МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Система підтримки прийняття рішень забезпечення функціональної стійкості організаційної системи

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма «**Комп'ютерні науки»**

Освітній рівень: бакалавр

Виконала: студентка 4 курсу, групи КН- 41

Білоног А.І.
(прізвище та ініціали)

Керівник Гнатієнко Г. М.
(прізвище та ініціали)

к.т.н.
(науковий ступінь, звання)

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *інтелектуальних технологій*
Протокол № 11 від 06.06.2022 р.
зав. кафедри_____ доц. Іларіонов О.Є.

Київ - 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.

_____

"___"_____2022 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Білоног Анні Ігорівні

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)

Система підтримки прийняття рішень забезпечення функціональної стійкості організаційної системи

затверджена протоколом засідання кафедри від « 23 » грудня 2021 р. № 4

2. Термін здачі студентом закінченого проекту (роботи) 29 травня 2022 року

3. Вихідні дані до проекту (роботи)

Система підтримки прийняття рішень забезпечення функціональної стійкості організаційної системи створюється для визначення варіантів забезпечення стійкого функціонування організаційної системи та оцінки інтегральної якості функціонування системи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

Вибір та аналіз предметної області, огляд задач функціональної стійкості організаційної системи, проєктування архітектури системи, розробка системи, вибір тестових задач, візуалізація результатів, тестування роботи системи

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)

Огляд джерел інформації щодо задач функціональної стійкості організаційної системи, ілюстрація ситуації прийняття рішення, архітектура системи, програмна реалізація системи, презентація Power Point

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Вибір та аналіз предметної області		
2	Постановка задачі функціональної стійкості організаційної системи		
2	Огляд підходів до забезпечення функціональної стійкості організаційної системи		
3	Проєктування архітектури системи		
3	Програмна реалізація системи		
3	Вибір тестових задач, тестування роботи системи		

7. Дата видачі завдання 15 лютого 2022 року

Керівник _____ / /
(підпис) (ПІБ)

Завдання прийняв до виконання _____ / /
(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1.	Вибір та аналіз предметної області	25.01-01.02	
2.	Постановка задачі функціональної стійкості організаційної системи	02.02-04.02	
3.	Огляд підходів до забезпечення функціональної стійкості організаційної системи	05.02-14.02	
4.	Проєктування архітектури системи	15.02-07.03	
5.	Програмна реалізація системи	08.03-25.04	
6.	Вибір тестових задач, тестування роботи системи	26.04-07.05	

Студент-дипломник _____ / /
(підпис) (ПІБ)

Керівник випускної кваліфікаційної роботи _____ / /
(підпис) (ПІБ)

Анотація

Білоног Анна Ігорівна виконала випускну кваліфікаційну роботу на тему «Система підтримки прийняття рішень забезпечення функціональної стійкості організаційної системи» за спеціальністю 122 – «Комп'ютерні науки».

У випускній кваліфікаційній роботі проведено аналіз методів забезпечення функціональної стійкості, виконано розробку алгоритму знаходження заміни відсутньому працівнику з урахуванням функціональної стійкості організаційної системи, розроблено базу даних, спроектовано архітектуру веб-додатку та його реалізацію.

Ключові слова: система підтримки прийняття рішень, функціональна стійкість, організаційна система, економічна безпека.

Summary

The degree project: «Decision support system for ensuring the functional stability of the organizational system» has completed by **Anna Bilonoh** specialty 122 – «Computer Scienses».

In this graduation thesis the methods of ensuring functional stability are analyzed, an algorithm for finding a replacement for the absent employee, taking into account the functional stability of the organizational system is developed, database is developed, the architecture of the web application and its implementation is designed.

Keywords: decision support system, functional stability, organizational system, economic security.

ЗМІСТ

ВСТУП

Функціональна стійкість організаційної системи пов’язана з безпекою організації, що є однією з головних цінностей підприємства. Сьогодні, в умовах ринкових відносин, забезпечення безпеки підприємства є однією з найбільш важливих і актуальних проблем його діяльності. Головним інструментом успішного, ефективного та стійкого функціонування підприємства є забезпечення ефективної системи заходів безпеки.

Важливе значення має саме економічна безпека організаційної системи. В свою чергу економічна безпека включає в себе інформаційну, фінансову, інтелектуальну, кадрову безпеку та інші. Дотримання економічної безпеки важливо необхідне для забезпечення стійкості функціонування та досягнення головних цілей діяльності організаційної системи. Рівень економічної безпеки організаційної системи залежить від ефективності прийнятих керівництвом рішень щодо уникнення можливих загроз та врегулювання шкідливих наслідків збурень зовнішнього та внутрішнього середовища. Стійкість розвитку та функціонування, конкурентоспроможність та економічна ефективність діяльності підприємства напряму залежить від рівня економічної безпеки. Тому для забезпечення ефективної діяльності бізнесу важливим є забезпечення економічної безпеки підприємства, що в свою чергу тягне за собою забезпечення його функціональної стійкості. Отже економічна успішність нерозривно пов’язана з економічною безпекою, тому керівники підприємств мають бути зацікавлені в забезпеченні функціональної стійкості.

Метою розроблюваної системи підтримки прийняття рішень буде створення додатку, для знаходження заміни відсутньому працівнику, враховуючи функціональну стійкість організаційної системи, тобто організації або підприємства.

Об’єктом дослідження є забезпечення функціональної стійкості організаційної системи

Предметом дослідження є алгоритм знаходження заміни відсутньому працівнику з урахуванням функціональної стійкості

Однією з важливих задач дипломної роботи буде розробка алгоритму знаходження заміни працівнику, враховуючи показники функціональної стійкості. Наступною задачею є розробка системи, зручної і зрозумілої для користувача.

В подальшому розроблений додаток можна буде використовувати в організаціях, де кількість працівників дуже велика, а тому не зручно і не можливо підібрати заміну працівнику, щоб це не призвело до втрат.

РОЗДІЛ 1 ОПИС ФУНКЦІОНАЛЬНОЇ СТІЙКОСТІ ОРГАНІЗАЦІЙНОЇ СИСТЕМИ

1.1. Аналітичний огляд задачі та підходів до забезпечення функціональної стійкості організаційності системи

1.1.1. Поняття організаційної системи

Організаційною системою називають об’єднання людей (учасників), які разом реалізують деяку програму та діють на основі узгоджених правил [1]; упорядковану сукупність служб, відділів, підрозділів і окремих посадових осіб, що знаходяться у взаємозв'язку і співпідпорядкованості і виконують певні управлінські функції. Організаційна система володіє внутрішньою ієрархією та структурою [2]. Для покращення ефективності функціонування організаційної системи сучасні компанії використовують інформаційні технології та системи.

Механізми функціонування та взаємодії елементів організаційної системи задаються сукупністю процедур, законів та нормативних правил, що регламентують взаємодію учасників організаційної системи. Ці механізми не є чітко регламентованими та можуть змінюватися від системи до системи. Проста організаційна система не має властивості емерджентності (неможливість зведення властивостей системи до суми властивостей її компонентів) [7], оскільки її можливості залежать лише від кадрового потенціалу учасників. Оскільки поведінка людей залежить від їх миттєвої мотивації і може змінюватися в залежності від ментальних та психофізичних факторів, то моделювання такої системи містить елементи ймовірнісного аналізу. Використання моделей дозволяє передбачати ймовірну поведінку організаційної системи [3].

Розрізняють також складну організаційну систему. На відміну від простої організаційної системи складна організаційна система – це система, яка має властивостями цілісності та емерджентності. Оскільки організаційна

система складна, то вона має властивості складної системи описані в теорії складних систем [2]. В основі опису такої системи лежать певні системні принципи [3]:

- принцип цілісності системи, який означає з’язність елементів системи в загальну сукупність і неможливість видалення будь-якого елемента без втрат властивостей системи. Принцип цілісності системи допускає надлишковість елементів.

- принцип єдності, що полягає в спільному розгляді системи як цілого і як сукупності частин (елементів). Принцип дає змогу виконати декомпозицію системи, зберігаючи уявлення про систему як про цілісність.

- принцип емерджентності полягає в тому, що будь-яка система має властивості не притаманні її елементам та підсистемам. Ці властивості зумовлюються системними з’язками.

- принцип структурності полягає у впорядкованості системи, певному наборі та розташуванні елементів із зв'язками між ними.

- принцип функціональності системи, що полягає у наявності загальної функції системи та функцій її елементів;

- принцип ієрархічності будови полягає у підпорядкуванні елементів нижчого рівня елементам вищого рівня. Будь-яка організація є взаємодією двох підсистем: керуючої та керованої. Одна підпорядковується іншій.

Важливою характеристикою складної організаційної системи є наявність людського капіталу та техніки (інформаційних систем, електронного документообігу).

Складні системи, в тому числі і складні організаційні системи, мають ряд функціональних характеристик, таких як ефективність, надійність, якість управління та стійкість. Під ефективністю складної системи розуміють величину (числова характеристика), що характеризує рівень пристосованості системи до виконання поставлених перед нею завдань. За допомогою показника ефективності складної системи оцінюють якість виконання цільової функції та вартість витрат, що пішли на її досягнення. Якість

управління напряму пов’язана з показником ефективності, ціллю управління є збільшення ефективності функціонування системи, та з стійкістю роботи системи при наявності помилок. Під стійкістю функціонування складної системи розуміють можливість системи зберігати потрібні властивості в умовах дії збурень [2].

