати записи у кожній з таблиць, підходяще поле у таблицях із яких здійснюється посилення, які іменуються за назвою таблиць, на котрі посиляється, та назви полів цих таблиць. Зокрема, якщо ми зв'яжемо таблицю `Invoice` по полю `id`, відповідне поле в таблиці, котру потрібно зв'язати із цією таблицею, матиме таку назву як `invoice_id`. При цьому типи даних двох полів зобов'язані бути ідентичними та у цьому інциденті – це буде тип `int` або `bigint`.

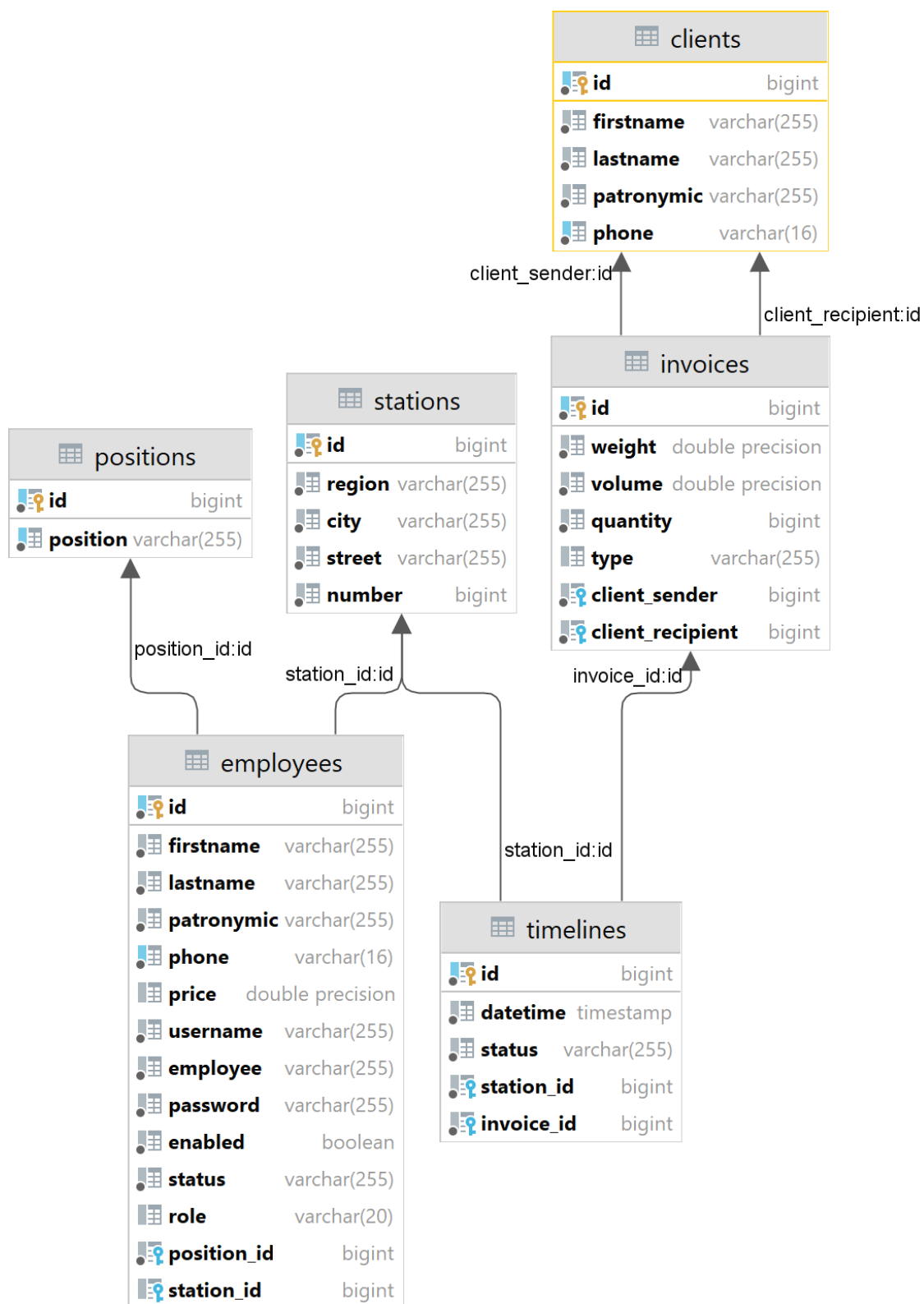


Рисунок 2.5. Фізична модель БД.

Діаграма класів формулює типи класів системи та зв'язки між ними, котрі бувають між ними. На діаграмі класів описують поля та атрибути класів їх обмеження та методи, котрі накладаються на зв'язок між цими класами. Інтерпретація та вид діаграми класів залежить зокрема від сторони та точки зору (абстрактного рівня): класи також можуть репрезентувати собою сутності даної предметної області у процесі аналізу та/або частини програмної системи у процесі реалізації та проектування. На базі спроектованих моделей та структур реалізована діаграма класів, котра показує відносини між нашими сутностями та об'єктами.

Проект БД буде реалізовуватись у процесі формування програмного забезпечення нашої системи.

2.3. Рішення по програмному забезпеченню

На основі спроектованої логічної і фізичної моделі бази даних вирішено користуватися СКБД PostgreSQL. Вона ґрунтується на мові SQL та підтримує доволі багато можливостей та стандартів SQL: 2011. На міркування більшості це надзвичайно стабільна СКБД, у ній практично неможливо зламати таблиці і знищити дані та неможливо покласти. Це також грамотна і стара СКБД. Вона приблизно як MySQL та тільки краще, але потрібно уміти її налаштувати. Важливими чинниками при обранні PostgreSQL – це гнучкість у використанні, надійність, вкрай структуровані дані та легке масштабування і розширення. За допомогою сторонніх засобів та бібліотек дуже зручно та легко розширювати та працювати з БД

Код програми розроблено у IDE IntelliJ IDEA на мові програмування Java. Одна з найбільш популярних та поширених мов програмування. Java – універсальна мова програмування, яка застосовується для вирішення багатоманітного походження завдань. За весь період існування переросла із простої універсальної мови у повну екосистему та платформу, яка єднає різні технології, які застосовуються для широкого спектру задач: від написання великих веб-порталів і сервісів до створення десктопних додатків. Ще мова Java завзято використовується для створення програмного забезпечення для неймовірної кількості різних пристроїв: побутової техніки, ПК, смартфонів, планшетів.

Для написання рівня бізнес логіки і серверу використано фреймворк Java - Spring. Spring Boot – проект мета якого являє собою спрощення створення додатків на базі фреймворку Spring. Він дозволяє легко створювати

web – додатки, які вимагатимуть від розробника мінімальних зусиль за написанням коду та його налаштуванням. Особливість Spring Boot проекту – це інтегрований Tomcat сервер.

Для роботи із базою даних разом з Spring framework використовується Hibernate бібліотека, яка дозволяє зіставити із таблиці нашої бази даних та класи програми, шляхом перетворення типів мови Java на типи СКБД і навпаки, перетворивши сутності на класи. Hibernate на основі сутностей автоматично формує таблиці зі всіма заданими атрибутами і генерує зовнішні та первинні ключі, а також поліпшує обробку отриманих з записів таблиць, роботу з таблицями і формування запитів.

Spring MVC – Spring веб – фреймворк який дозволяє створювати веб – сайти, сервіси або RESTful сервіси (XML / JSON) та дуже добре вбудовується у систему Spring, підтримує REST контролери та контролери в додатках Spring Boot. Інтерфейс користувача написаний на мові гіпертексту HTML та використовує каскадні шаблони та стилі CSS за для придання зручного інтерфейсу для користувача.

Веб-дизайн – різновид графічного дизайну, спрямований на розроблення і формування об'єктів середовища Інтернет який забезпечує їх високими естетичними атрибутами та споживчими особливостями. Гідне трактування відділяє веб-дизайн від веб-програмування та виділяє специфіку діяльності веб-дизайнера який позиціонує веб-дизайн як різновид графічного дизайну.

У теперішній час, тезисом веб-дизайн розуміють забезпечення зручності використання ресурсу та розроблення структури веб-ресурсу для кінцевих користувачів.

Вагомим елементом розроблення ресурсу в останні пори стає доведення продукту до відповідності W3C стандарту який дозволяє забезпечити верстки ресурсу такі як кроссплатформенна верстка або у нашому випадку крос-

браузерна верстка, а також доступність ресурсу для користувачів з портативними пристроями та інвалідам.

Верстка веб-сторінок – це процес розроблення структури документа гіпертексту на базі мови HTML-розмітки, як норма, при використанні клієнтських сценаріїв та таблиць стилів, щоб деталі дизайну показувалися аналогічно дизайну макету.

Поліграфічної верстка відрізняється від верстки веб-сторінок тим, що потрібно враховувати відмінність у форматах робочої території пристроїв та відмінність відображення деталей у різноманітних браузерах.

Хід роботи важкий та має за собою творчу основу так як аніякий із засобів не являє собою прийнятим та канонічну основу. Усі заходи до верстки мають як недоліки так і переваги.

HTML («мова розмітки гіпертексту») – стандартизована мова розмітки веб-сторінок. HTML-код браузерами інтерпретується та в результаті отримана інтерпретацією сторінка виводиться на екран монітора мобільного пристрою або комп'ютера.

CSS застосовується верстальниками веб-сторінок для завдання розташування окремих блоків, шрифтів, кольорів, стилів та інших аспектів та видів представлення зовнішнього відображення веб-сторінок. Головною ціллю розроблення CSS - винести опис логічної моделі веб-сторінки (яка розробляється за допомогою мови HTML або інших мов розмітки) від описання зовнішнього відображення цих веб-сторінок які стилізуються за допомогою CSS мови. Подібним розподілом можна підвищити можливість управління його поданням, доступність документа та надати велику гнучкість, а ще зменшити повторюваність та складність у структурованому вмісті.

Так як мова програмування Java є кроссплатформенною, а значить для використання і встановлення спроектованої системи, можемо застосувати яку

завгодно операційну систему. Нараз найбільш розповсюджені та популярні ОС, це ОС Linux, ОС Windows та ще Unix подібні системи.

Для впровадження відповідної безпеки інформації потрібно щоб користувач інсталиував собі на сервер антивірусну систему. Безкоштовний антивірус Avast є рекомендацією для інсталиування та доступний на ОС Linux і Windows та вдосконалює безпеку системи від троянських, шпигунських та шкідливих програмних продуктів і також від всяких інших видів загроз, таких як фішинг, черв'яків, adware та інших.

Елементи системи розроблені як ізольовані, модульні підпрограми, які мають функції які описані у розділі 2.1. Розроблення програми обробки і візуалізації даних.

Програмний комплекс має наступні елементи:

1. Контролери:

- AuthenticationController – обробка даних по аутентифікації, авторизації та ідентифікації користувачів;
 - WaybillController – обробка даних по роботі вантажівкою;
 - BaseController – обробка запитів головного меню;
 - TimeshiftController – обробка даних роботи зі зміною;
 - StorageController – обробка даних посилки, які зберігаються на складі;
 - InvoiceController – обробка даних по формуванню та закриттю накладних посилки;
 - EmployeeController – обробка даних робітників;
2. Сутності – об'єкти бази даних які потрібні для роботи у програмі;

3. Інтерфейс роботи із СКБД – реалізація отримання відповідей та запитів до СКБД;

4. Web сторінки і шаблони – відображення графічного інтерфейсу користувача системою.

Також існує ще один програмний модуль – “Application”. Це базовий елемент розробки, котрий агрегує функціонал для ініціалізації нашої програми. У цьому модулі реалізує функціонал, який є центральною точкою для входу у програму.

Розроблений продукт налагоджено та виконано тестування його: правильності виконання операцій, загрузки у базу даних, розрахунок значень на тестових наборах даних. Верифікація і валідація системи пройшла успішно та система абсолютно відповідає призначенням та умовам які було задано ще на стадії формування вимог.

2.4. Рішення по технічному забезпеченню

Головні технічні елементи АРМ оператора поштового відділення котрі забезпечують функціонування системи – це:

- Роутер Wi-Fi – який дає можливість серверу доступ до інтернет мережі.
- Пристрій із доступом до Інтернет мережі – який дозволяє підключитись до серверної частини і одержати доступ до АРМ оператора у віддаленому режимі що надає нам здатність застосовувати усі можливості АРМ оператора з підтримкою user-friendly графічного інтерфейса у відділенні, наприклад сформувати маршрутний лист.
- Сервер – на котрому встановлено програмне забезпечення АРМ оператора, СКБД, та розташовується БД поштового відділення. Сервер одержує та направляє всі запити до програмного забезпечення АРМ оператора.

Встановлене обладнання сервера необхідно щоб воно відповідало подібним найменшим технічними параметрам:

1. Процесор – Intel Core i3 4220 і більше;
2. Оперативна пам'ять – 8 Gb;
3. Мережева карта – не менше 100Мбіт з Wi-Fi підтримкою;
4. Місце на SSD диску більше ніж 120 Гбайт;
5. Монітор з мінімальною роздільною здатністю 1280*720 і більше.

За для забезпечення можливості працювати із системою АРМ оператора або щоб запобігти втраті у БД інформації, необхідно розв'язати питання серверного живлення, слід оснастити сервер засобами безпечного вимкнення або поставити джерело резервного живлення при його втраті.

Для використання та підключення АРМ дозволено користуватися практично будь-яким пристроєм, персональним комп'ютером, ноутбуком, смартфон, та інші. Вимоги до таких приладів не високі, а саме: підтримувати

HTML5 і доступ до Інтернет мережі. Розмір оперативної пам'яті для смартфона повинен бути більше ніж 4ГБ, а для ПК на Linux або Windows 6-8ГБ.

Всі технічні елементи декларують взаємозв'язок між собою за допомогою бездротового зв'язку. Wi-fi підключення роутера до мережі інтернет здійснюється кабелем оптоволоконним кабелем або крутою парою, також можна користуватися модемом 3G/4G/5G. Wi-Fi Роутер роздає мережу на принтер, сервер та інші пристрої локальні мережі таким чином єднаючи їх в одну мережу. Також принтер, роутер та сервер дозволено зв'язати LAN кабелем. Для підключення пристрою до серверу, з якого хочемо працювати, це може здійснюватися ззовні локальної мережі за допомогою бездротового зв'язку 3G/4G/5G.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОЕКТУ АРМ ОПЕРАТОРА ПОШТОВОГО ВІДДІЛЕННЯ

Автоматизоване робоче місце оператора поштового відділення призначено для поліпшення обліку роботи з посилками, формування та закриття накладних, ефективного та швидкого формування прибуття та від'їзду вантажівки з посилками. Інтерфейс користувача АРМ побудований на базі web – додатку та надає зручні можливості користування системою та реалізованими функціями, такими як сформувати накладну на посилку, закрити накладну, сформувати накладний маршрутний лист на прибуття або від'їзд вантажівки з посилками, швидко відкрити та закрити зміну у відділенні, вести облік та історії посилок та інші.

Програмне забезпечення системи поставляється у вигляді серверного Java додатку, який поєднує інтерфейс користувача(візуалізація даних) та бізнес логіку(обробка запитів).

Для збереження даних програмний додаток використовує СКБД PostgreSQL.

Для безперебійного якісного користувацького досвіду користування продуктом, сервер потрібен відповідати наступним мінімальним технічним характеристикам:

1. Процесор – Intel Core i3 4220 і більше;
2. Оперативна пам'ять – не менше 8 Gb;
3. Мережева карта – не менше а ніж 100Мбіт та з Wi-Fi підтримкою;
4. Місце на SSD диску більше за 120 Гбайт;
5. Монітор з роздільною здатністю не менше 1280*720 або більше;
6. Встановлену ОС Linux або Windows.

Перед початком інсталяції програми, потрібно встановити СКБД PostgreSQL. Для встановлення даного програмного продукту необхідно запустити файл програми інсталяції «postgresql-11.0-1.exe», який йде у комплекті поставки разом з іншими файлами інформаційної системи. Спочатку інсталятор перевірить наявність всіх необхідних компонентів, зокрема Visual C ++ Redistributable, в разі необхідності, тобто їх відсутності, він їх сам встановить.

Після цього відкриється вікно вітання представлене на рисунку 3.1, натискаємо «Next».

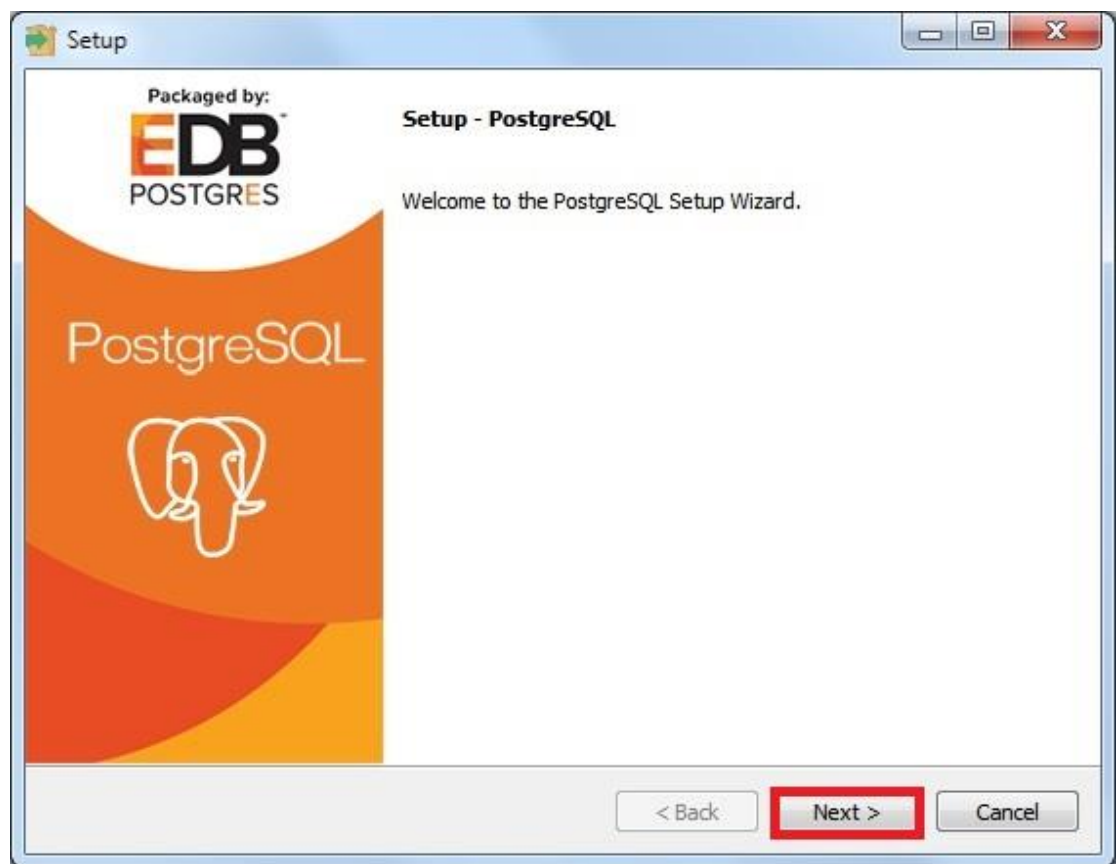


Рисунок 3.1. PostgreSQL 11 Інсталятор.

Потім нам потрібно вказати шлях до каталогу, в який ми хочемо встановити PostgreSQL 11, потрібно залишити за замовчуванням. Натискаємо «Next».

Переходимо до етапу установки пароля для користувача рисунок 3.2 postgres, іншими словами, для адміністратора PostgreSQL Server. Вводимо пароль і підтверджуємо його. Натискаємо «Next».

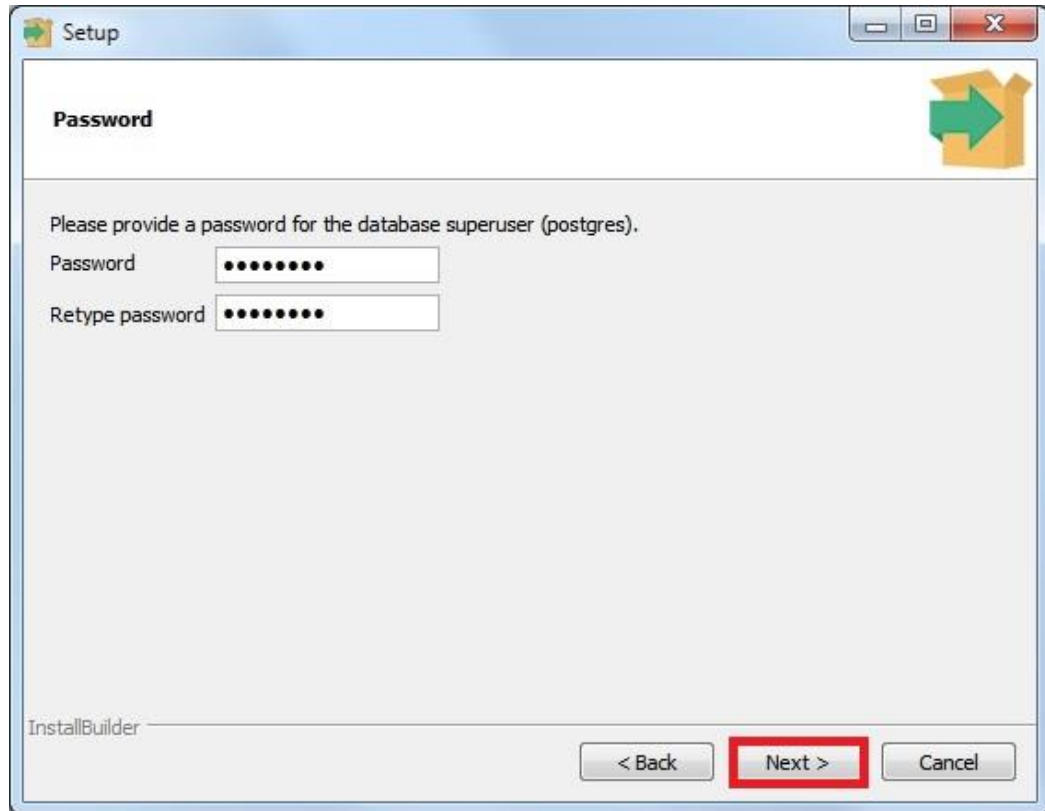


Рисунок 3.2. Задавання паролю серверу PostgreSQL.

Прослухований порт сервера залишаємо за замовчуванням.

Для того щоб запустити процес інсталювання PostgreSQL в даному вікні рисунок 3.3, натискаємо «Next».

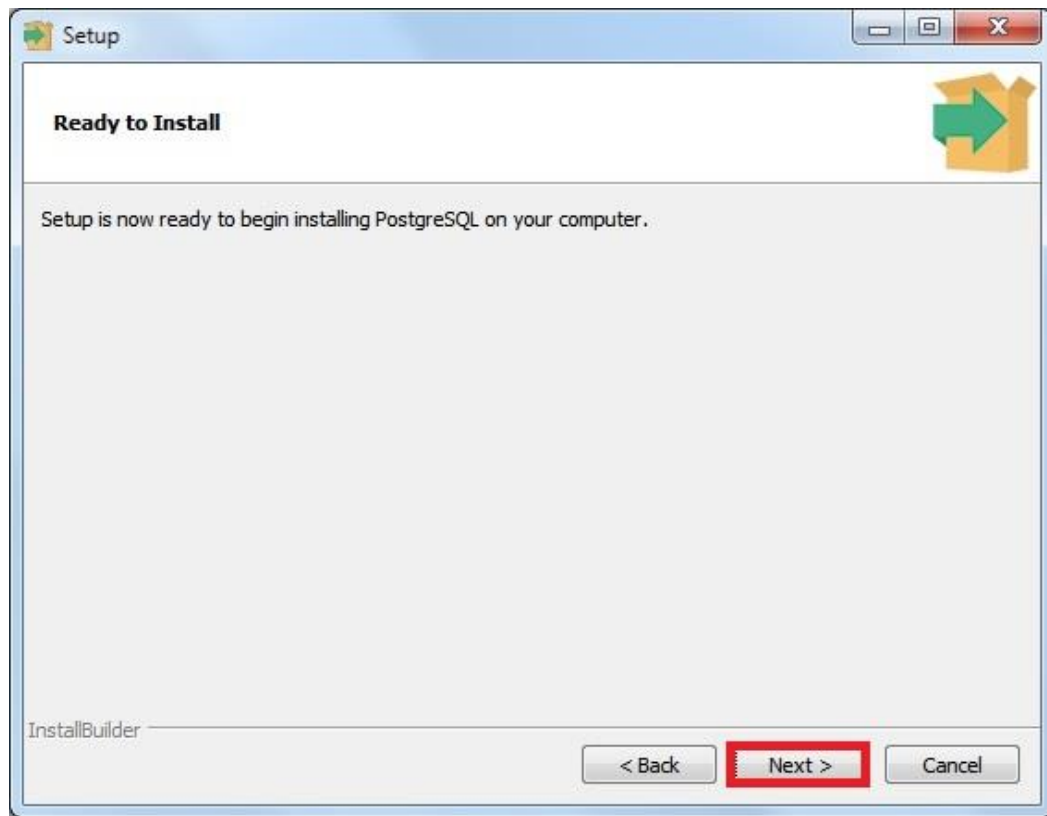


Рисунок 3.3.Запуск інсталяції.

Коли з'явиться вікно з повідомленням «Completing the PostgreSQL Setup Wizard» установка PostgreSQL, pgAdmin 4 і інших компонентів буде завершена, це значить що можна працювати з СКБД.

Після цього можна приступати к другому етапу – встановленню серверного додатку, це відбувається за допомогою розпаковки архіву у потрібний каталог.