1.1.2. Поняття функціональної стійкості

Функціональна стійкість – це здатність системи виконувати свої функції впродовж заданого інтервалу часу за умови впливу на неї потоку експлуатаційних відмов, навмисних пошкоджень, втручання в обмін і обробку інформації, а також у разі помилок обслуговуючого персоналу. Функціональна стійкість тісно пов'язана з властивостями стійкості, надійності, живучості і відмовостійкості. Вперше термін «функціональна стійкість» вжив професор Машков О. А.

Для досягнення функціональної стійкості при виникненні позаштатних ситуацій, які мають негативні наслідки для складної системи, потрібно застосовувати існуючу надмірність системи та виконати перерозподіл ресурсів. Найважливішим є те, що не повинна вводитись додаткова надмірність в систему, а має проводитись розподіл уже існуючих ресурсів [5]. Важливо відмітити, що забезпечення функціональної стійкості не зосереджене на зменшенні кількості відмов складної системи, а на забезпечення виконання найважливіших функцій складної системи та неперервність бізнес-процесів організаційної системи, коли відмови вже відбулися [6].

Задачі забезпечення функціональної стійкості спрямовані на оптимальне використання на кожному етапі функціонування складної системи всіх наявних ресурсів для досягнення головної мети цього етапу, дотримуючись обмежень, тому їх відносять до задач адаптивного оптимального управління [6].

На сьогодні тема функціональної стійкості складних систем активно досліджується та розвивається українськими науковцями. На даний момент

існують дослідження стосовно забезпечення функціональної стійкості інтелектуалізованої системи автоматичного управління польотом літака, мобільних систем, функціональної стійкості автоматизованої системи управління повітряним рухом, функціональної стійкості мультисервісних комп'ютерних мереж, функціональної стійкості інформаційно-телекомунікаційних мереж. Однією із задач забезпечення функціональної стійкості є забезпечення функціональної стійкості організаційної системи, що й буде розглянута в даній роботі.

1.1.3. Підходи для забезпечення функціональної стійкості організаційної системи

Першим кроком у забезпеченні функціональної стійкості організаційної системи є визначення структури організаційної системи та побудова її моделі (моделювання організаційної системи). Цей крок можна також розглядати як процес прийняття рішень, оскільки потрібно визначити підпорядкованість елементів один одному, взаємозв’язок та взаємозаміну елементів організаційної системи.

Оскільки організаційна система має ієрархічну структуру та функції, виконувані елементами системи не є незалежними, то при моделюванні організаційної системи потрібно враховувати наступні моменти:

- рівень управління елемента;
- підлеглість елемента;
- функції елемента та рівень якості виконання цих функцій;
- рівень впливу елемента на інший елемент, на підрозділ до якого він належить, на підрозділ до якого він не належить, на систему;
- взаємодію між елементами відповідно до виконуваних ними функцій;
- встановлення ієрархічних зв’язків між елементами та підсистемами.

Таким чином, організаційну систему можна подати у вигляді орієнтованого графа, вершини якого є елементами системи і які містять вартість виконання елементом функції та оцінку якості виконання цієї функцій, а на ребрах графа задано характеристики послідовності та взаємозв'язків між елементами організаційної системи. Орграф також зручно подавати у вигляді матриці суміжності.

Для побудови моделі організаційної системи для визначення рівня управління, впливу та критичність елемента будується матриця відповідальності. Ця матриця містить бізнес-процеси по одній осі та виконавців (ролі) по іншій. На перетині рядків і стовпців вказуються повноваження, ступінь участі і розподіл відповідальності за виконання кожного процесу, тобто вноситься один з кодів Responsible (R) (Виконавець), Accountable (A) (Який затверджує - відповідає за кінцевий результат перед вищим керівництвом), Consulted (C) (Узгоджувальний - погоджує прийняті рішення), Informed (I) (Спостерігач - його інформують про вже прийняте рішення) (найрозповсюдженіша матриця відповідальності RACI).

Наступним важливим кроком є побудова інструменту визначення оцінки якості виконання функції елементом. Такий інструмент повинен враховувати зміни у структурі організаційної системи при заміні та перерозподілі елементів. Якість виконання функцій може задаватись функцією належності $\mu_{ij}(x), i \in J, j \in J^i, x < 100\%, J^i$ - множина індексів функцій, що належать підмножині функцій конкретного елемента системи. При цьому варто зазначити, що при перерозподілі функцій між елементами втрачається якість виконання їх штатних функцій.

Після визначення якості виконання кожної функції рахується інтегральний показник функціонування організаційної системи. Це є одним з найважливіших етапів задачі, оскільки після нього приймається рішення.

Для визначення інтегральної оцінки будуємо матрицю частот різних рівнів якості виконання функцій $V = (v_{ij}), i = 1, \ldots, 100, j < n$. Кожен рядок цієї матриці відображує оцінений рівень якості виконання функції від 0% до

100%, а стовпчик – кількість функцій з зазначеним рівнем якості виконання. Далі здійснюємо класифікацію функцій за рівнем якості та повноти виконання. Таким чином, використовуючи адитивний критерій, отримуємо інтегральний показник функціонування організаційної системи.

І останнім етапом є прийняття рішення. При цьому ОПР може прийняти одне з наступних рішень:

- розподілити виконання функції між кількома елементами системи;
- передати функцію на виконання одному елементу системи, який при цьому виконує свої штатні функції;
- проігнорувати виконання функції, якщо це не значно вплине на функціональну стійкість системи.

1.2. Постановка задачі забезпечення функціональної стійкості організаційної системи

1.2.1. Задача випускної кваліфікаційної роботи

В даній роботі буде розглянуто розробка СППР забезпечення функціональної стійкості організаційної системи, а саме у ситуації відсутності працівника та забезпечення неперервності виконання бізнес-процесів, що є важливим аспектом економічної та інформаційної безпеки організації. Детальніше цю задачу можна описати так: є працівники, які в кожен поточний момент часу знаходяться на лікарняному, у відпустці, у відрядженні, звільнені з роботи, порушують трудову дисципліну, проходять адаптацію і тому недостатньо якісно та ефективно виконують поставлені перед ними задачі та функції організації, підприємства тощо, тому для забезпечення функціональної стійкості організаційної системи потрібно прийняти рішення про передання виконання цих функцій іншому працівнику.

Вимоги до розроблюваної системи:

Функціональні вимоги:

- користувач повинен мати можливість вводити дані елементів (працівників, підрозділів), визначаючи їхнє місце в ієрархії організаційної системи, їхню підпорядкованість.
- користувач повинен мати можливість переглядати дані про елементи організаційної системи, їхню характеристику;
- система повинна надати користувачу відповідні варіанти для прийняття рішення щодо заміни елемента системи.

Нефункціональні вимоги:

- система має бути захищеною, доступ до неї можуть мати тільки авторизовані користувачі;
- система має бути зручною та простою у використанні;
- збереження даних працівників у системі не повинно порушувати законодавство України збереження й обробку персональних даних.

1.2.2. Математична модель

Нехай маємо n функцій, які мають виконуватись елементами (працівниками, підрозділами) організаційної системи. Тоді множину усіх функцій, які виконуються елементами організаційної системи позначимо через $A = \{a_1, \dots, a_n\}$, множину усіх індексів функцій позначимо через $J = \{1, \dots, n\}$. Кожна функція організаційної системи є унікальною, тобто $A^{i_1} \cap A^{i_2} = \emptyset, i_2, i_2 \in J$, де $\emptyset$ - порожня множина, а отже $n = \sum_{i \in J} n_i$.

Зв'язок між функціями, виконуваними в системі, та послідовність їх виконання задамо бінарним відношенням B, яке є підмножиною декартового добутку $A \times A$. Побудоване бінарне відношення відображає послідовність та логіку цілей, які стоять перед елементами організаційної системи. Кожна функція організаційної системи виконується деяким елементом організаційною системи $e^i, i \in I = \{1, \dots, k\}$ і може бути виконана деяким іншим працівником $e^j, i \neq j,\ i, j \in I = \{1, \dots, k\}$ з різними ступенями якості.

Нехай маємо деякий інструмент, який дає змогу оцінити якість виконання функцій елементами організаційної системи. Таким чином маємо змогу оцінити поточний рівень виконання кожної функції організаційної системи деяким елементом та потенційну якість виконання функції деяким іншим елементом. При оцінці потенційної якості виконання функції можемо оцінити та врахувати:

- вартість виконання кожної функції певним елементом системи;
- якість виконання функцій іншим елементом системи;
- вартість витрат на заміну елемента системи;
- вартість та час на процедуру адаптації елемента системи;
- вартість витрат при повторному виконанні функцій організаційної системи.