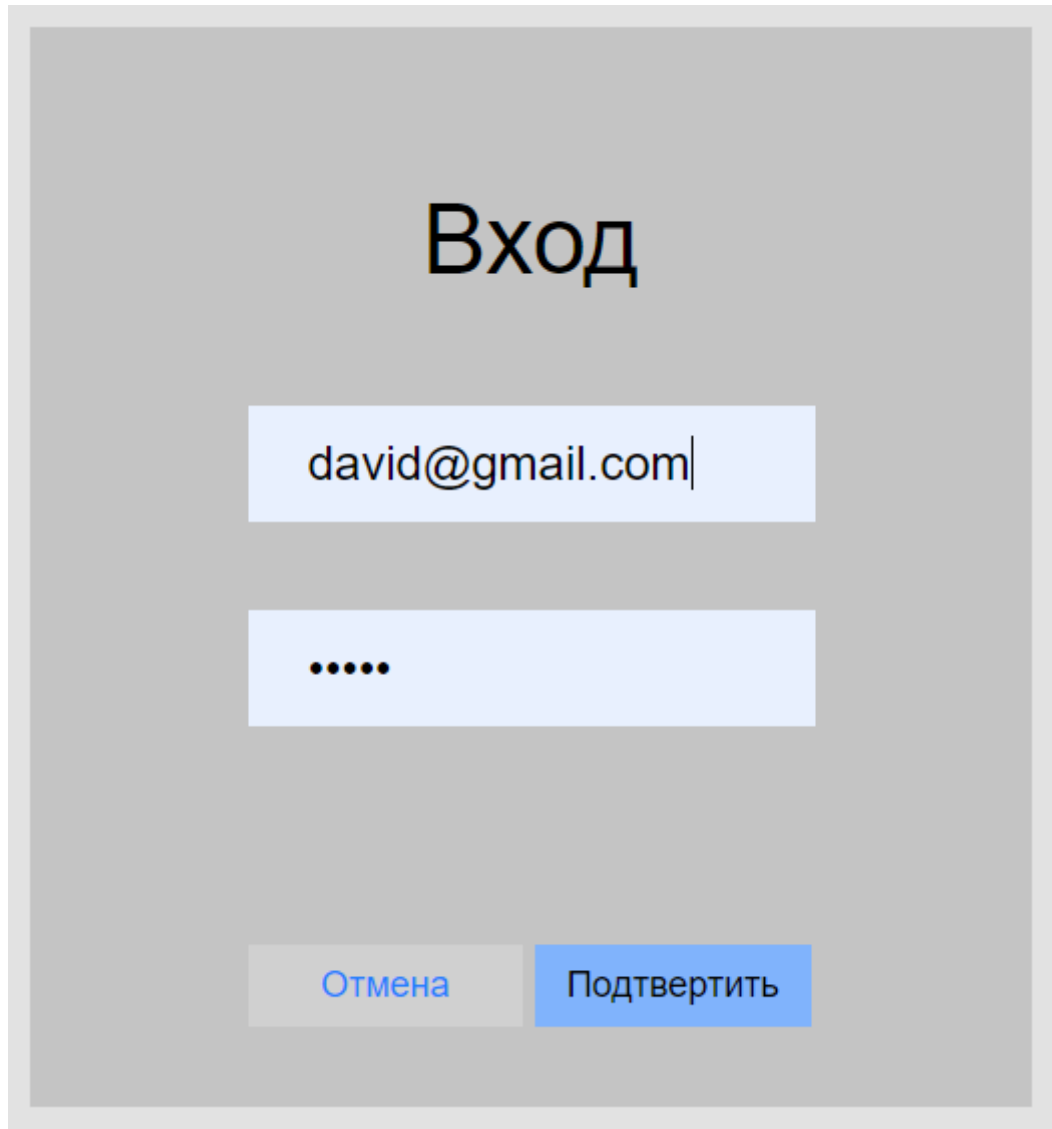
Після розпаковки, при першому запуску системи автоматично буде виконано скрипт із додатку А, це призведе до створювання та ініціалізації окремих таблиць бази даних, з якими працює програмне забезпечення системи.

Для того, щоб постійно не завантажувати систему вручну при включенні сервера, потрібно додати файл запуску системи до автозапуску операційної системи. По завершенню встановлення серверної програми та СКБД необхідно

перезавантажити сервер. Після цього можна підключатися до сервера та розпочинати користування.

Після успішного запуску та переходу по відповідному Url, відобразиться сторінка входу до системи – рисунок 3.4а та рисунок 3.4б.

Рисунок 3.4а. Вхід до системи.



The image shows a login interface with a light gray background. At the top, the word "Вход" (Login) is displayed in a large, black, serif font. Below it, there are two light blue input fields. The first field contains the email address "david@gmail.com" followed by a vertical cursor. The second field contains five black dots, representing a masked password. At the bottom of the form, there are two buttons: a light gray button labeled "Отмена" (Cancel) and a blue button labeled "Подтвердить" (Confirm).

Рисунок 3.4б. Вхід до системи.

Потрібно ввести логін та пароль робітника щоб відкрилась відповідна панель.

Після натискання «Підтвердити» може відбутись 3 події:

- При введенні невірного логіну або паролю система не дасть увійти та перекине до сторінки з помилкою та дасть змогу повторити спробу входу у систему;

- Якщо данні введено вірно то система перекине нас на сторінку для відкриття зміни – рисунок 3.5;
- Якщо данні введено вірно та зміна була відкрита до цього то система перенесе нас до переліку посилок на складі;

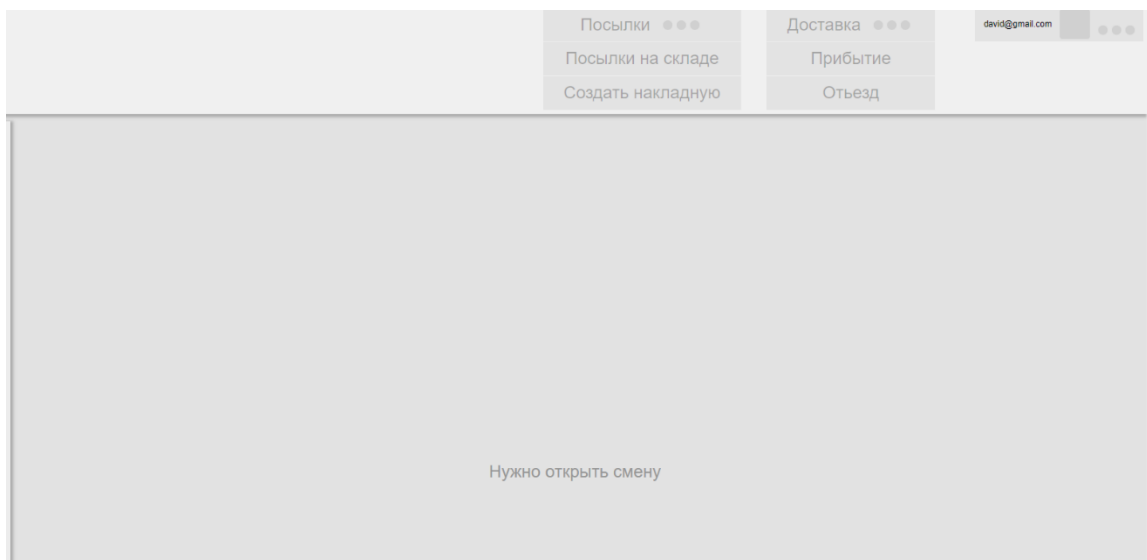


Рисунок 3.5. Успішний вхід але зміна не відкрита і система просить відкрити зміну.

Посылки						
Доставка						
Посылки на складе						
Прибытие						
Создать накладную						
Отъезд						
Номер накладной	Получатель	Количество	Вес	Объем	Тип	
1	shakov anton vitalevich	2	12.0	3.0	Metal construction	1
2	shakov david vitalevich	1	2.0	4.0	Monitor	2

Рисунок 3.5. Успішний вхід та зміна відкрита і одразу відображається перелік посилок на складі.

Головна панель містить наступні елементи:

- Посилки на складі;
- Сформувати накладну;
- Прибуття вантажівки;
- Від'їзд вантажівки;
- Підменю для роботи зі зміною та виходу з системи;

Інтерфейс користувача було розроблено із використанням методології user-friendly, усе інтуїтивно зрозуміло та виконано у спокійних не дратівливих тонах. Екрани додатку представлені на рисунках 3.5. – 3.10.

The screenshot shows a web application interface for creating a shipping document. The interface is divided into three main sections: Sender Data, Package Data, and Delivery Data. Each section contains several input fields. At the top right, there are tabs for 'Посылки' (Packages) and 'Доставка' (Delivery), and a user profile 'david@gmail.com'. At the bottom right, there are 'Отмена' (Cancel) and 'Подтвердить' (Confirm) buttons.

Данные отправителя	Данные посылки	Данные доставки
Имя	Тип посылки	Платательщик
Фамилия	Вес	Стоимость
Отчество	Объем	Форма расчета
№ тел	Количество	Дополнительные услуги
		Срок хранения
		Доставка в руки
Данные получателя		
Имя		
Фамилия		
Отчество		
№ тел		
Регион, город, улица, номер		

Отмена Подтвердить

Рисунок 3.5. Формування накладної та запис даних посилки

Посылки

Посылки на складе

Создать накладную

Доставка

Прибытие

Отъезд

david@gmail.com

Номер накладной	Получатель	Количество	Вес	Объем	Тип	
1	shakov anton vitalevich	2	12.0	3.0	Metal construction	
2	shakov david vitalevich	1	2.0	4.0	Monitor	
3	shakov maria vitalevich	1	0.5	0.5	Phone	

Рисунок 3.6. Поточний перелік посилок на складі

Для реалізації проекту було розроблено декілька документів щодо розробки інформаційної системи, які наведено у додатках:

Код створення БД (Додаток А);

Код програми (Додаток Б).

В результаті реалізовано проект АРМ оператора поштового відділення, який дозволяє вирішити поставлені задачі, такі як: формування накладної, закриття накладної, формування зворотної доставки, поточний перелік посилок на складі, відстеження посилок та їх історія перевезень, облік посилок, закриття та відкриття зміни, формування маршрутних листів на прибуття та від'їзд вантажівки з посылками. Це підвищило ефективність та швидкість обробки посилок та формування накладних на них, за рахунок раціонального використання часу та ресурсів, система дозволила значно підвищити продуктивність, ефективність праці і спростила роботу з даними оператору поштового відділення.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ ТА ТЕХНІКА БЕЗПЕКИ

Охорона праці – система правових, соціально-економічних, організаційних, технічних, гігієнічних та лікувально-профілактичних заходів та засобів, спрямованих на забезпечення збереження здоров'я та працездатності людини в процесі праці.

В процесі роботи людина взаємодіє з виробничим середовищем, з засобами виробництва та з предметами праці. При цьому, вона зіштовхується з великою кількістю факторів, різних по своїй природі, формам прояву та характеру впливу, які діють на стан здоров'я та працездатність людини.

Виробничі фактори залежно від наслідків, до яких може привести їх дія, поділяють на небезпечні та шкідливі.

Небезпечний виробничий фактор – фактор, вплив якого на працюючого у визначених умовах приводить до травми або різкого погіршення здоров'я.

Шкідливий виробничий фактор – фактор, вплив якого на працюючого у визначених умовах призводить до захворювання або зниженню працездатності.

В залежності від сили та тривалості впливу будь-який шкідливий фактор може стати небезпечним. За природою впливу на організм людини небезпечні та шкідливі виробничі фактори поділяються на чотири групи: фізичні, хімічні, біологічні та психофізіологічні.

До фізичних небезпечних та шкідливих виробничих факторів належать фактори, що відносяться до технологічного процесу(верстати, механізми, рухомі машини, рухомі частини машин, переміщувані вироби, заготовки та матеріали, гострі грані виробів або машин, задирки, підвищена або знижена температура поверхонь обладнання або матеріалів, підвищене значення електричної напруги у мережі, підвищений рівень статичної електрики на корпусі, одязі та інших поверхнях), та фактори, що характеризують повітря виробничих приміщень (підвищена запиленість, кількість шкідливих випарів у

повітрі робочої зони, значний рівень шуму, ультразвукові коливання, вібрації на робочому місці, недостатня кількість або не правильно підібране світлове освітлення робочої зони, метеорологічні умови).

Хімічні небезпечні та шкідливі виробничі фактори поділяються:

- за характером впливу на людину на: токсичні (викликають отруєння організму), дратівні, сенсibiliзуючі (викликають алергію), канцерогенні (викликають злоякісні утворення), мутагенні (впливають на зміни спадковості), репродуктивні;
- за шляхом проникнення у організм людини: проникаючі через органи дихання, шлунковокишечний тракт, шкіру та слизові оболонки. Біологічні небезпечні та шкідливі виробничі фактори містять такі біологічні об'єкти: мікроорганізми (бактерії, віруси та ін.) та продукти їхньої життєдіяльності, макроорганізми (рослини та тварини) [9].

Психофізіологічні – фізичні та нервово-психічні перевантаження.

На даний момент відомо – цілком безпечних і нешкідливих виробництв не існує, тому основною задачею є зведення до мінімуму імовірності поразки чи захворювання працюючого з одночасним забезпеченням комфорту при максимальній продуктивності праці.

Комплекс організаційних і технічних заходів і засобів, що запобігають нещасні випадки, що виникають при впливі на працюючого небезпечних виробничих факторів, називається технікою безпеки. Виробнича санітарія містить у собі комплекс організаційних, гігієнічних і санітарно-технічних заходів, що запобігають виробничі шкідливості.

Турбота про охорону праці являється 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. Основними пунктами в цьому законі є: стаття про гарантію прав громадян на охорону праці, стаття про відшкодування моральної шкоди, статті про охорону праці жінок, неповнолітніх, інвалідів.

У Законі "Про охорону праці" зазначені органи, покликані здійснювати нагляд і контроль за дотриманням законодавства про працю і правил по охороні праці. До них відносяться спеціально уповноважені на те державні органи й інспекції, що не залежать у своїй діяльності від адміністрації підприємств, установ, організацій і їхніх вищих органів.

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.

Згідно із Законом України «Про охорону праці» за порушення законодавчих та інших нормативних актів про охорону праці (розділ VIII) встановлені різні види відповідальності: дисциплінарна, адміністративна, матеріальна, кримінальна. Передбачена відповідальність як підприємств, так і самих працівників.

Дисциплінарна відповідальність регулюється Кодексом законів про працю і передбачає такі види покарання, як догана та звільнення.

Адміністративна відповідальність — це відповідальність посадових осіб і працівників перед органами державного нагляду, що полягає у застосуванні до них штрафних санкцій. Умови притягнення до адміністративної відповідальності передбачені Кодексом України про адміністративні правопорушення. Ст. 41 КУпАП передбачає, що порушення вимог законодавчих та інших нормативних актів з охорони праці тягне за собою накладення штрафу на винних осіб. Контроль і нагляд за дотриманням законодавчих та інших нормативних актів з охорони праці, з безпеки проведення робіт здійснюють органи державного нагляду за охороною праці.

Матеріальною відповідальністю передбачено відшкодування збитків, заподіяних підприємствами працівникам (або членам їх сімей), які постраждали від нещасного випадку чи профзахворювання.