РОЗДІЛ 2 РОЗРОБКА АРХІТЕКТУРИ СППР ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНАЛЬНОЇ СТІЙКОСТІ ОРГАНІЗАЦІЙНОЇ СИСТЕМИ

2.1. Модель використання СППР забезпечення функціональної стійкості організаційної системи

Детально розглянемо використання СППР. Для цього було побудовано діаграму прецедентів.

Було виділено наступних акторів:

- Користувач системи – попередньо зареєстрований користувач сппр, який може переглядати інформацію про працівників та вводити інформацію про їх відсутність;
- Менеджер з персоналу – користувач сппр, який наслідує всі варіанти використання «Користувача системи», має розширені права, тільки він може редагувати БД, вносити дані про працівників, додавати нових працівників, приймати рішення про заміну працівника;
- Менеджер підрозділу - користувач сппр, який наслідує всі варіанти використання «Користувача системи».

Було виділено наступні варіанти використання (прецеденти):

- Авторизація – підтвердження для входу в систему;
- Введення даних про відсутність працівника – введення дат відсутності та занесення запису про відсутність до БД;
- Отримання детальної інформації про працівника;
- Управління БД включає в себе:
  - Введення даних про працівників організації;
  - Введення даних про якість виконання функцій;
  - Редагування даних;
  - Видалення даних;

- Отримання варіантів для прийняття рішення – для знаходження заміни працівнику система формує варіанти, з яких ОПР (Менеджер з персоналу) обирає;

- Прийняття рішення про заміну вимагає виконання попереднього прецеденту:

- Обрання варіанту із запропонованих;

- Збереження в БД – після прийняття і обрання рішення, записуємо його в БД.

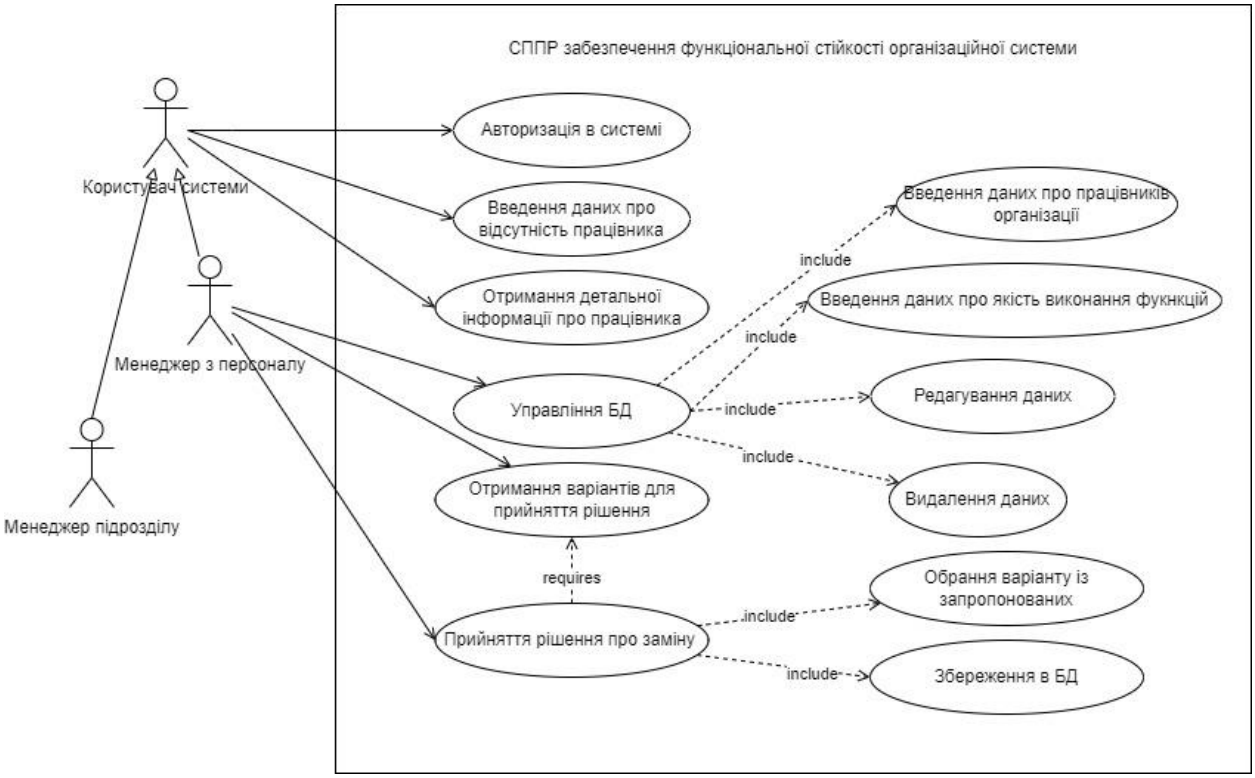

Рис. 1 – Діаграма прецедентів

2.2. Моделювання процесів роботи СППР у нотації IDEF0

Загалом СППР можна представити у вигляді діаграми IDEF0 (рис. 2).

На вхід системи поступають:

- дані про відсутність працівника: ПІБ та дати відсутності;

Елементи керування:

- База даних – інформація про працівників;

- Правила бази знань – правила за допомогою яких підбирається підходящі працівники для заміни відсутнього;

- Оцінки виконання функцій працівником – оцінки, за якими відбувається обрахунок функціональної стійкості організаційної системи.

Механізми:

- Менеджер з підрозділу, який вносить дані про відсутність працівника;

- Менеджер з персоналу – запускає систему на виконання для знаходження найкращих варіантів заміни відсутнього працівника;

- Модуль підбору варіантів, за допомогою бази знань та оцінок виконання функцій працівником обчислює функціональну стійкість для усіх можливих варіантів заміни та повертає найкращий.

Рис. 2 - Діаграма IDEF0

На рис. 3 зображено декомпозицію діаграми IDEF0. На ній деталізований процес роботи системи підтримки прийняття рішень. Як видно з діаграми IDEF0 на вхід ми отримуємо інформацію про відсутність працівника. Після цього або менеджер з персоналу, або керівник департаменту вводять цю інформацію в базу даних. Далі менеджер з персоналу запускає систему на виконання. За переліком функцій працівника,

що планує бути відсутнім відбувається підбір можливих кандидатів для його заміни. Отримуємо перелік кандидатів, передаючи функції яким обчислюємо функціональну стійкість організації. Отримуємо найкращі значення функціональної стійкості з яких потім формуються варіанти заміни відсутнього працівника. Отже, отримуємо варіанти рішень про заміну працівника.

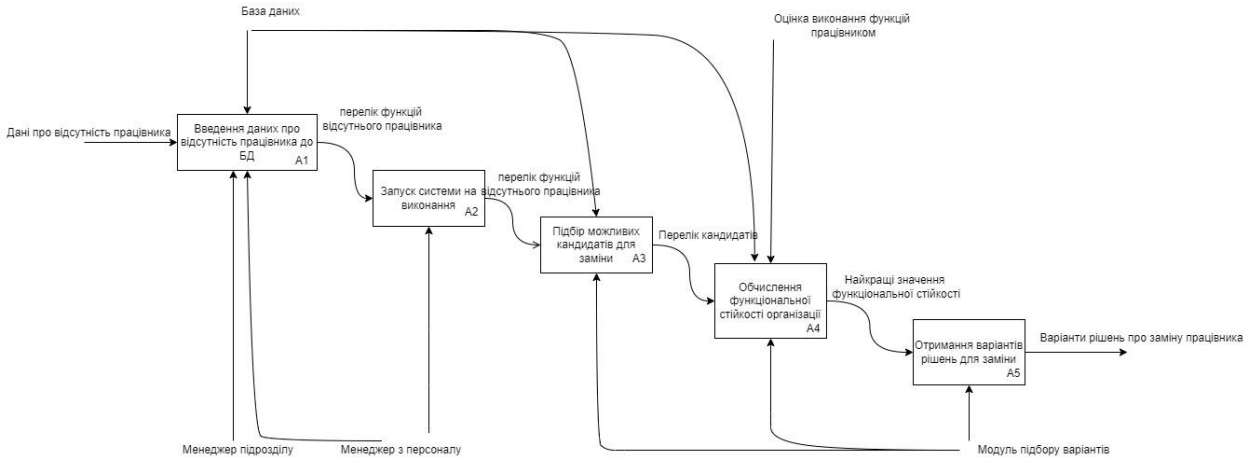

Рис. 3 – Декомпозиція IDEF0

2.4. Алгоритм визначення функціональної стійкості організаційної системи

Загалом процес прийняття рішення про заміну відсутнього працівника можна описати наступним чином:

1. Менеджер з персоналу відкриває в розробленому застосунку «журнал присутності», який містить перелік працівників, що будуть відсутні.
2. Зі списку обирає працівника якому б хотів знайти заміну.
3. Після обрання працівника система підбирає всі можливі варіанти та обчислює функціональну стійкість організації.
4. Система повертає три варіанти заміни працівника та значення функціональної стійкості.
5. На основі одержаних варіантів ОПР (менеджер з персоналу у нашому випадку) обирає один з варіантів та приймає рішення.