Кримінальна відповідальність за порушення правил охорони та безпеки праці передбачена ст. 135, 218, 219 і 220 Кримінального кодексу України. Ст. 135 містить загальне поняття складу та суб'єкта злочину. Суб'єктом злочину з питань охорони праці є будь-яка службова особа підприємства, установи, організації незалежно від форм власності, а також громадянин — власник підприємства чи уповноважена ним особа. Згідно з цією статтею порушення вимог законодавчих та інших нормативних актів про охорону праці посадовою особою підприємства, установи, організації незалежно від форм власності, громадянином — власником підприємства чи уповноваженою ним особою, якщо це порушення створило небезпеку для життя або здоров'я громадян, карається виправними роботами на строк до одного року або штрафом до 15 мінімальних розмірів заробітної плати. Те саме діяння, якщо воно спричинило

нещасні випадки з людьми, карається позбавленням волі на строк до чотирьох років [13].

4.1. Аналіз небезпечних і шкідливих факторів при роботі з посилками

На підприємствах на самопочуття, стан здоров'я людини впливає багато різних факторів, таких як освітлення, шум, мікроклімат виробничих приміщень. Які визначається дією на організм людини яскравістю, шумністю, температурою, вологістю, рухомості повітря і теплового випромінювання. Виробничий мікроклімат, як правило, відрізняється значною мінливістю, нерівномірністю по горизонталі та вертикалі, різноманітністю сполучень температури, вологості, рухомістю повітря, інтенсивністю випромінювання залежно від особливостей технології виробництва, перевезень, зберігання, кліматичних особливостей місцевості, організації повітрообміну із зовнішнім середовищем.

При роботі з посилками використовується дуже багато обладнання та сам процес доставки посилок представляє собою багато етапів, таких як прийом посилок та зберігання їх у відділенні, транспортування їх та інших. На кожному з них можуть виникнути ситуації зіткнення з небезпечними або шкідливими факторами.

При роботі з накладними використовується комп'ютерна техніка. Використовуючи ЕРМ користувач стикається з фізичними та хімічними шкідливими і небезпечними виробничими факторами, до яких відносяться: підвищені рівні електромагнітного, рентгенівського, ультрафіолетового та інфрачервоного випромінювання, підвищений рівень статичної електрики і запиленості повітря робочої зони, підвищений рівень відпльовування і засліпленості, нерівномірність розподілу яскравості в полі зору, підвищена яскравість світлового зображення, підвищене значення напруги в електричному

ланцюзі, замикання якого може відбутися через тіло людини, підвищений вміст в повітрі робочої зони двоокису вуглецю та озону.

4.2. Розробка інструкції для забезпечення сприятливих умов праці при виробництві вітрил

Робоче місце – просторова зона, оснащена необхідними засобами, в якій здійснюється трудова діяльність робітника або групи робітників, які спільно виконують виробничі завдання. Робоче місце є частиною виробничо – технологічної структури підприємства (організації), воно призначене для виконання частини технологічного (виробничого) процесу та визначається на основі трудових та інших діючих норм та нормативів.

Під час роботи за автоматизованим робочим місцем та будь-яким ПК необхідно дотримуватись наступних правил, вимог пропонованих нормами охорони праці.

Під час виконання робіт за ПК на працюючого можуть діяти фізичні та психофізіологічні небезпечні та шкідливі фактори.

Робоче місце користувача зазвичай обладнано системним блоком, одним або декількома моніторами, клавіатурою, мишкою та принтером.

Згідно з нормами робоче місце повинно бути розташоване на відстані не менше ніж 1,5 м від стіни з вікнами, від інших стін на відстані 1м, між усіма потрібно дотримуватися відстань не менше 1,5 м. Відносно вікон робоче місце слід розміщувати так, щоб природне світло, яке надходить з вікна, падало на нього збоку, переважно зліва. Не слід розташовуватися прямо напроти джерела світла, це призведе до погіршення сприйняття інформації з екрану та змусить напружувати очі, щоб розглядіти на тьмяному моніторі картинку. Джерела освітлення варто розташовувати по бокам від екрану паралельно напрямку погляду. Щоб запобігти відблискам світла на моніторі, клавіатурі, від світильників у приміщенні або сонячного світла з вікна, необхідно

використовувати спеціальні окуляри з поляризаційним ефектом, анти блискові сітки, фільтри для екранів(але це застаріла технологія), захисні піддашки, а на вікнах – ролети, жалюзі або щільні фіранки.

Сітчасті фільтри з нейлону або металевої плетінки використовувати не варто та не рекомендується, така сітка псує та викривляє зображення через інтерференцію світла. Найкращу якість зображення надають скляні або пластикові захисні фільтри з поляризаційним ефектом. Вони забезпечують видалення усіх відблисків, роблять зображення чітким і контрастним, а також, завдяки цій технології, не дозволяють впливати шкідливому блакитному світінню.

При роботі з документами або текстом(в режимі введення даних та редагування, читання з екрану) найбільш правильним, та легшим для сприймання нашим зором – є зображення чорних знаків на світлому, трішки сірому фоні.

Монітор слід розташовувати на робочому місці так, щоб поверхня екрана знаходилася в центрі поля зору на відстані не менше 400мм від очей користувача, більш детально відстань можна вирахувати знаючи діагональ монітору. Елементи робочого місця потрібно розмістити так, щоб витримувалася однакова відстань очей від екрана, клавіатури, тексту, можливих записників, потрібних речей то що.

Фахівці в області ергономіки вже зрозуміли, що не можна знайти ідеальне положення, у якому можна перебувати і працювати протягом усього робочого дня. Для більшості людей комфортабельним робочим місцем повинне бути таке, котре можна пристосувати не менш чим для двох позицій, при цьому положення крісла, монітора повинні щораз відповідати виконуваній роботі і звичкам. Багато хто вважають, що для роботи на комп'ютері більше підходять

вертикальне і злегка похиле положення. Можливо, щоб крісло було злегка нахилене вперед.

Щоб зайняти робочу позу для роботи за комп'ютером потрібно відрегулювати висоту робочого столу, крісла та підставки для ніг, якщо така мається. Найоптимальнішою робочою позою вважається таке положення, при якому ступні працівника розташовані перпендикулярно гомілкам та горизонтально підлозі або підставці для ніг(якщо висота крісла занадто велика), стегна також повинні знаходитись горизонтально до полу, верхня частина рук – вертикально. Ліктьовий вигиб суглоба знаходиться в межах 70-90°, зап'ястя зігнуті під кутом не більше ніж 20°, нахил голови 15-20°.

Для запобігання накопиченню зарядів статичної електрики на поверхнях приміщення, де знаходяться та працюють комп'ютери, лазерні або світлодіодні принтери, рекомендується тримати вологість повітря біля 50%, це можна зробити за допомогою станції моніторингу вологості повітря. А також, не слід вдягати синтетичний одяг, він дуже легко накопичує заряд.

Згідно статті 18 Закону України «Про охорону праці» працівник зобов'язаний:

- знати і виконувати вимоги нормативних актів про охорону праці, правила поведінки з устаткуванням та іншими засобами виробництва, користуватися засобами колективного та індивідуального захисту;
 - співробітничати з власником у справі організації безпечних і нешкідливих умов праці, особисто вживати посильних заходів щодо усунення будь-якої виробничої ситуації, яка створює загрозу його життю чи здоров'ю, або людей, які його оточують, повідомляти про небезпеку 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.

Вимоги безпеки перед початком роботи:

- яскравість свічення екрана - не менше 100 kd/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.

Вимоги безпеки під час виконання роботи:

- Необхідно стійке розташовувати клавіатуру на робочому столі, не опускати її. 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 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;
- 1 раз на півроку потрібно відкривати процесор і вилучати пиłosосом пил і бруд, що накопичилися;
- Класти будь-які предмети на апаратуру комп'ютера;
- Закривати будь-чим вентиляційні отвори апаратури, це може призвести до її перегрівання і виходу з ладу.

Щоб розрядити накопичений статичний заряд, час від часу потрібно доторкатися до заземлених металевих поверхонь.

Принтер або сканер необхідно розташувати так, щоб з'єднувальний шнур був не натягнутий та не створював перешкод при роботі та переміщенні приміщенням, найкращий вибір – це принтер з бездротовим підключенням. На системний блок заборонено ставити будь що, в тому числі принтери та сканери.

Для досягнення найбільш чистих, з високою чіткістю зображень при печаті і щоб не зіпсувати механізми принтеру, має використовуватися папір щільності згідно інструкції. При зминанні, потрібно обережно, без зусиль витягнути пом'ятий папір, поправити та назад зарядити лоток .

Також згідно з інструкцією потрібно дотримуватися правил зберігання картриджів, на кшталт:

- Тримати без упаковки;
- Ставити вертикально;
- Перевертати етикеткою донизу;
- Відкривати кришку валика і доторкатися до нього;
- Заповнювати використаний картридж.

Вимоги безпеки закінчення роботи:

- Перевірити та зберегти на диск комп'ютера файли, що знаходяться в роботі;
- Вимкнути принтер та інші периферійні пристрої. Штепсельні вилки витягнути з розеток. Накрити клавіатуру кришкою запобігання попаданню в неї пилу;
- Прибрати від бруду та сторонніх речей робоче місце;
- Ретельно вимити руки теплою водою з милом, вмити лице прохолодною водою для зняття напруги очей;
- Вимкнути кондиціонер, освітлення і на загальному щитку електроживлення;
- Пройти в спеціально призначеному, обладнаному приміщенні сеанс психофізіологічного розвантаження і зняття втоми з виконанням спеціальних вправ аутогенного тренування [12].

Під час роботи зі складом та посилками потрібно забезпечити належний рівень вентиляції приміщення, так щоб концентрація шкідливих речовин не встигала перевищити гранично допустиму норму. Потрібно слідкувати за працездатністю витяжного пристрою, та завчасно проводити ремонтні роботи та очистку вентиляційних патрубків.

ВИСНОВОК

У даній дипломній роботі розроблено систему для оператора поштового відділення, який займається оформленням посилок, їх відправкою до інших відділень, приймання вантажу посилок та видачу посилок клієнтам. При розробці було проаналізовано предметну область поштового відділення, досліджено інформаційні процеси та структуру роботи оператора відділення. Знайдено недоліки і запропоновано ефективні вирішення проблем. Виконано дослідження об'єкта, аналіз інформаційних процесів, аналіз предметної області, формування вимог. Розроблено концепцію АРМ, технічний проект системи, проведено дослідження покращення умов праці та техніки безпеки. Для реалізації системи, базуючись на розробленому технічному проекті, було визначено сучасні програмні та технічні засоби, для розробки проекту: об'єктно орієнтована мова програмування Java, фреймворк Spring Boot, Spring MVC, Spring Security, Spring Data JPA, Hibernate, СКБД PostgreSQL, HTML 5, CSS/SCSS, Template engine Thymeleaf, сервер TomCat.

Реалізовано проект АРМ, який дозволяє вирішити поставлені задачі, такі як: формування накладної посилок, відстеження їх переміщень поміж відділеннями, облік посилок на складі, формування звітів про отримання посилок та звітів про прибуття посилок у відділення. Це підвищило ефективність та швидкість оформлення посилок, за рахунок раціонального використання часу на формування накладних та звітів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ГОСТ 19.701–90. УСЛОВНЫЕ ОБОЗНАЧЕНИЯ И ПРАВИЛА ВЫПОЛНЕНИЯ – Введено 01.01.1992.
2. ГОСТ 34.601–90. АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ СТАДИИ СОЗДАНИЯ – Введено 01.01.1992.
3. Что такое User friendly? [Блог]: Глосарій інтернет–маркетингу. – Режим звернення: https://www.glossary-internet.ru/terms/U/user_friendly/ – дата звернення: 25.04.2022.
4. Електронна версія програми побудови діаграм <https://www.draw.io/> – дата звернення 28.04.2022
5. Джошуа Блох. Эффективный Java;[В оригіналі] 2016. – 440 с.
6. Дейт К. Введение в системы баз данных: Пер. с англ. – К.:Диалектика,1998. –781 с.
7. Джексон Г. Проектирование реляционных баз данных для использования с ЭВМ: Пер. с англ. – М.: Мир, 1991. – 252 с.
8. Грекул В. И. Проектирование информационных систем. Практикум. М.: Национальный открытый университет «ИНТУИТ», 2012. – 159 с.
9. Вендров А. М. Проектирование программного обеспечения экономических информационных систем. М.: Фінанси та статистика, 2006. – 544 с.
10. UML [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим звернення: <https://ru.wikipedia.org/wiki/UML> – дата звернення: 08.04.2022.
11. CRUD [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим звернення: <https://ru.wikipedia.org/wiki/CRUD> – дата звернення: 08.04.2022.

12. ОСНОВНІ ПОНЯТТЯ В ГАЛУЗІ ОХОРОНИ ПРАЦІ, ЇЇ ТЕРМІНИ ТА ВИЗНАЧЕННЯ[Електронний ресурс]: Інфопедія. Вільна енциклопедія. – Режим звернення: <https://infopedia.su/17xca2b.html> – дата звернення: 03.05.2022.

13. Відповідальність за порушення законодавства про охорону праці[Електронний ресурс]: Інфопедія. Вільна енциклопедія. – Режим звернення: <https://infopedia.su/10x1e72.html> – дата звернення: 03.05.2022.

14. Техніка безпеки при роботі з ПК[Електронний ресурс]: Студфайл. Файловий архів. – Режим звернення: <https://studfile.net/preview/9743523/page:3/> – дата звернення: 08.05.2022.

ДОДАТКИ

Додаток А

Технічне завдання

1. ВСТУП

1.1. Назва – Автоматизоване робоче місце оператора поштового відділення

1.2. Призначення програми: автоматизація роботи системи поштового відділення засобами Web – технологій.

1.3. Область застосування: Використання у поштових відділеннях

2. Призначення розробки

Розробка призначена для автоматизації роботи оператора з посилками, клієнтами та роботи з доставкою посилок у поштовому відділенні

3. Вимоги до системи

3.1. Вимоги до функціональних характеристик

Система буде містити наступні функціональні характеристики:

- Додання інформації про посилки, клієнтів та історію перевезень до системи
- Відображення інформації про посилки та їх поточне знаходження на складі
- Формування накладних для посилок

4. Вимоги до надійності

4.1. Вимоги до забезпечення надійності функціонування програми

Надійне забезпечення на функціонування системи має бути забезпечено надійним сервером з резервним живленням, доступом до мережі інтернет та програмною системою захисту ОС та даних.

4.2. Відмова виконання програмного продукту через некоректні дії користувача.

- Перевірка формату вводу даних для коректної роботи

- Функція автоматичного резервування даних кожного вечора або вночі.

5. Умови експлуатації

Даною програмою будуть користуватися оператори поштового відділення які добре володіють комп'ютером та не потребують додаткового навчання по даному ПЗ.

6. Вимоги до інформаційної та програмної сумісності.

6.1. Вимоги до інформаційних структур і методів розв'язання.

Вимоги до інформаційних структур і методів розв'язання відсутні.

6.2. Вимоги до вихідних кодів і мов програмування.

Додаткові вимоги не пред'являються.

6.3. Вимоги до програмних засобів , які використовуються програмою.

Програмні засоби що використовуються мають бути вільно поширювані

6.4. Спеціальні вимоги

Спеціальні вимоги відсутні.

7. Вимоги до програмної документації

7.1. Попередній склад програмної документації повинен включати в себе:

- технічне завдання ;
- устав проекту.

7.2. Склад програмної документації.

Програмна документація складається з наступних програмних документів:

- Устав проекту;
- Технічне завдання ;
- Опис системи ;
- Код програми ;
- Керівництво користувача.

Технічне завдання містить опис завдання роботи , склад функціональних характеристик програми , опис вимог для розробки.

Опис програми містить опис логічної структури програми , форматів вхідних та вихідних даних , використовуваних технічних засобів.

Код програми містить опис блоків програми і запис блоків в програмному коді.

Керівництво користувача містить інформацію , необхідну для ознайомлення користувача з програмним продуктом

8. Техніко-економічні показники.

8.1.Економічні переваги розробки

Орієнтовна економічна ефективність не розраховується.

9. Стадії та етапи розробки програмного забезпечення.

9.1. Стадії розробки.

Розробка повинна бути проведена в 5 стадій:

1. Технічне завдання
2. Ескізний проект
3. Технічний проект
4. Робочий проект
5. Впровадження

Додаток Б

Програма і методика випробувань

1 Об'єкт випробувань

Назва об'єкта випробувань: «Security module» , надалі називається «модуль». Модуль дає можливість оператору пройти аутентифікацію та отримати доступ до інтерфейсу.

2 Ціль випробувань

Перевірка придатності програми до використання за призначенням, її відповідність заданим вимогам і програмним документам. Перевірка реально досягнутих характеристик програми.

3 Вимоги до програми

Програма повинна задовольняти наступним функціональним характеристикам:

- 1) перевірка правильності логіну та паролю користувача;
- 2) надання користувачеві доступу до свого інтерфейсу.

Користувач повинен вказати логін та пароль.

Вимоги до програмної документації

У комплект програмної документації повинні входити наступні документи:

- Загальний опис системи.
- Посібник користувача – оператора відділення.

- Інструкція з формування і ведення бази даних.
- Посібник адміністратора з установки програмного забезпечення на серверах.
- Програма і методика випробувань.
- Текст програми.

Засоби і порядок випробувань

Вимоги до складу і параметрів технічних засобів:

Сервер на базі комп'ютера с архітектурою x64 або ARM

Вимоги до складу і параметрів програмних засобів :

- 1) операційна система Ubuntu 20.04;
- 2) встановлене з'єднання з сервером баз даних Postgresql 11

Група людей, що будуть проводити випробування, повинна складатися з автора програми (розробника) і представника і/або самого замовника. При наявності відповідної можливості, ця група може бути розширена програмістом, спеціалістом-тестувальником і фахівцем-проектувальником, що не брали участь у розробці програми.

Попередньо (не пізніше 3 днів) до початку випробувань уточнюється кінцевий склад групи, що буде проводити випробування. Кожен член групи одержує необхідну документацію до програми (список вимог до програми, опис програми, керівництво користувача, програму і методику випробувань).

Порядок проведення випробувань:

- послідовно виконуються контрольні тести; результати заносяться у відповідний журнал;

- виробляється аналіз отриманих результатів і встановлюється відповідність програми заданим вимогам і програмним документам.

Методи випробувань

Робота програми залежить тільки від двох параметрів, що вводяться користувачем: від логіну та паролю оператора.

Іспити будуть проводитися методом «чорної шухляди».

Тести будуть створюватися на основі класів еквівалентності.

Класи еквівалентності виділяються шляхом вибору кожної вхідної умови і розбивкою його на двох або більш груп. Розрізняють два типи класів еквівалентності: правильні класи еквівалентності, що представляють правильні вхідні дані програми, і неправильні класи еквівалентності, що представляють всі інші можливі стани умов (помилкові вхідні значення). При цьому існує ряд правил:

- один тест може покривати тільки один неправильний клас еквівалентності;
- один тест повинний покривати як можна більше число непокритих правильних класів еквівалентності.

Виділені класи еквівалентності для складання тестів мають унікальні номери і приводяться в таблиці Д.1.

Таблиця Б.1 – Класи еквівалентності

Операція, що тестується	Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності

Перевірка логіну та паролю користувача	Ім'я правильне	Так (1)	Ні (2)
	Пароль правильний	Так (3)	Ні (4)
Надання користувачеві доступу до свого інтерфейсу	Належність користувача до інтерфейсу встановлена	Так (5)	Ні (6)

Тести

Тести складені на основі класів еквівалентності таблиці Д.1. Тести використовують наступні початкові умови:

1) користувач – user, його пароль – pass (власник товару);

Тест 1

Класи еквівалентності:

1, 3, 5

Вхідні дані:

Ім'я користувача: user

Пароль користувача: pass

Результат:

Ім'я та пароль користувача розпізнані. Належність встановлена. Наданий доступ до інтерфейсу оператора.

Тест 2

Класи еквівалентності:

2, 3, 5

Вхідні дані:

Ім'я користувача: users

Пароль користувача: pass

Результати:

Ім'я та пароль користувача не розпізнані. Належність до користувачів системи не встановлена. Видане повідомлення про помилку введення логіну чи паролю. Доступу до системи не надано.

Тест 3

Класи еквівалентності :

1, 4, 5

Вхідні дані:

Ім'я користувача: user

Пароль користувача: pas

Результати:

Логін та пароль користувача не розпізнані. Належність до користувачів системи не встановлена. Видане повідомлення про помилку введення імені чи паролю. Доступу до системи не надано.

Додаток В

Коди розробки

Код створення таблиць БД

Створення таблиці clients

```
create table clients
(
  id      bigint generated by default as identity,
  firstname varchar(255) not null,
  lastname varchar(255) not null,
  patronymic varchar(255) not null,
  phone   varchar(16) not null,
  primary key (id),
  unique (phone)
);
```

Створення таблиці positions

```
create table positions
(
  id      bigint generated by default as identity,
  position varchar(255) not null,
  primary key (id),
  unique (position)
);
```

Створення таблиці invoices

```
create table invoices
(
  id      bigint generated by default as identity,
  weight   double precision not null,
  volume   double precision not null,
  quantity bigint default 1 not null,
  type     varchar(255) default 'DELIVERY'::character varying,
  client_sender bigint not null,
  client_recipient bigint not null,
```

```
primary key (id),
constraint fk_client_sender_clients
foreign key (client_sender) references clients,
constraint fk_client_recipient_clients
foreign key (client_recipient) references clients
);
```

Створення таблиці stations

```
create table stations
(
id bigint generated by default as identity,
region varchar(255) not null,
city varchar(255) not null,
street varchar(255) not null,
number bigint not null,
primary key (id)
);
```

Створення таблиці employees

```
create table employees
(
id bigint generated by default as identity,
firstname varchar(255) not null,
lastname varchar(255) not null,
patronymic varchar(255) not null,
phone varchar(16) not null,
price double precision default 6500.00,
username varchar(255) not null,
employee varchar(255) not null,
password varchar(255) not null,
enabled boolean not null,
status varchar(255) default 'ACTIVE'::character varying not null,
role varchar(20) default 'USER'::character varying,
position_id bigint not null,
station_id bigint,
primary key (id),
```

```

unique (phone),
constraint fk_position_id_positions
    foreign key (position_id) references positions,
constraint fk_station_id_employees
    foreign key (station_id) references stations
);

```

Створення таблиці timelines

```

create table timelines
(
    id      bigint generated by default as identity,
    datetime timestamp not null,
    status  varchar(255) not null,
    station_id bigint not null,
    invoice_id bigint not null,
    primary key (id),
    constraint fk_station_id_stations
        foreign key (station_id) references stations,
    constraint fk_invoice_timelines_invoices
        foreign key (invoice_id) references invoices
);

```

Код розробки

Сутності

Сутність Client:

```

package com.post.terminal.model.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Builder
@AllArgsConstructor

```

```
@NoArgsConstructor
@Table(name = "clients")
public class Client {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private long id;
    @Basic
    @Column(name = "firstname")
    private String firstname;
    @Basic
    @Column(name = "lastname")
    private String lastname;
    @Basic
    @Column(name = "patronymic")
    private String patronymic;
    @Basic
    @Column(name = "phone")
    private String phone;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }

    public String getPatronymic() {
        return patronymic;
    }

    public void setPatronymic(String patronymic) {
        this.patronymic = patronymic;
    }

    public String getPhone() {
        return phone;
    }
}
```

```

    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Client client = (Client) o;

        if (id != client.id) return false;
        if (firstname != null ? !firstname.equals(client.firstname) : client.firstname != null) return false;
        if (lastname != null ? !lastname.equals(client.lastname) : client.lastname != null) return false;
        if (patronymic != null ? !patronymic.equals(client.patronymic) : client.patronymic != null) return false;
        if (phone != null ? !phone.equals(client.phone) : client.phone != null) return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = (int) (id ^ (id >>> 32));
        result = 31 * result + (firstname != null ? firstname.hashCode() : 0);
        result = 31 * result + (lastname != null ? lastname.hashCode() : 0);
        result = 31 * result + (patronymic != null ? patronymic.hashCode() : 0);
        result = 31 * result + (phone != null ? phone.hashCode() : 0);
        return result;
    }

    @Override
    public String toString() {
        return "Client{" +
            "id=" + id +
            ", firstname=" + firstname + "\"" +
            ", lastname=" + lastname + "\"" +
            ", patronymic=" + patronymic + "\"" +
            ", phone=" + phone + "\"" +
            "}";
    }
}

```

Сутність Employee:

```

package com.post.terminal.model.entity;

import javax.persistence.*;

@Entity
@Table(name = "employees")
public class Employee {

```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Id
@Column(name = "id")
private long id;
@Basic
@Column(name = "firstname")
private String firstname;
@Basic
@Column(name = "lastname")
private String lastname;
@Basic
@Column(name = "patronymic")
private String patronymic;
@Basic
@Column(name = "phone")
private String phone;
@Basic
@Column(name = "price")
private Double price;
@Basic
@Column(name = "username")
private String username;
@Basic
@Column(name = "employee")
private String employee;
@Basic
@Column(name = "password")
private String password;
@Basic
@Column(name = "enabled")
private boolean enabled;
@Basic
@Column(name = "status")
private String status;
@Basic
@Column(name = "role")
private String role;
@Basic
@Column(name = "position_id")
private long positionId;
@Basic
@Column(name = "station_id")
private Long stationId;

public long getId() {
    return id;
}

public void setId(long id) {
    this.id = id;
}

public String getFirstname() {
    return firstname;
}
```



```
public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}

public String getPatronymic() {
    return patronymic;
}

public void setPatronymic(String patronymic) {
    this.patronymic = patronymic;
}

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone;
}

public Double getPrice() {
    return price;
}

public void setPrice(Double price) {
    this.price = price;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getEmployee() {
    return employee;
}

public void setEmployee(String employee) {
    this.employee = employee;
}

public String getPassword() {
    return password;
}
```

```

}

public void setPassword(String password) {
    this.password = password;
}

public boolean isEnabled() {
    return enabled;
}

public void setEnabled(boolean enabled) {
    this.enabled = enabled;
}

public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

public String getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}

public long getPositionId() {
    return positionId;
}

public void setPositionId(long positionId) {
    this.positionId = positionId;
}

public Long getStationId() {
    return stationId;
}

public void setStationId(Long stationId) {
    this.stationId = stationId;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Employee employee = (Employee) o;

    if (id != employee.id) return false;
    if (enabled != employee.enabled) return false;

```