Описаний процес можна зобразити на діаграмі (рис. 4):

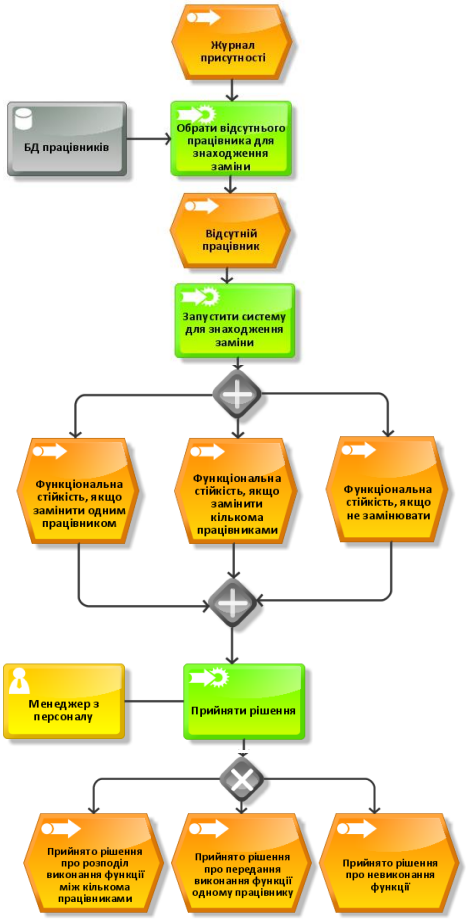

Рис. 4 – Процес прийняття рішення про заміну працівника

На основі теорії викладеної в попередньому розділі було сформовано алгоритм визначення функціональної стійкості організаційної системи для трьох випадків, який лежить в основі системи підтримки прийняття рішень забезпечення функціональної стійкості організаційної системи.

Кожен бізнес-процес розбитий на дрібніші функції. Кожен працівник має перелік своїх функції, які він має щоденно виконувати та їхню оцінку у відсотках, попередньо задану експертами. Попередньо задані коефіцієнти за якими змінюється оцінка виконання функцій працівником: k_1=0.05 – коефіцієнт, що змінює значення оцінки власної та додаткової у працівника, k_2=0.07 – коефіцієнт, що змінює значення оцінки власної та додаткової у працівника, якщо відсутній працівник та працівник, що замінює функцію працюють на різних посадах, k_3=0.1– коефіцієнт, що змінює значення оцінки

власної та додаткової у працівника, якщо відсутній працівник та працівник, що замінює функцію працюють у різних департаментах. Отже, маємо:

1. Зчитуємо дані працівника, якому шукаємо заміну (функції, дати відсутності).
2. Знаходимо кандидатів, які мають ті ж функції, що і відсутній працівник, але які уже не виконують додаткові функції.
3. Далі розглядаємо кілька варіантів:
   1) Не замінювати працівника:
      a) Оцінку виконання всіх функцій прирівнюємо до 0.
      b) Обчислюємо функціональну стійкість системи:

         будуємо матрицю частот різних рівнів якості виконання функцій $V = (v_{ij}), i = 1,\ldots,100, j < n.$ Кожен рядок цієї матриці відображує оцінений рівень якості виконання функції від 0% до 100%, а стовпчик – кількість функцій з зазначеним рівнем якості виконання, далі обчислюємо середнє арифметичне зважене.

   2) Всі можливі функції передати одному працівнику:
      a) Обираємо кандидата зі списку.
      b) Кандидату передаємо усі функції, що співпадають у відсутнього працівника та кандидата.
      c) Функції, які не співпадають визначаємо такими, що не виконуються, отже їх оцінка виконання прирівнюється до 0.
      d) Функцій, що співпадають змінюємо відповідно наступним чином: оцінку виконання функцій множимо на коефіцієнт k_1 (k_1=0.05), отримане значення віднімаємо від оцінки та отримуємо нову оцінку виконання. Якщо кандидат працює не на тій же посаді, що і відсутній працівник змінюємо оцінку: оцінку виконання функцій множимо на коефіцієнт k_2 (k_2=0.07), отримане значення віднімаємо від оцінки та отримуємо нову оцінку виконання. Якщо кандидат працює не в тому ж департаменті, що і відсутній працівник змінюємо оцінку: оцінку

виконання функцій множимо на коефіцієнт k_3 (k_3=0.1), отримане значення віднімаємо від оцінки та отримуємо нову оцінку виконання.

e) Обчислюємо функціональну стійкість організаційної системи в цілому.

f) Значення зберігаємо в масиві.

g) Переходимо до наступного кандидата, якщо він є повертаємось до пункту a), інакше до пункту g).

h) Обираємо максимальне значення функціональної стійкості та відповідного кандидата.

3) Всі можливі функції розділити між двома працівниками:

a) За допомогою алгоритму перебору формуємо пари кандидатів так, що функція, яку ми передаємо передавалась повністю лише одному з них, інший, навіть, якщо він може її виконувати не отримує її.

b) Обираємо пару зі сформованого списку. Передаємо їм можливі функції.

c) Функції, які не співпадають визначаємо такими, що не виконуються, отже їх оцінка виконання прирівнюється до 0.

d) Функції, що співпадають змінюємо відповідно наступним чином: оцінку виконання функцій множимо на коефіцієнт k_1 (k_1=0.05), отримане значення віднімаємо від оцінки та отримуємо нову оцінку виконання. Якщо кандидат працює не на тій же посаді, що і відсутній працівник змінюємо оцінку: оцінку виконання функцій множимо на коефіцієнт k_2 (k_2=0.07), отримане значення віднімаємо від оцінки та отримуємо нову оцінку виконання. Якщо кандидат працює не в тому ж департаменті, що і відсутній працівник змінюємо оцінку: оцінку виконання функцій множимо на коефіцієнт k_3 (k_3=0.1),

отримане значення віднімаємо від оцінки та отримуємо нову оцінку виконання.

e) Обчислюємо функціональну стійкість організаційної системи в цілому.

f) Значення зберігаємо в масиві.

g) Переходимо до наступної пари, якщо вона, то повертаємось до пункту b), інакше до пункту h).

h) Обираємо максимальне значення функціональної стійкості та відповідну їй пару кандидатів.

4. Повертаємо три значення функціональної стійкості організації та відповідних кандидатів, далі ОПР приймає рішення.

Цей алгоритм можна зобразити блок-схемою наступним чином:

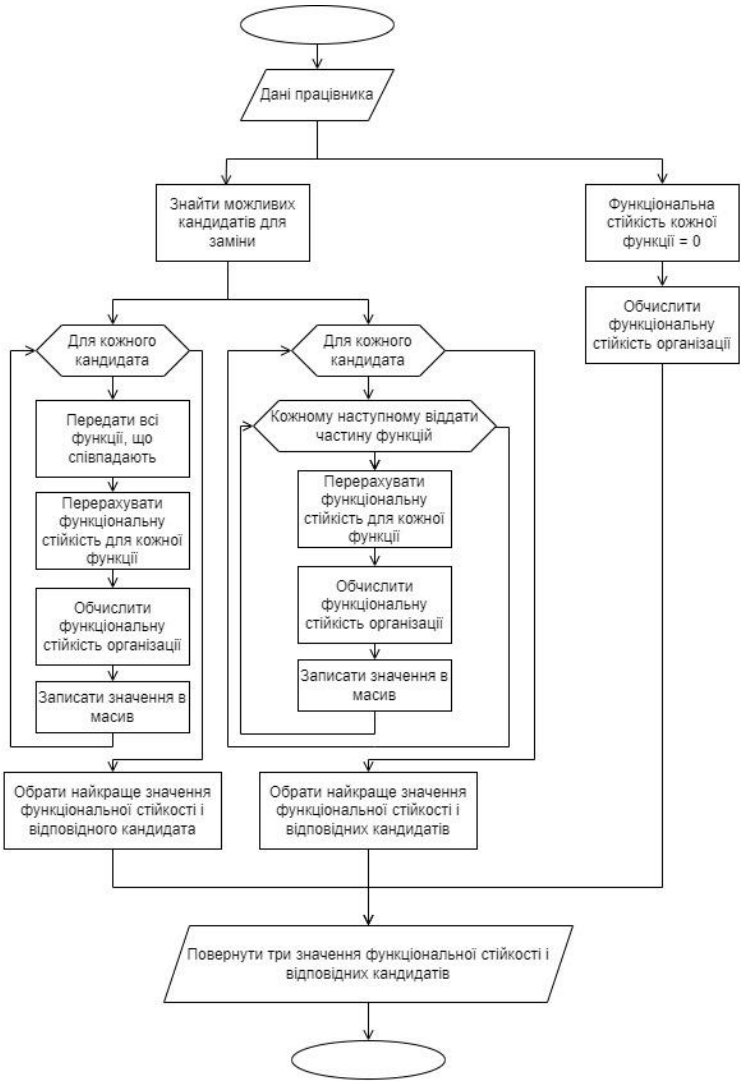

Рис. 5 – Блок-схема алгоритму обчислення функціональної стійкості

2.5. Проектування моделі бази даних

Для застосунку була спроектована реляційна база даних. Подамо її у вигляді концептуальної моделі, де зображені зв’язки між сутностями та їх характеристиками. Отже, маємо наступні сутності:

- Співробітник;
- Посада;
- Департамент;
- Функція;
- Журнал відсутності;
- Обліковий запис.