```

    if (positionId != employee.positionId) return false;
    if (firstname != null ? !firstname.equals(employee.firstname) : employee.firstname != null) return false;
    if (lastname != null ? !lastname.equals(employee.lastname) : employee.lastname != null) return false;
    if (patronymic != null ? !patronymic.equals(employee.patronymic) : employee.patronymic != null) return
false;
    if (phone != null ? !phone.equals(employee.phone) : employee.phone != null) return false;
    if (price != null ? !price.equals(employee.price) : employee.price != null) return false;
    if (username != null ? !username.equals(employee.username) : employee.username != null) return false;
    if (this.employee != null ? !this.employee.equals(employee.employee) : employee.employee != null) return
false;
    if (password != null ? !password.equals(employee.password) : employee.password != null) return false;
    if (status != null ? !status.equals(employee.status) : employee.status != null) return false;
    if (role != null ? !role.equals(employee.role) : employee.role != null) return false;
    if (stationId != null ? !stationId.equals(employee.stationId) : employee.stationId != null) return false;

    return true;
}

@Override
public int hashCode() {
    int result = (int) (id ^ (id >>> 32));
    result = 31 * result + (firstname != null ? firstname.hashCode() : 0);
    result = 31 * result + (lastname != null ? lastname.hashCode() : 0);
    result = 31 * result + (patronymic != null ? patronymic.hashCode() : 0);
    result = 31 * result + (phone != null ? phone.hashCode() : 0);
    result = 31 * result + (price != null ? price.hashCode() : 0);
    result = 31 * result + (username != null ? username.hashCode() : 0);
    result = 31 * result + (employee != null ? employee.hashCode() : 0);
    result = 31 * result + (password != null ? password.hashCode() : 0);
    result = 31 * result + (enabled ? 1 : 0);
    result = 31 * result + (status != null ? status.hashCode() : 0);
    result = 31 * result + (role != null ? role.hashCode() : 0);
    result = 31 * result + (int) (positionId ^ (positionId >>> 32));
    result = 31 * result + (stationId != null ? stationId.hashCode() : 0);
    return result;
}
}

```

Сутність Invoice:

```
package com.post.terminal.model.entity;
```

```
import lombok.Builder;
```

```
import lombok.NoArgsConstructor;
```

```
import javax.persistence.*;
```

```
@Entity
```

```
@Table(name = "invoices")
```

```
@Builder
```

```
@NoArgsConstructor
```

```
public class Invoice {
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    @Id
```

```

@Column(name = "id")
private long id;
@Basic
@Column(name = "weight")
private double weight;
@Basic
@Column(name = "volume")
private double volume;
@Basic
@Column(name = "quantity")
private long quantity;
@Basic
@Column(name = "type")
private String type;
@Basic
@Column(name = "client_sender")
private long clientSender;
@Basic
@Column(name = "client_recipient")
private long clientRecipient;

public Invoice(long id, double weight, double volume, long quantity, String type, long clientSender, long
clientRecipient) {
    this.id = id;
    this.weight = weight;
    this.volume = volume;
    this.quantity = quantity;
    this.type = type;
    this.clientSender = clientSender;
    this.clientRecipient = clientRecipient;
}

public long getId() {
    return id;
}

public void setId(long id) {
    this.id = id;
}

public double getWeight() {
    return weight;
}

public void setWeight(double weight) {
    this.weight = weight;
}

public double getVolume() {
    return volume;
}

public void setVolume(double volume) {
    this.volume = volume;
}

```

```

public long getQuantity() {
    return quantity;
}

public void setQuantity(long quantity) {
    this.quantity = quantity;
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public long getClientSender() {
    return clientSender;
}

public void setClientSender(long clientSender) {
    this.clientSender = clientSender;
}

public long getClientRecipient() {
    return clientRecipient;
}

public void setClientRecipient(long clientRecipient) {
    this.clientRecipient = clientRecipient;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Invoice invoices = (Invoice) o;

    if (id != invoices.id) return false;
    if (Double.compare(invoices.weight, weight) != 0) return false;
    if (Double.compare(invoices.volume, volume) != 0) return false;
    if (quantity != invoices.quantity) return false;
    if (clientSender != invoices.clientSender) return false;
    if (clientRecipient != invoices.clientRecipient) return false;
    if (type != null ? !type.equals(invoices.type) : invoices.type != null) return false;

    return true;
}

@Override
public int hashCode() {
    int result;
    long temp;

```

```

        result = (int) (id ^ (id >>> 32));
        temp = Double.doubleToLongBits(weight);
        result = 31 * result + (int) (temp ^ (temp >>> 32));
        temp = Double.doubleToLongBits(volume);
        result = 31 * result + (int) (temp ^ (temp >>> 32));
        result = 31 * result + (int) (quantity ^ (quantity >>> 32));
        result = 31 * result + (type != null ? type.hashCode() : 0);
        result = 31 * result + (int) (clientSender ^ (clientSender >>> 32));
        result = 31 * result + (int) (clientRecipient ^ (clientRecipient >>> 32));
        return result;
    }

    @Override
    public String toString() {
        return "Invoice{" +
            "id=" + id +
            ", weight=" + weight +
            ", volume=" + volume +
            ", quantity=" + quantity +
            ", type=" + type + '\n' +
            '}';
    }
}

```

Сутність Position:

```

package com.post.terminal.model.entity;

import javax.persistence.*;

@Entity
@Table(name = "positions")
public class Position {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private long id;
    @Basic
    @Column(name = "position")
    private String position;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {

```

```

    this.position = position;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Position position = (Position) o;

    if (id != position.id) return false;
    if (this.position != null ? !this.position.equals(position.position) : position.position != null) return false;

    return true;
}

@Override
public int hashCode() {
    int result = (int) (id ^ (id >>> 32));
    result = 31 * result + (position != null ? position.hashCode() : 0);
    return result;
}
}

```

Станція:

```

package com.post.terminal.model.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Table(name = "stations")
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Station {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private long id;
    @Basic
    @Column(name = "region")
    private String region;
    @Basic
    @Column(name = "city")
    private String city;
    @Basic
    @Column(name = "street")
    private String street;
    @Basic

```

```

@Column(name = "number")
private long number;

public long getId() {
    return id;
}

public void setId(long id) {
    this.id = id;
}

public String getRegion() {
    return region;
}

public void setRegion(String region) {
    this.region = region;
}

public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

public String getStreet() {
    return street;
}

public void setStreet(String street) {
    this.street = street;
}

public long getNumber() {
    return number;
}

public void setNumber(long number) {
    this.number = number;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Station station = (Station) o;

    if (id != station.id) return false;
    if (number != station.number) return false;
    if (region != null ? !region.equals(station.region) : station.region != null) return false;
    if (city != null ? !city.equals(station.city) : station.city != null) return false;
    if (street != null ? !street.equals(station.street) : station.street != null) return false;

```



```

        return true;
    }

    @Override
    public int hashCode() {
        int result = (int) (id ^ (id >>> 32));
        result = 31 * result + (region != null ? region.hashCode() : 0);
        result = 31 * result + (city != null ? city.hashCode() : 0);
        result = 31 * result + (street != null ? street.hashCode() : 0);
        result = 31 * result + (int) (number ^ (number >>> 32));
        return result;
    }
}

```

Сутність Timeline:

```

package com.post.terminal.model.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import java.sql.Timestamp;

@Entity
@Table(name = "timelines")
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class TimeLine {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "id")
    private long id;
    @Basic
    @Column(name = "datetime")
    private Timestamp datetime;
    @Basic
    @Column(name = "status")
    private String status;
    @Basic
    @Column(name = "station_id")
    private long stationId;
    @Basic
    @Column(name = "invoice_id")
    private long invoiceId;

    public long getId() {
        return id;
    }

    public void setId(long id) {

```

```

    this.id = id;
}

public Timestamp getDatetime() {
    return datetime;
}

public void setDatetime(Timestamp datetime) {
    this.datetime = datetime;
}

public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

public long getStationId() {
    return stationId;
}

public void setStationId(long stationId) {
    this.stationId = stationId;
}

public long getInvoiceId() {
    return invoiceId;
}

public void setInvoiceId(long invoiceId) {
    this.invoiceId = invoiceId;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    TimeLine timelines = (TimeLine) o;

    if (id != timelines.id) return false;
    if (stationId != timelines.stationId) return false;
    if (invoiceId != timelines.invoiceId) return false;
    if (datetime != null ? !datetime.equals(timelines.datetime) : timelines.datetime != null) return false;
    if (status != null ? !status.equals(timelines.status) : timelines.status != null) return false;

    return true;
}

@Override
public int hashCode() {
    int result = (int) (id ^ (id >>> 32));
    result = 31 * result + (datetime != null ? datetime.hashCode() : 0);
}

```

```

        result = 31 * result + (status != null ? status.hashCode() : 0);
        result = 31 * result + (int) (stationId ^ (stationId >>> 32));
        result = 31 * result + (int) (invoiceId ^ (invoiceId >>> 32));
        return result;
    }

    @Override
    public String toString() {
        return "TimeLine{" +
            "id=" + id +
            ", datetime=" + datetime +
            ", status='" + status + '\'' +
            ", stationId=" + stationId +
            ", invoiceId=" + invoiceId +
            '}';
    }
}

```

Репозиторії для доступу до БД

Репозиторій Client:

```

@Repository
public interface ClientRepository extends JpaRepository<Client, Long> {

    Collection<Client> findAllByFirstnameContainingOrLastnameContainingOrPatronymicContaining(String
firstname, String lastname, String patronymic);

    Collection<Client> findAllByPhoneContaining(String phone);

    Optional<Client> findByFirstnameAndLastnameAndPatronymic(String firstname, String lastname, String
patronymic);
}

```

Репозиторій Employee:

```

package com.post.terminal.model.repository;

import com.post.terminal.model.entity.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

```

```
Optional<Employee> findByUsername(String username);
}
```

Репозиторій Invoice:

```
package com.post.terminal.model.repository;
```

```
import com.post.terminal.model.entity.Invoice;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
```

```
import java.util.Collection;
```

```
@Repository
```

```
public interface InvoiceRepository extends JpaRepository<Invoice, Long>
{
```

```
    @Query(value = "select i.id,i.weight,i.volume,i.quantity,i.type,i.client_sender,i.client_recipient from invoices as i inner join timelines t on i.id = t.invoice_id where station_id = ?1 and datetime between ?2 and ?3 group by i.id;", nativeQuery = true)
```

```
    Collection<Invoice> findAllByStationId(Long stationId, String startDay, String endDay);
```

```
    @Query(value = "select i.id,i.weight,i.volume,i.quantity,i.type,i.client_sender,i.client_recipient from invoices as i inner join timelines t on i.id = t.invoice_id where station_id = ?1 group by i.id;", nativeQuery = true)
```

```
    Collection<Invoice> findAllByStationId(Long stationId);
```

```
    Collection<Invoice> findAllByClientSender(long clientSenderId);
```

```
    Collection<Invoice> findAllByClientRecipient(long clientRecipientId);
```

```
    Collection<Invoice> findAllByClientSenderOrClientRecipient(long clientSenderId, long clientRecipientId);
```

```
}
```

Репозиторій Position:

```
package com.post.terminal.model.repository;
```

```
import com.post.terminal.model.entity.Position;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
import java.util.Collection;
```

```
import java.util.Optional;
```

```
@Repository
```

```
public interface PositionRepository extends JpaRepository<Position, Long> {
```

```
    Collection<Position> findAllByPositionContaining(String position);
```

```
    Optional<Position> findByPosition(String position);
```

```
}
```

Репозиторій Station:

```
package com.post.terminal.model.repository;

import com.post.terminal.model.entity.Station;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Collection;
import java.util.Optional;

@Repository
public interface StationRepository extends JpaRepository<Station, Long>
{
    Collection<Station>
    findAllByRegionContainingOrCityContainingOrStreetContainingOrNumberContaining(String region,
        String city, String street, long number);

    Optional<Station> findByRegionAndCityAndStreetAndNumber(String region, String city, String street, long
number);
}
```

Репозиторій Timeline:

```
package com.post.terminal.model.repository;

import com.post.terminal.model.entity.TimeLine;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Collection;

@Repository
public interface TimelineRepository extends JpaRepository<TimeLine, Long> {

    Collection<TimeLine> findAllByInvoiceId(Long invoiceId);
}
```

Контроллери

Головний контроллер MainPageController:

```
package com.post.terminal.rest;

import com.post.terminal.log.Log;
import com.post.terminal.users.employee.operator.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
```

```

@Controller
@RequestMapping("/")
public class MainPageController {

    private final EmployeeService employeeService;

    @Autowired
    public MainPageController(@Qualifier("employeeService") EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    @GetMapping
    public String getMain() {
        Log.log("GET /", this.getClass().getName());
        return "index";
    }

    @GetMapping("/user")
    public String user(Model model) {
        model.addAttribute("employee", employeeService.getEmployee());

        Log.log("GET /user", this.getClass().getName());
        return "user/home";
    }

    @GetMapping("/driver")
    public String driver(Model model) {
        model.addAttribute("employee", employeeService.getEmployee());
        Log.log("GET /driver", this.getClass().getName());
        return "user/employee/driver/driver";
    }
}

```

Модуль аутентифікації та авторизації користувачів. Далі модуль Security.

Моделі

Модель статусів користувача CredentialStatus:

```

package com.post.terminal.security.model;

public enum CredentialStatus
{
    ACTIVE, BANDED
}

```

Модель перерахувати прав та дозволень користувача Permission:

```
package com.post.terminal.security.model;
```

```
public enum Permission
{
    USER_READ("user:read"),
    USER_WRITE("user:write"),
    ADMIN_READ("admin:read"),
    ADMIN_WRITE("admin:write"),
    OPERATOR_READ("operator:read"),
    OPERATOR_WRITE("operator:write"),
    EMPLOYEE_READ("employee:read"),
    EMPLOYEE_WRITE("employee:write");

    private final String permission;

    Permission(String permission)
    {
        this.permission = permission;
    }

    public String getPermission()
    {
        return permission;
    }
}
```

Модель ролей користувача Role:

```
package com.post.terminal.security.model;
```

```
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
```

```
import java.util.Arrays;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;
```