Кожна сутність має власні атрибути. Так сутність «Співробітник» має атрибути: код працівника – це унікальний номер співробітника, дату народження, ім’я, прізвище, гендер, дата прийому на роботу, пошта, номер телефону та адресу фотографії співробітника. Сутність «Посада» має атрибути код посади та назву посади. Сутність «Департамент» має атрибути код департаменту та назву департаменту. Сутність «Функція» має атрибути код функції та назву функції. Сутність «Журнал відсутності» має атрибути дату початку та дату закінчення відсутності, а також поле в яке заноситься значення 1, якщо заміну працівнику знайшли та значення 0, якщо – не знайшли. Сутність «Обліковий запис» має атрибут пароль від облікового запису менеджера департаменту. У таблиці 2.1 опишемо зв’язки між сутностями.

Таблиця 2.1 Зв’язки між сутностями

Опис зв’язків між сутностями

№	**Сутності, що утворюють зв’язок**	**Тип зв’язку**	**Зміст зв’язку**
1	Співробітник -	1..M :	Один співробітник може займати

	Посада	1..N	одну або кілька посад протягом всієї роботи в організації. Одну посаду може займати один або кілька співробітників.
2	Співробітник - Департамент	M : 1..N	Один співробітник може працювати в одному або різних департаментах протягом всієї роботи в організації. В одному департаменті може працювати кілька співробітників.
3	Співробітник - Департамент	1..M : 0..N	Один співробітник може займати посаду менеджера в кількох департаментах протягом всієї роботи в організації, а може не займати зовсім. В одному департаменті може змінюватись багато менеджерів.
4	Співробітник - Функція	1..M : 1..N	Один співробітник може виконувати одну або багато функцій. Одну й ту саму функцію може виконувати один або кілька співробітників.
5	Співробітник - Функція	1..M : 0..N	Один співробітник може виконувати поточного дня багато функцій, або не виконувати жодної (через відсутність). Одну й ту саму ж функцію поточного дня може виконувати один або кілька співробітників.
6	Співробітник - Журнал	0..M : 0..N	Одного співробітника можна записати до журналу відсутності

	відсутності		багато разів протягом його роботи в організації, або він може жодного разу не бути записаним в цей журнал. Може бути відсутніми кілька співробітників, або всі постійно присутні.
7	Співробітник – Обліковий запис	1 : 0..1	Один співробітник може мати тільки один обліковий запис, або його взагалі не мати (якщо він не керівник департаменту). Один обліковий запис належить тільки одному співробітнику.

Виходячи із вище описаного побудуємо схема «Сутність-зв’язок», концептуальну модель бази даних (рис. 6).

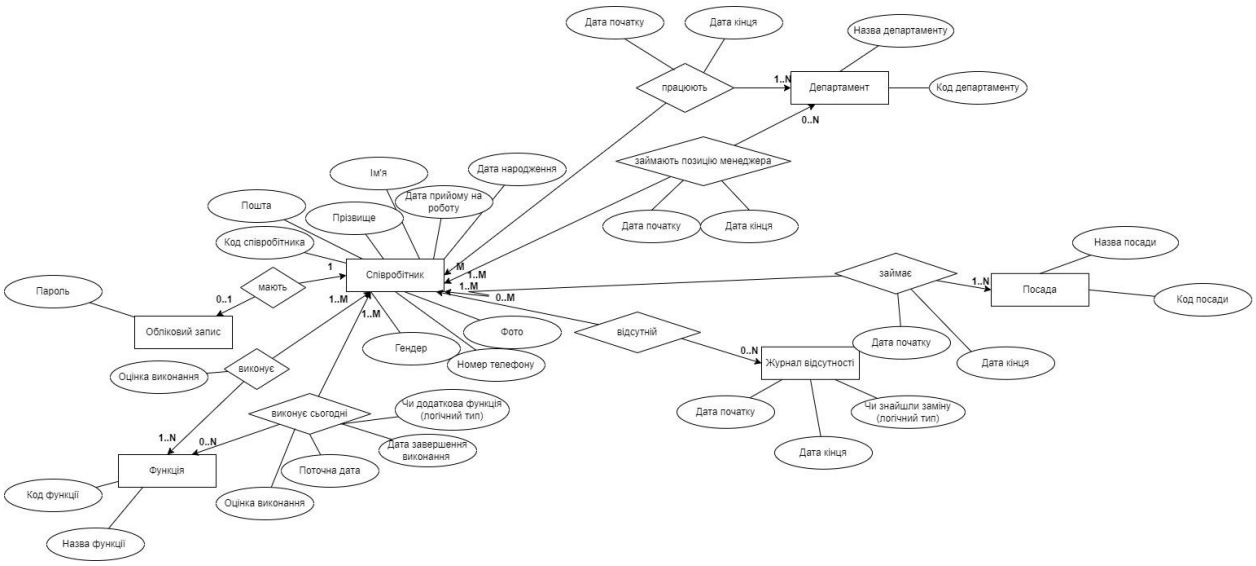

Рис. 6 – Концептуальна модель бази даних

Далі представимо базу даних у вигляді логічної моделі даних (рис. 7). Логічна схема містить детальнішу інформацію про розроблену базу даних, а саме вона містить ключі, первинні та зовнішні. Також, оскільки зв’язки концептуальної моделі мають атрибути, то отримаємо нові додаткові таблиці. А саме таблицю з функціями, які будуть виконуватись працівниками поточного дня, таблиця з займаною працівником посадою, таблиця з депараментом в якому працює працівник та таблицю, що містить дані про те, хто з працівників організації є менеджером (керівником) кожного з департаментів, а також таблицю з відсутністю.

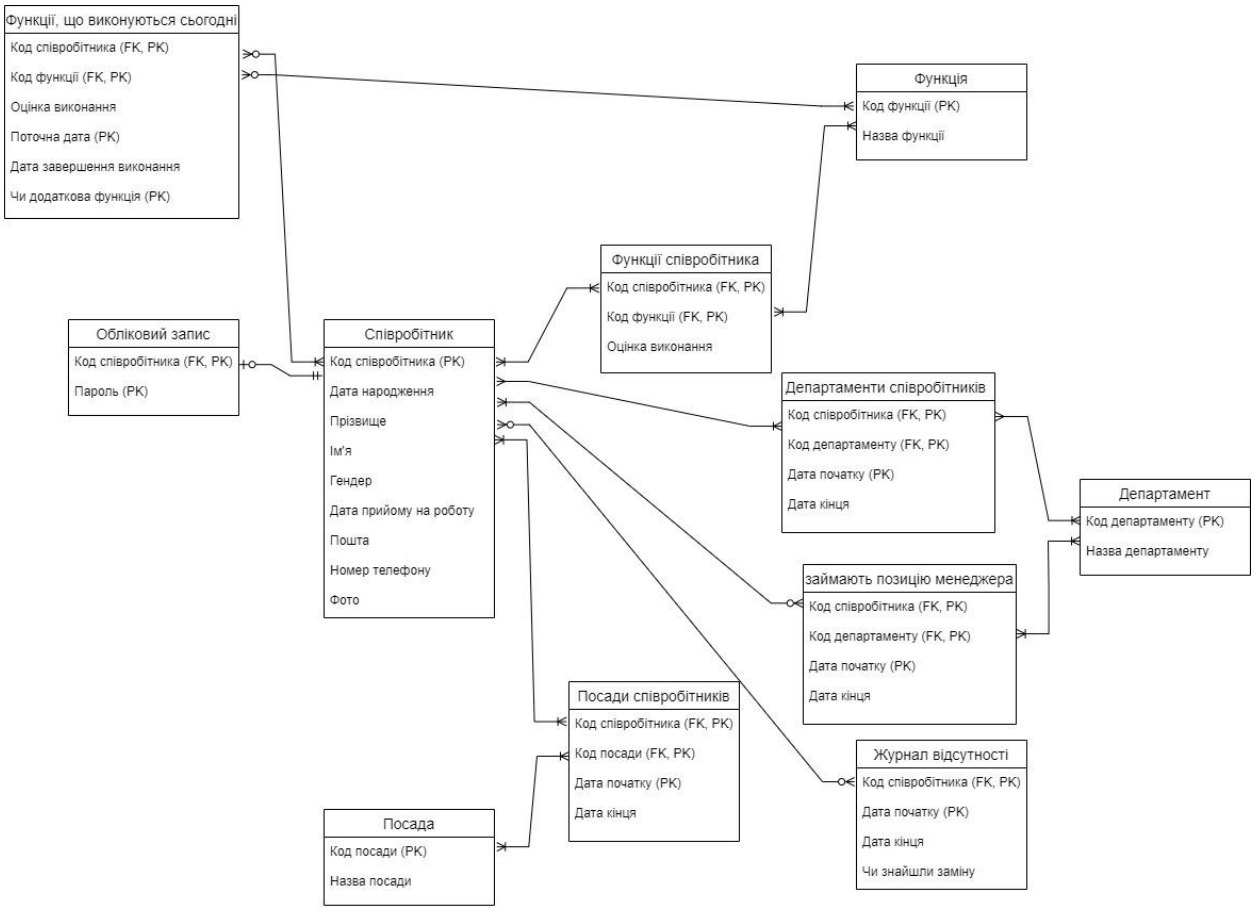

Рис. 7 –Логічна модель бази даних

Тепер відповідно до логічної моделі бази даних представимо фізичну модель бази даних. В цій моделі приймається рішення щодо ідентифікаторів таблиць і атрибутів, а також типів даних та інших

характеристик атрибутів. Ці рішення зведемо в таблицю (табл. 2.2). В таблиці прийняті такі позначення ключів: ПК –первинний ключ; ЗК – зовнішній ключ; СПК – складений первинний ключ.