```
public enum Role
{
    VIEWER(Set.of(Permission.USER_READ)),
    USER(Set.of(Permission.USER_READ, Permission.USER_WRITE)),
    ADMIN(Set.of(Permission.ADMIN_READ, Permission.ADMIN_WRITE)),
    OPERATOR(Set.of(Permission.OPERATOR_READ, Permission.OPERATOR_WRITE)),
    EMPLOYEE(Set.of(Permission.EMPLOYEE_READ, Permission.EMPLOYEE_WRITE)),
    DRIVER(Set.of(Permission.EMPLOYEE_READ, Permission.EMPLOYEE_WRITE)),
    ;

    private final Set<Permission> permissions;

    Role(Set<Permission> permissions)
    {

```

```

    this.permissions = permissions;
}

public static Set<SimpleGrantedAuthority> find(String role)
{
    Optional<Role> first = Arrays.stream(Role.values()).filter(role1 -> role1.name().equals(role)).findFirst();
    if ( first.isPresent() )
    {
        return first.get().grantedAuthorities();
    }
    throw new UsernameNotFoundException("Role does not exists");
}

public Set<Permission> getPermissions()
{
    return permissions;
}

public Permission[] getArrayPermissions()
{
    return getPermissions().toArray(new Permission[0]);
}

public String[] getPermissionsLikeStringArray()
{
    return getPermissions().stream().map(Permission::getPermission).collect(Collectors.toSet())
        .toArray(new String[getPermissions().size()]);
}

public Set<SimpleGrantedAuthority> grantedAuthorities()
{
    return getPermissions().stream().map(permission -> new
SimpleGrantedAuthority(permission.getPermission()))
        .collect(
            Collectors.toSet());
}
}

```

Модель користувача для Security модуля SecurityUser:

```

package com.post.terminal.security.model;

import com.post.terminal.log.Log;
import com.post.terminal.model.entity.Employee;
import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;
import java.util.List;

```


@Data

public class SecurityUser implements UserDetails

{

private final boolean isActive;

private final boolean enabled;

List<SimpleGrantedAuthority> authorities;

private String username;

private String password;

public static UserDetails fromUserCredentials(Employee employee)

{

Log.log("fromUserCredentials() started", SecurityUser.class.getName());

User user = new User{

employee.getUsername(),

employee.getPassword(),

employee.getStatus().equals(CredentialStatus.ACTIVE.name()),

employee.getStatus().equals(CredentialStatus.ACTIVE.name()),

employee.getStatus().equals(CredentialStatus.ACTIVE.name()),

employee.getStatus().equals(CredentialStatus.ACTIVE.name()),

Role.find(employee.getRole())

);

Log.log("fromUserCredentials() ended, " + user, SecurityUser.class.getName());

return user;

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities()

{

return authorities;

}

@Override

public String getPassword()

{

return password;

}

@Override

public String getUsername()

{

return username;

}

@Override

public boolean isAccountNonExpired()

{

return isActive;

}

@Override

public boolean isAccountNonLocked()

{

return isActive;

}

```

@Override
public boolean isCredentialsNonExpired()
{
    return isActive;
}

@Override
public boolean isEnabled()
{
    return enabled;
}
}

```

Конфігурація модуля Security

Конфігурація SecurityConfig:

```
package com.post.terminal.security.config;
```

```

import com.post.terminal.log.Log;
import com.post.terminal.security.model.Role;
import org.apache.commons.lang3.ArrayUtils;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.boot.autoconfigure.security.servlet.PathRequest;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

```

```
@Configuration
```

```
@EnableWebSecurity
```

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

```
    private final UserDetailsService userDetailsService;
```

```
    private final AuthenticationSuccessHandler authenticationSuccessHandler;
```

```
    public SecurityConfig(
```

```
        @Qualifier(value = "customAuthenticationSuccessHandler") AuthenticationSuccessHandler
        authenticationSuccessHandler,
```

```
        @Qualifier(value = "userDetailsServiceImpl") UserDetailsService userDetailsService) {
```

```
        this.authenticationSuccessHandler = authenticationSuccessHandler;
```

```
        this.userDetailsService = userDetailsService;
```

```
    }
```

@Override

```
protected void configure(AuthenticationManagerBuilder auth) {  
    Log.log("configure() AuthenticationManagerBuilder started configuring", this.getClass().getName());  
    auth.authenticationProvider(daoAuthenticationProvider());  
    Log.log("configure() AuthenticationManagerBuilder ended configuring", this.getClass().getName());  
}
```

@Bean

```
protected DaoAuthenticationProvider daoAuthenticationProvider() {  
    Log.log("daoAuthenticationProvider() started configuring", this.getClass().getName());  
    DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();  
    daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());  
    System.out.println(passwordEncoder().encode("admin"));  
    daoAuthenticationProvider.setUserDetailsService(userDetailsService());  
    Log.log("daoAuthenticationProvider() ended configuring", this.getClass().getName());  
    return daoAuthenticationProvider;  
}
```

@Bean

```
protected PasswordEncoder passwordEncoder() {  
    BCryptPasswordEncoder bCryptPasswordEncoder = new  
BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2A, 12);  
    String encode = bCryptPasswordEncoder.encode("admin");  
    Log.log("Configuring password encoder: $2A type 12 difficult, (admin:" + encode + ") is: " +  
bCryptPasswordEncoder.matches("admin", encode), this.getClass().getName());  
    return bCryptPasswordEncoder;  
}
```

@Override

```
protected void configure(HttpSecurity http) throws Exception {  
    Log.log("configure(), HttpSecurity start configuring", this.getClass().getName());  
    http  
        .csrf()  
        .disable()  
        .authorizeRequests()  
        .antMatchers("/resources/**", "/view/**", "/", "/signup", "/about")  
        .permitAll()  
        .antMatchers("/panel/**").hasAnyAuthority(Role.ADMIN.getPermissionsLikeStringArray())  
  
    .antMatchers("/storage/**").hasAnyAuthority(ArrayUtils.addAll(Role.ADMIN.getPermissionsLikeStringArray(),  
Role.OPERATOR.getPermissionsLikeStringArray()))  
        .antMatchers("/statistics/**").hasAnyAuthority(ArrayUtils.addAll(  
            Role.ADMIN.getPermissionsLikeStringArray(),  
            ArrayUtils.addAll(  
                Role.EMPLOYEE.getPermissionsLikeStringArray(),  
                Role.OPERATOR.getPermissionsLikeStringArray())))  
        .anyRequest()  
        .authenticated()  
        .and()  
        .formLogin()  
        .loginPage("/auth/login").permitAll()  
        .successHandler(authenticationSuccessHandler)  
        .failureUrl("/auth/error")  
        .and()
```

```

        .logout()
        .logoutRequestMatcher(new AntPathRequestMatcher("/auth/logout", "POST"))
        .invalidateHttpSession(true)
        .clearAuthentication(true)
        .deleteCookies("JSESSIONID")
        .logoutSuccessUrl("/auth/login");
Log.log("configure(), HttpSecurity end configuring", this.getClass().getName());
    }
}

```

Конфігурація обробника аутентифікованих устаткованих запитів CustomAuthenticationSuccessHandler:

```
package com.post.terminal.security.config;
```

```

import com.post.terminal.log.Log;
import com.post.terminal.security.model.Permission;
import com.post.terminal.security.model.Role;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;

```

```

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Arrays;
import java.util.Collection;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.Stream;

```

```
@Configuration
```

```

public class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandler {
    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
Authentication authentication) throws IOException, ServletException {

```

```

        Set<String> roles = AuthorityUtils.authorityListToSet(authentication.getAuthorities()).stream().map(s ->
s.split(":")[0].toUpperCase()).collect(Collectors.toSet());

```

```
        System.out.println(authentication.isAuthenticated());
```

```
        Log.log(roles.stream().reduce("", (s, s2) -> s + ", " + s2), this.getClass().getName());
```

```

        if (roles.contains(Role.VIEWER.name())) {
            response.sendRedirect("/view");
        }

```

```

        if (roles.contains(Role.USER.name())) {
            response.sendRedirect("/user");
        }

```

```

        if (roles.contains(Role.EMPLOYEE.name())) {
            response.sendRedirect("/employee");
        }
        if (roles.contains(Role.OPERATOR.name())) {
            response.sendRedirect("/storage");
        }
        if (roles.contains(Role.ADMIN.name())) {
            response.sendRedirect("/panel");
        }
    }
}

```

Сервіси

Сервіс для обробки даних користувачів UserDetailsService:

```

package com.post.terminal.security.service;

import com.post.terminal.log.Log;
import com.post.terminal.model.entity.Employee;
import com.post.terminal.model.repository.EmployeeRepository;
import com.post.terminal.security.model.SecurityUser;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class UserDetailsServiceImpl implements UserDetailsService
{
    private final EmployeeRepository userRepository;

    @Autowired
    public UserDetailsServiceImpl(EmployeeRepository userRepository)
    {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
    {
        Log.log("loadUserByUsername() before",this.getClass().getName());
        Employee credential = userRepository.findByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("User not found with this email"));
        Log.log("loadUserByUsername() after",this.getClass().getName(), credential.toString());

        return SecurityUser.fromUserCredentials(credential);
    }
}

```

Контроллери

Контроллер аутентифікації AuthenticationController:

```
package com.post.terminal.security.rest;

import com.post.terminal.log.Log;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/auth")
public class AuthenticationController {

    @GetMapping("/login")
    public String getLogin(Model model) {
        model.addAttribute("title", "Login");
        Log.log("GET /login", this.getClass().getName());
        return "auth/login";
    }

    @GetMapping("/success")
    public String getSuccess(Model model) {
        model.addAttribute("title", "Success");
        Log.log("GET /success", this.getClass().getName());
        return "auth/success";
    }

    @GetMapping("/error")
    public String getError(Model model) {
        model.addAttribute("title", "Error");
        Log.log("GET /error", this.getClass().getName());
        return "auth/error";
    }

    @GetMapping("/logout")
    public String getLogout(Model model) {
        model.addAttribute("title", "Logout");
        Log.log("GET /logout", this.getClass().getName());
        return "auth/logout";
    }
}
```

Модуль логіки для роботи з відділенням

Сервіси

Клієнтський сервіс ClientService:

```
package com.post.terminal.users.employee.operator.service;
```

```

import com.post.terminal.model.dto.ClientDto;
import com.post.terminal.model.entity.Client;
import com.post.terminal.model.repository.ClientRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Collection;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class ClientService {
    private final ClientRepository clientRepository;

    @Autowired
    public ClientService(ClientRepository clientRepository) {
        this.clientRepository = clientRepository;
    }

    public Collection<ClientDto> findAll() {
        return clientRepository.findAll().stream().map(ClientDto::fromClient).collect(Collectors.toList());
    }

    public ClientDto findById(long id) {
        Optional<Client> byId = clientRepository.findById(id);
        if (byId.isPresent()) {
            return ClientDto.fromClient(byId.get());
        }
        throw new IllegalArgumentException("Id is not contains in data base");
    }

    public Collection<ClientDto> getClients() {
        return clientRepository.findAll().stream().map(ClientDto::fromClient).collect(Collectors.toList());
    }

    public Client saveClient(ClientDto clientDto) {
        if (clientDto == null) {
            throw new IllegalArgumentException("ClientDto is null");
        }
        Client client = new Client();
        client.setFirstname(clientDto.getFirstname());
        client.setLastname(clientDto.getLastname());
        client.setPatronymic(clientDto.getPatronymic());
        client.setPhone(clientDto.getPhone());
        return clientRepository.save(client);
    }

    public Collection<ClientDto> findAllLikeNameOrLastnameOrPatronymic(String s) {
        return clientRepository.findAllByFirstnameContainingOrLastnameContainingOrPatronymicContaining(s, s, s).stream().map(ClientDto::fromClient).collect(Collectors.toList());
    }

    public Collection<ClientDto> findAllLikePhone(String s) {
        return
        clientRepository.findAllByPhoneContaining(s).stream().map(ClientDto::fromClient).collect(Collectors.toList());
    }
}

```

```

public ClientDto findByFullName(String fullName, String separator) {
    String[] split = fullName.split(separator);
    if (split.length!=3){
        throw new IllegalArgumentException("Full name isn't full or can't to be parse");
    }
    return ClientDto.fromClient(clientRepository.findByFirstnameAndLastnameAndPatronymic(split[0],
split[1], split[2]).get());
}
}

```

Сервіс робітників EmployeeService:

```
package com.post.terminal.users.employee.operator.service;
```

```

import com.post.terminal.log.Log;
import com.post.terminal.model.dto.EmployeeDto;
import com.post.terminal.model.entity.Employee;
import com.post.terminal.model.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

```

```

import java.util.Collection;
import java.util.Optional;
import java.util.function.Supplier;
import java.util.stream.Collectors;

```

```
@Service
```

```
public class EmployeeService {
```

```

    private static EmployeeDto employeeDto = null;
    private static Employee employee;
    private final EmployeeRepository employeeRepository;

```

```
@Autowired
```

```

public EmployeeService(EmployeeRepository employeeRepository) {
    this.employeeRepository = employeeRepository;
}

```

```

public EmployeeDto getEmployee() {
    if (employeeDto == null) {
        employeeDto = findEmployeeDto();
    }
    return employeeDto;
}

```

```

public Employee getEmployeeEntity() {
    if (employee == null) {
        employeeDto = findEmployeeDto();
    }
}

```



```

    }
    return employee;
}

private EmployeeDto findEmployeeDto() {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    System.out.println(authentication.getName());
    Object principal = authentication.getPrincipal();
    String username = "User";
    Optional<Employee> employee;
    if (principal instanceof UserDetails) {
        UserDetails details = (UserDetails) principal;
        username = (details).getUsername();
        Log.log(username, this.getClass().getName());
        employee = employeeRepository.findByUsername(username);
        if (employee.isPresent()) {
            EmployeeService.employee = employee.get();
            Log.log("getEmployee() Passwords: " + details.getPassword() + " : " + employee.get().getPassword(),
this.getClass().getName());
            return EmployeeDto.fromEmployee(employee.get());
        }
    }
    Log.log("Exception, getEmployee", this.getClass().getName());
    throw new UsernameNotFoundException("User not found:" + username);
}

public Collection<EmployeeDto> findAll() {
    return
employeeRepository.findAll().stream().map(EmployeeDto::fromEmployee).collect(Collectors.toList());
}

public EmployeeDto findById(long id) {
    Optional<Employee> byId = employeeRepository.findById(id);
    if (byId.isPresent()) {
        return EmployeeDto.fromEmployee(byId.get());
    }
    throw new IllegalArgumentException("Employee with this id is not contains in data base: "+id);
}

public EmployeeDto findByUsername(String username) {
    Optional<Employee> byUsername = employeeRepository.findByUsername(username);
    if (byUsername.isPresent()) {
        return EmployeeDto.fromEmployee(byUsername.get());
    }
    throw new IllegalArgumentException("Employee with this username is not contains in data base: " +
username);
}
}

```

Сервіс для роботи з посилками InvoiceService:

```
package com.post.terminal.users.employee.operator.storage.invoice.service;
```

```
import com.post.terminal.model.dto.*;
import com.post.terminal.model.entity.Client;
import com.post.terminal.model.entity.Invoice;
import com.post.terminal.model.entity.Station;
import com.post.terminal.model.entity.TimeLine;
import com.post.terminal.model.entity.status.TimelineStatus;
import com.post.terminal.model.repository.ClientRepository;
import com.post.terminal.model.repository.InvoiceRepository;
import com.post.terminal.model.repository.StationRepository;
import com.post.terminal.model.repository.TimelineRepository;
import com.post.terminal.users.employee.operator.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import java.sql.Timestamp;
import java.util.*;
import java.util.stream.Collectors;
```

```
@Service
```

```
public class InvoiceService {
```

```
    private final EmployeeService employeeService;
    private final InvoiceRepository invoiceRepository;
    private final TimelineRepository timelineRepository;
    private final ClientRepository clientRepository;
    private final StationRepository stationRepository;
```

```
@Autowired
```

```
    public InvoiceService(EmployeeService employeeService, InvoiceRepository invoiceRepository,
        TimelineRepository timelineRepository, ClientRepository clientRepository, StationRepository
        stationRepository) {
        this.employeeService = employeeService;
        this.invoiceRepository = invoiceRepository;
        this.timelineRepository = timelineRepository;
        this.clientRepository = clientRepository;
        this.stationRepository = stationRepository;
    }
```

```
    public Map<Invoice, TimeLine> getInvoices() {
        Collection<Invoice> allByStationId =
invoiceRepository.findAllByStationId(employeeService.getEmployeeEntity().getStationId());
        Map<Invoice, TimeLine> invoiceTimeLineMap = new HashMap<>();
        allByStationId.forEach(invoice -> {
            long id = invoice.getId();
            List<TimeLine> allByInvoiceId = (List<TimeLine>) timelineRepository.findAllByInvoiceId(id);
            if (!allByInvoiceId.isEmpty()) {
                int size = allByInvoiceId.size();
                TimeLine timeLine = allByInvoiceId.get(size - 1);
                if (!timeLine.getStatus().equals(TimelineStatus.RECEIVED.name()) ||
                    !timeLine.getStatus().equals(TimelineStatus.IN_TRANSIT_TO.name())) {
                    invoiceTimeLineMap.put(invoice, timeLine);
                }
            }
        });
    }
```

```

    }
    });
    return invoiceTimeLineMap;
}

public Collection<FullInvoiceDto> getFullInvoices() {
    Map<Invoice, TimeLine> invoices = getInvoices();
    Collection<FullInvoiceDto> invoicesCollection = new ArrayList<>();
    invoices.entrySet().stream().distinct().forEach(invoiceTimeLineEntry -> {
        Invoice invoice = invoiceTimeLineEntry.getKey();
        TimeLine timeLine = invoiceTimeLineEntry.getValue();
        Optional<Client> recipient = clientRepository.findById(invoice.getClientRecipient());
        Optional<Client> sender = clientRepository.findById(invoice.getClientSender());
        if (recipient.isPresent() && sender.isPresent()) {
            invoicesCollection.add(FullInvoiceDto.builder()
                .invoiceDTO(InvoiceDTO.fromInvoice(invoice))
                .timeLineDto(TimeLineDto.fromTimeLine(timeLine))
                .recipient(ClientDto.fromClient(recipient.get()))
                .recipient(ClientDto.fromClient(sender.get()))
                .build());
        }
    });
    return invoicesCollection;
}

public Collection<InvoiceDTO> findAll() {
    return invoiceRepository.findAll().stream().map(InvoiceDTO::fromInvoice).collect(Collectors.toList());
}

public InvoiceDTO findById(long id) {
    Optional<Invoice> byId = invoiceRepository.findById(id);
    if (byId.isPresent()) {
        return InvoiceDTO.fromInvoice(byId.get());
    }
    throw new IllegalArgumentException("Invoice with this id had not found: " + byId);
}

public InvoiceDTO saveInvoice(InvoiceDTO invoiceDTO, ClientDto clientSender, ClientDto clientRecipient,
    StationDto stationDto) {
    Client sender;
    Client recipient;
    Station station;
    Invoice invoice;
    Optional<Client> clientSend =
        clientRepository.findByFirstnameAndLastnameAndPatronymic(clientSender.getFirstname(),
            clientSender.getLastname(), clientSender.getPatronymic());
    Optional<Client> clientRecipe =
        clientRepository.findByFirstnameAndLastnameAndPatronymic(clientRecipient.getFirstname(),
            clientRecipient.getLastname(), clientRecipient.getPatronymic());
    String[] split = stationDto.getFullStationLocation().strip().split(",");
    Optional<Station> byRegionAndCityAndStreetAndNumber =
        stationRepository.findByRegionAndCityAndStreetAndNumber(split[0], split[1], split[2],
            Long.getLong(split[3]));
}

```

```

if (clientSend.isEmpty()) {
    sender = clientRepository.save(Client.builder()
        .phone(clientSender.getPhone())
        .patronymic(clientSender.getPatronymic())
        .lastname(clientSender.getLastname())
        .firstname(clientSender.getFirstname()).build());
} else {
    sender = clientSend.get();
}

if (clientRecipe.isEmpty()) {
    recipient = clientRepository.save(Client.builder()
        .phone(clientRecipient.getPhone())
        .patronymic(clientRecipient.getPatronymic())
        .lastname(clientRecipient.getLastname())
        .firstname(clientRecipient.getFirstname()).build());
} else {
    recipient = clientRecipe.get();
}

if (byRegionAndCityAndStreetAndNumber.isEmpty()) {
    station = stationRepository.save(Station.builder()
        .region(split[0])
        .city(split[1])
        .street(split[2])
        .number(Long.getLong(split[3]))
        .build());
} else {
    station = byRegionAndCityAndStreetAndNumber.get();
}

invoice = new Invoice();
invoice.setClientSender(sender.getId());
invoice.setClientRecipient(recipient.getId());
invoice.setType(invoiceDTO.getType());
invoice.setWeight(invoiceDTO.getWeight());
invoice.setVolume(invoiceDTO.getVolume());
invoice.setQuantity(invoiceDTO.getQuantity());

invoice = invoiceRepository.save(invoice);

List<TimeLine> timeLines = new ArrayList<>();

timeLines.add(TimeLine.builder()
    .invoiceId(invoice.getId())
    .stationId(station.getId())
    .datetime(new Timestamp(System.currentTimeMillis()))
    .status(TimelineStatus.CREATED.name())
    .build());

timeLines.add(TimeLine.builder()
    .invoiceId(invoice.getId())
    .stationId(station.getId())
    .datetime(new Timestamp(System.currentTimeMillis()))
    .status(TimelineStatus.ON_STATION.name())

```

```

        .build());

    timelineRepository.saveAll(timeLines);

    return InvoiceDTO.fromInvoice(invoice);
}
}

```

Сервіс для роботи із змінами TimeShiftService:

```

package com.post.terminal.users.employee.operator.storage.timeshift.service;

import com.post.terminal.model.entity.Invoice;
import com.post.terminal.model.entity.TimeLine;
import com.post.terminal.model.entity.status.TimelineStatus;
import com.post.terminal.users.employee.operator.storage.invoice.service.InvoiceService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Map;

@Service
public class TimeShiftService {

    private final InvoiceService invoiceService;

    @Autowired
    public TimeShiftService(InvoiceService invoiceService) {
        this.invoiceService = invoiceService;
    }

    public boolean isTimeShiftOpened() {
        Map<Invoice, TimeLine> invoices = invoiceService.getInvoices();
        invoices.values().forEach(System.out::println);
        return invoices.values().stream().map(TimeLine::getStatus).anyMatch(s ->
s.equals(TimelineStatus.OPEN_STATION.name()));
    }
}

```

Контроллери

Контроллер складу з посилками InvoiceController:

```

package com.post.terminal.users.employee.operator.storage.invoice.rest;

import com.post.terminal.model.dto.ClientDto;
import com.post.terminal.model.dto.InvoiceDTO;
import com.post.terminal.model.dto.StationDto;
import com.post.terminal.model.entity.Client;

```

```
import com.post.terminal.model.entity.Invoice;
import com.post.terminal.model.entity.TimeLine;
import com.post.terminal.users.employee.operator.service.ClientService;
import com.post.terminal.users.employee.operator.service.EmployeeService;
import com.post.terminal.users.employee.operator.storage.invoice.service.InvoiceService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import java.util.Map;
import java.util.stream.Stream;
```

```
@Controller
@RequestMapping("/storage")
public class InvoiceController {
```

```
    private final EmployeeService employeeService;
    private final InvoiceService invoiceService;
    private final ClientService clientService;
```

```
    @Autowired
    public InvoiceController(EmployeeService employeeService, InvoiceService invoiceService, ClientService
clientService) {
        this.employeeService = employeeService;
        this.invoiceService = invoiceService;
        this.clientService = clientService;
    }
```

```
    @GetMapping("/invoices")
    public String getInvoices(Model model) {
        model.addAttribute("employee", employeeService.getEmployee());
        model.addAttribute("invoices", invoiceService.getFullInvoices());
        return "user/employee/operator/invoice/invoices";
    }
```

```
    @GetMapping("/invoice/create")
    public String getFormToCreateInvoice(Model model) {
        model.addAttribute("employee", employeeService.getEmployee());
        model.addAttribute("clientSender", new ClientDto());
        model.addAttribute("clientRecipient", new ClientDto());
        model.addAttribute("invoice", new InvoiceDTO());
        model.addAttribute("station", new StationDto());
        return "user/employee/operator/invoice/create-invoice";
    }
```

```
    @PostMapping("/invoice/create")
    public String createInvoice(
        ClientDto clientSender,
```

```

        ClientDto clientRecipient,
        StationDto stationDto,
        InvoiceDTO invoiceDTO
    ){
        invoiceService.saveInvoice(invoiceDTO, clientSender, clientRecipient, stationDto);
        return "user/employee/operator/invoice/create-invoice";
    }
}

```

Контроллер обработки запитів після успішного входу до системи

BaseStorageController:

```

package com.post.terminal.users.employee.operator.storage.rest;

import com.post.terminal.log.Log;
import com.post.terminal.users.employee.operator.service.EmployeeService;
import com.post.terminal.users.employee.operator.storage.timeshift.service.TimeShiftService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/storage")
public class BaseStorageController {

    private final EmployeeService employeeService;
    private final TimeShiftService timeShiftService;

    @Autowired
    public BaseStorageController(EmployeeService employeeService, TimeShiftService timeShiftService) {
        this.employeeService = employeeService;
        this.timeShiftService = timeShiftService;
    }

    @GetMapping
    public String getStorage(Model model) {

        model.addAttribute("employee", employeeService.getEmployee());

        Log.log("GET /storage", this.getClass().getName());
        if (timeShiftService.isTimeShiftOpened()) {
            return "redirect:/storage/invoices";
        }
        return "user/employee/operator/empty-shift";
    }
}

```

Контроллер змін TimeShiftController:

```
package com.post.terminal.users.employee.operator.storage.timeshift.rest;

import com.post.terminal.log.Log;
import com.post.terminal.users.employee.operator.service.EmployeeService;
import com.post.terminal.users.employee.operator.storage.timeshift.service.TimeShiftService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/storage/shift")
public class TimeShiftController {

    private final TimeShiftService timeShiftService;
    private final EmployeeService employeeService;

    @Autowired
    public TimeShiftController(TimeShiftService timeShiftService, EmployeeService employeeService) {
        this.timeShiftService = timeShiftService;
        this.employeeService = employeeService;
    }

    @GetMapping
    public String getShift(Model model) {
        model.addAttribute("title", "Storage");
        model.addAttribute("employee", employeeService.getEmployee());

        Log.log("GET /storage/shift", this.getClass().getName());
        if (timeShiftService.isTimeShiftOpened()) {
            return "user/employee/operator/invoice/invoices";
        }
        return "user/employee/operator/empty-shift";
    }

    @GetMapping("/open")
    public String getOpenShift(Model model) {
        model.addAttribute("title", "Open shift");
        model.addAttribute("employee", employeeService.getEmployee());

        Log.log("GET /storage/shift/open", this.getClass().getName());
        return "user/employee/operator/time-shift/open-shift";
    }

    @GetMapping("/close")
    public String getCloseShift(Model model) {
        model.addAttribute("title", "Close shift");
        model.addAttribute("employee", employeeService.getEmployee());
    }
}
```



```
Log.log("GET /storage/shift/close", this.getClass().getName());  
return "user/employee/operator/time-shift/close-shift";  
}  
}
```