Таблиця 2.2

Склад та характеристики атрибутів таблиць кінцевого варіанта ДЛМ

Сутність	**Назва атрибуту**	**Ключ**	**Обов'язкове значення**	**Тип даних**	**Обмеження**
Таблиця employees					
employees	emp_no	ПК	Так	Ціле число	Від 1 і більше
employees	birth_date		Так	Дата	YYYY-MM-DD
employees	first_name		Так	Текст	До 14 символів
employees	last_name		Так	Текст	До 16 символів
employees	gender		Так	ENUM	'M', 'F'
employees	hire_date		Так	Дата	YYYY-MM-DD
employees	email			Текст	До 100 символів
employees	number			Текст	До 20 символів
employees	img			Текст	До 200 символів
Таблиця titles_name					
titles_name	title_no	ПК	Так	Ціле число	Від 1 і більше
titles_name	title_name		Так	Текст	До 100 символів
Таблиця departments					
departments	dept_no	ПК	Так	Текст	До 4 символів
departments	dept_name		Так	Текст	До 40 символів
Таблиця functions					
functions	func_no	ПК	Так	Ціле число	Від 1 і більше

functions	func_name		Так	Текст	До 200 символів
Таблиця manager_account					
employees	emp_no	ПК, ЗК	Так	Ціле число	Від 1 і більше
manager_account	password		Так	Текст	До 45 символів
Таблиця titles					
employees	emp_no	СПК, ЗК	Так	Ціле число	Від 1 і більше
titles_name	title_no	СПК, ЗК	Так	Ціле число	Від 1 і більше
titles	from_date	ПК, СПК	Так	Дата	YYYY-MM-DD
titles	to_date		Так	Дата	YYYY-MM-DD
Таблиця presence_log					
employees	emp_no	СПК, ЗК	Так	Ціле число	Від 1 і більше
presence_log	from_date	ПК, СПК	Так	Дата	YYYY-MM-DD
presence_log	to_date		Так	Дата	YYYY-MM-DD
presence_log	replaced		Так	Ціле число	Від 0 до 1
Таблиця dept_emp					
employees	emp_no	СПК, ЗК	Так	Ціле число	Від 1 і більше
departments	dept_no	СПК, ЗК	Так	Текст	До 4 символів
dept_emp	from_date	ПК, СПК		Дата	
dept_emp	to_date		Так	Дата	
Таблиця dept_manager					
employees	emp_no	СПК, ЗК	Так	Ціле число	Від 1 і більше
departments	dept_no	СПК, ЗК	Так	Текст	До 4 символів
dept_manager	from_date	ПК, СПК	Так	Дата	
dept_manager	to_date				
Таблиця emp_functions					
employees	emp_no	СПК, ЗК	Так	Ціле число	Від 1 і більше

functions	func_no	СПК, ЗК	Так	Ціле число	Від 1 і більше
emp_functions	func_assessment		Так	Дійсне число	
Таблиця emp_task_list					
employees	emp_no	СПК, ЗК	Так	Ціле число	Від 1 і більше
functions	func_no	СПК, ЗК	Так	Ціле число	Від 1 і більше
emp_task_list	new_func_assessment		Так	Дійсне число	
emp_task_list	date		Так	Дата	YYYY-MM-DD
emp_task_list	to_date		Так	Дата	YYYY-MM-DD
emp_task_list	add_func		Так	Ціле число	Від 0 і до 1

Отже, маємо наступні таблиці:

- employees – таблиця, яка містить дані про усіх співробітників;

- titles_name – таблиця, що містить перелік посад;

- titles – таблиця, що містить перелік працівників та посади, які вони займають;

- dept_manager - перелік співробітників, які є менеджерами підрозділів;

- dept_emp – перелік працівників департаментів;

- departments – перелік департаментів, підрозділів організації;

- functions – загальний перелік можливих функцій;

- emp_functions – перелік працівників та їх функцій, а також оцінки виконання функцій визначені експертами;

- emp_task-list – таблиця, що містить функції на поточний день, саме на основі даних цієї таблиці обчислюється функціональна стійкість організації;

- presence_log – журнал відсутності працівників;

- manager_account – таблиця, що містить паролі менеджерів для доступу до сторінок з співробітниками департаменту в якому вони є керівниками.

Згідно описаного вище отримуємо фізичну модель бази даних (рис. 8) на якій зображені зв’язки таблиць та типи даних полів, що зберігаються в кожній таблиці.

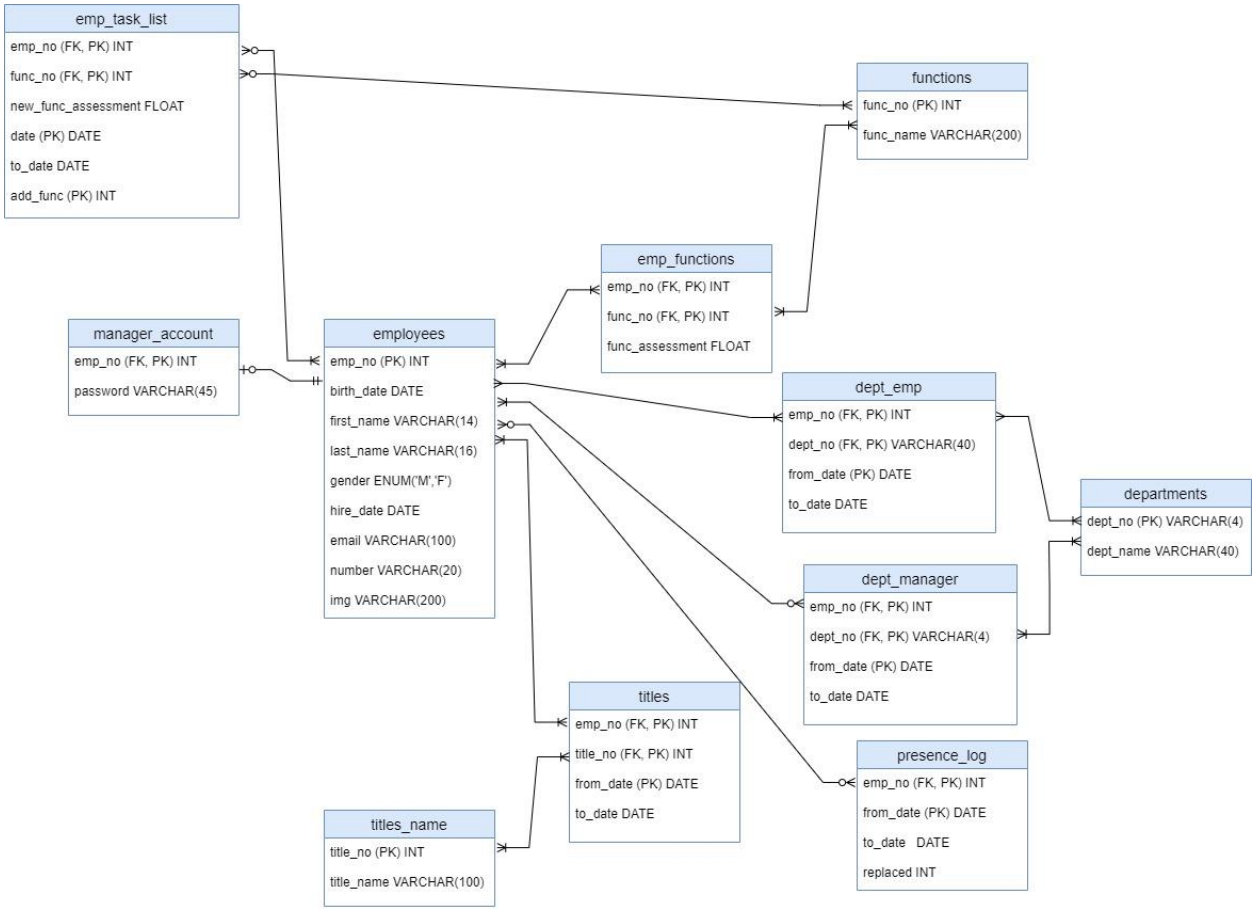

Рис. 8 – Даталогічна модель бази даних

РОЗДІЛ 3 РОЗРОБКА І РЕАЛІЗАЦІЯ СППР ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНАЛЬНОЇ СТІЙКОСТІ ОРГАНІЗАЦІЙНОЇ СИСТЕМИ

3.1. Програмна реалізація СППР забезпечення функціональної стійкості організаційної системи

Додаток реалізовувався в інтегрованому середовищі розробки для мови програмування Python. Відповідно веб-додаток було розроблено мовою програмування Python, а саме її фреймворк Django. Django один з найпопулярніших фреймворків для розробки веб-додатків. Він дуже зручний, адже має свою автоматично генеровану адмін-панель. Також він має власний ORM, за допомогою якого можна звертатись до бази даних без SQL запитів. Також він безпечний, адже захищений від таких розповсюджених атак, як sql-ін’єкції. Архітектура веб-додатку розробленого за допомогою Django схожа на модель MVC "Модель-Представлення-Контролер", але у цьому фреймворці вона має назву MTV “Model- Templates- Views” (рис. 9).

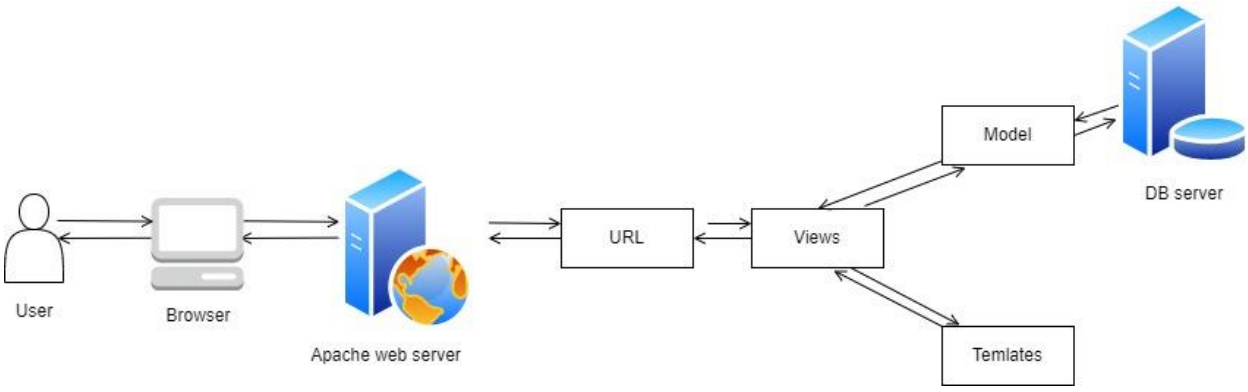

Рис. 9 – Архітектура веб-додатку

За допомогою model ми зв’язуємось з базою даних. Views відповідає за обробку запитів: коли http-запит надходить на сервер, views обробляє його за допомогою функцій, які там реалізовані, а також надсилає відповіді, крім цього через views можна редагувати, створювати та видаляти записи нашої

бази даних. В templates зберігаються html-сторінки. Використовуючи мову шаблонів Django можна виводити дані з бази даних на сторінку.

Загалом додаток має структуру як показано на рисунку 10. Маємо кореневий каталог проєкту diplom, яка містить каталог app з додатком, каталог diplom з допоміжними файлами для роботи додатку, каталог media та файл manage.py завдяки якому і відбувається запуск проєкту. каталог media містить ще один каталог images, який містить всі зображення, що використовуються для дизайну та відображення проєкту, а також в цю папку зберігаються фотографії співробітників. Підкаталог diplom каталогу diplom містить файл налаштувань settings.py, саме там налаштовуються каталоги, обирається база даних. Далі каталог app, що містить каталог static, який містить каталог css, де містяться файли .css з дизайном сторінок застосунку, які підключаються до html сторінки. В нашому випадку маємо два файли main.css з дизайном всіх сторінок, форм та кнопок, та login.css, який містить дизайн стартової сторінки з формою для авторизації в додатку. Далі маємо каталог templates, який містить файли з html розміткою сторінок додатку. Далі маємо папку templatetags, в ній міститься файл з описом python-функцій, які можна використовувати в html файлі. Далі файли: admin.py – який містить класи моделей бази даних для відображення їх у адміністративній частині застосунку; forms.py – файл, що містить класи, які визначають форми для роботи з базою даних, використовуючи їх, зручно створювати форми на сторінці, які зв’язані з певною моделлю (таблицею), а також створювати записи в базу даних та перевіряти введені дані на відповідність визначеним в базі даних обмеженням; models.py – файл, що містить класи, кожен з яких пов’язаний з певною таблицею бази даних, за допомогою цих моделей ми можемо створювати нові записи таблиці, редагувати та видаляти вже існуючі; views.py - це функції Python, які беруть http-запити і повертають відповідь http, як документи HTML, з кожної таклї функції ми можемо викликати інші визначені python-функції; urls.py – файл, що містить url шаблони до яких звертаються функції визначені у файлі views.py.

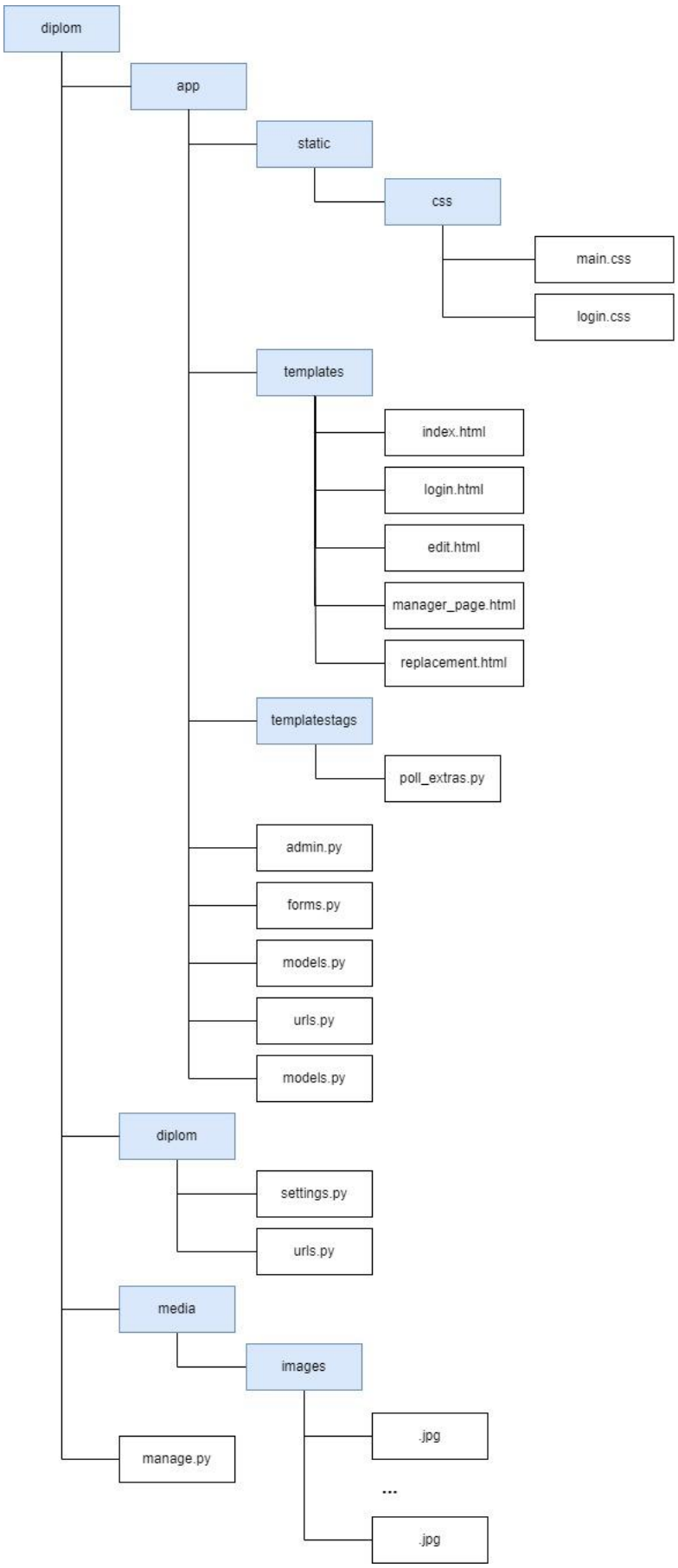

Рис. 10 – Структура розробленого додатку

Інтерфейс веб-додатку був створений за допомогою HTML та CSS, а також використовувалась мова програмування JavaScript, а саме її бібліотека jQuery. За допомогою цієї бібліотека дані зі сторінки можна переносити у форму. Для керування базою даних була використана система MySQL.

Розроблений веб-додаток має такі сторінки:

- login.html – сторінка авторизації користувача (рис. 14);

- manager_page.html – сторінка керівника департаменту на яку він потрапляє після авторизації (рис. 15);

- index.html – головна сторінка на яку потрапляє менеджер з персоналу після авторизації (рис. 16);

- edit.html – сторінка для редагування інформації про співробітників (рис. 14);

- replacement.html - сторінка для знаходження заміни відсутньому працівнику (рис. 21).

Загалом структуру інтерфейсу розробленого веб-додатку можна подати у вигляді діаграми наступним чином (рис. 11). З діаграми видно, що відкриваючи додаток, ми потрапляємо на сторінку login.html, тобто сторінку авторизації. Після цього залежно від того, хто авторизувався, менеджер з персоналу чи керівник департаменту, ми потрапляємо на сторінку index.html, тобто головну сторінку, якщо авторизувався менеджер з персоналу, та на сторінку manager_page.html, якщо авторизувався керівник. Функціонал цих сторінок відрізняється, оскільки менеджер з персоналу має більше можливостей і повноважень, і саме він відповідальний за прийняття рішення стосовно визначення працівника, який буде замінити відсутнього. Далі з описаних вище двох сторінок, натискаючи на посилання будь-якого працівника, у відкритому модальному вікні можна побачити кнопку «Редагувати», натискаючи на яку, ми переходимо на сторінку edit.html. На цій сторінці можна редагувати всі дані працівника, а також функції працівника, тобто додавати нові разом з їх оцінками. У менеджера з персоналу є додаткові можливості, натискаючи кнопку «Знайти заміну» в

тому ж модальному вікні, він переходить на сторінку replacement.html. Натискання цієї кнопки запускає на виконання алгоритм знаходження заміни відсутньому працівнику, тому перехід на сторінку replacement.html займає певний час. Після завершення обрахунку відкривається сторінка та виводяться результати алгоритму.

Менеджер обирає потрібний варіант після прийняття рішення та натискає кнопку «Замінити», що повертає його на головну сторінку, а обрані працівники отримують нові функції, що записується в базу даних.

Рис. 11 – Структура веб-додатку

3.2. Використання розробленого веб-додатку

Використання розробленого застосунку можна представити у вигляді User Flow діаграми (рис. 13). На діаграмі позначено:

Рис. 12 – Легенда діаграми

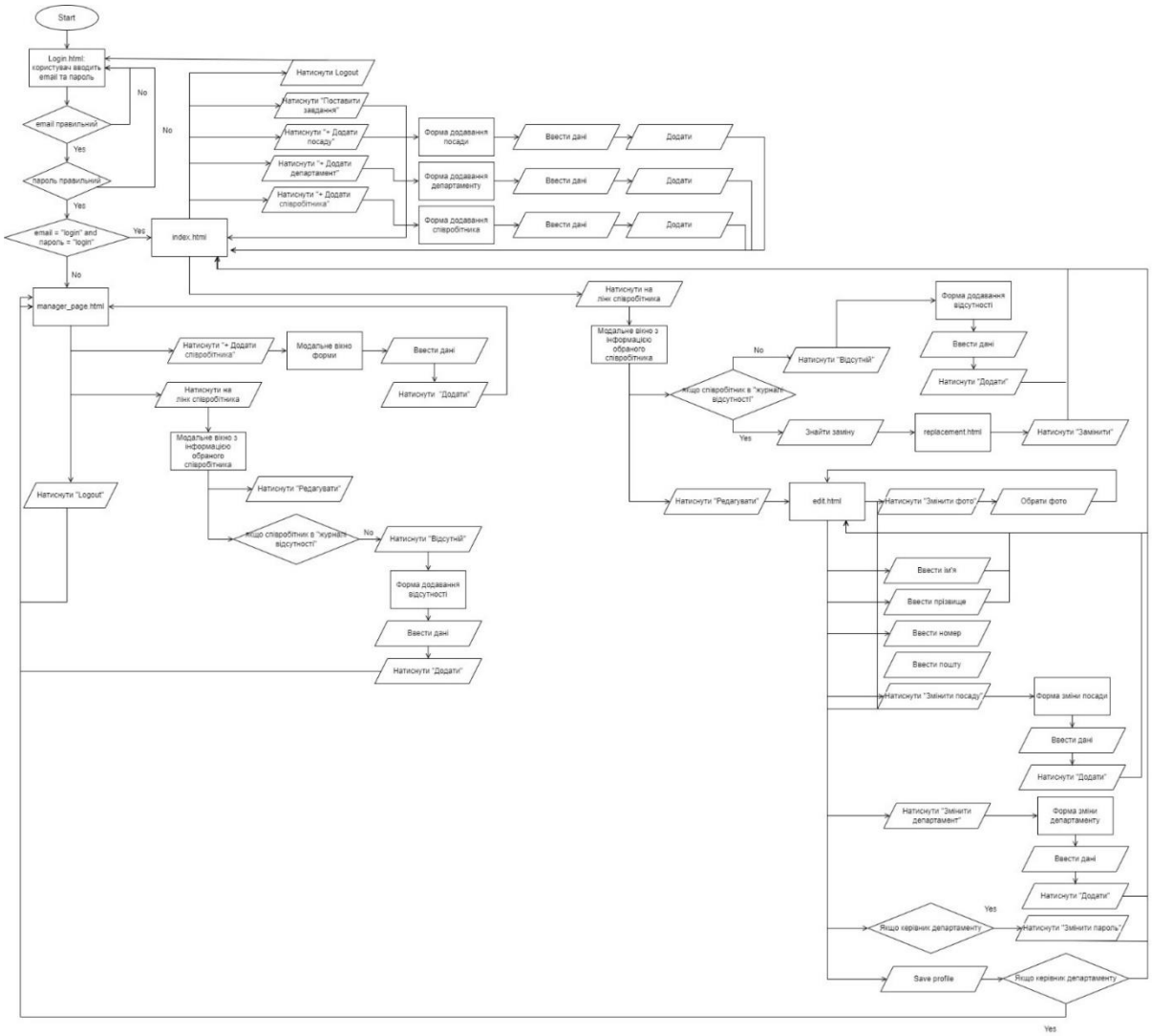

Рис. 13 – User Flow діаграма

Відкриваючи додаток, користувач потрапляє на сторінку з формою для введення логіну та паролю (рис. 14). Авторизація відбувається за email’ом. Якщо пошта введена невірно, то ми отримуємо повідомлення про те, що пошта введена невірно. Якщо пароль введено невірно, то ми отримуємо повідомлення про те, що пароль введено не вірно.

Паролі попередньо введені менеджером з персоналу. Він може встановить пароль або змінить його на сторінці редагування кожного окремого працівника. Залогінитися можуть лише менеджери (керівники) департаментів та менеджер з персоналу, головний користувач додатку. Для менеджерів департаментів та менеджера з персоналу відкриваються різні сторінки. Керівники департаментів мають обмежений функціонал, тоді як менеджеру з персоналу доступні всі функції додатку.

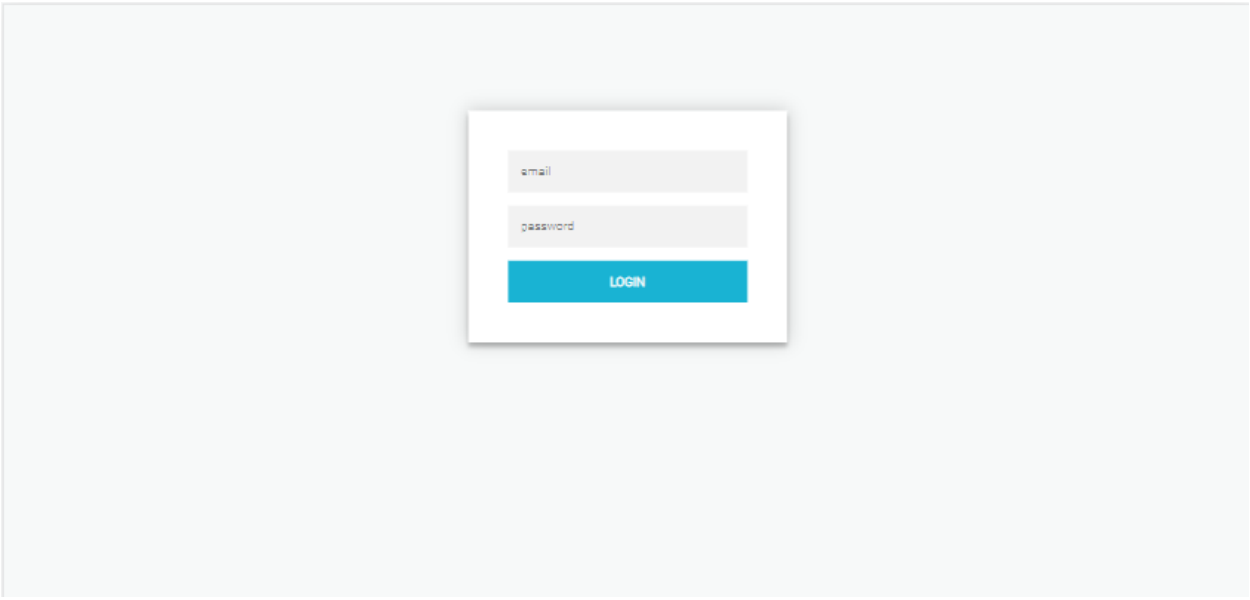

Рис. 14 – Сторінка авторизації

Після успішної авторизації, користувач, якщо це менеджер з персоналу, потрапляє на головну сторінку (рис. 16), якщо це керівник департаменту – на сторінку свого департаменту (рис. 15). На бічній панелі він бачить своє ім’я та фото. На сторінці виведені всі співробітники компанії з вказанням ім’я, прізвища, номером телефона, поштою, посадою та департаментом до якого вони належать (якщо користувач – керівник департаменту, то він бачить лише працівників свого департаменту).

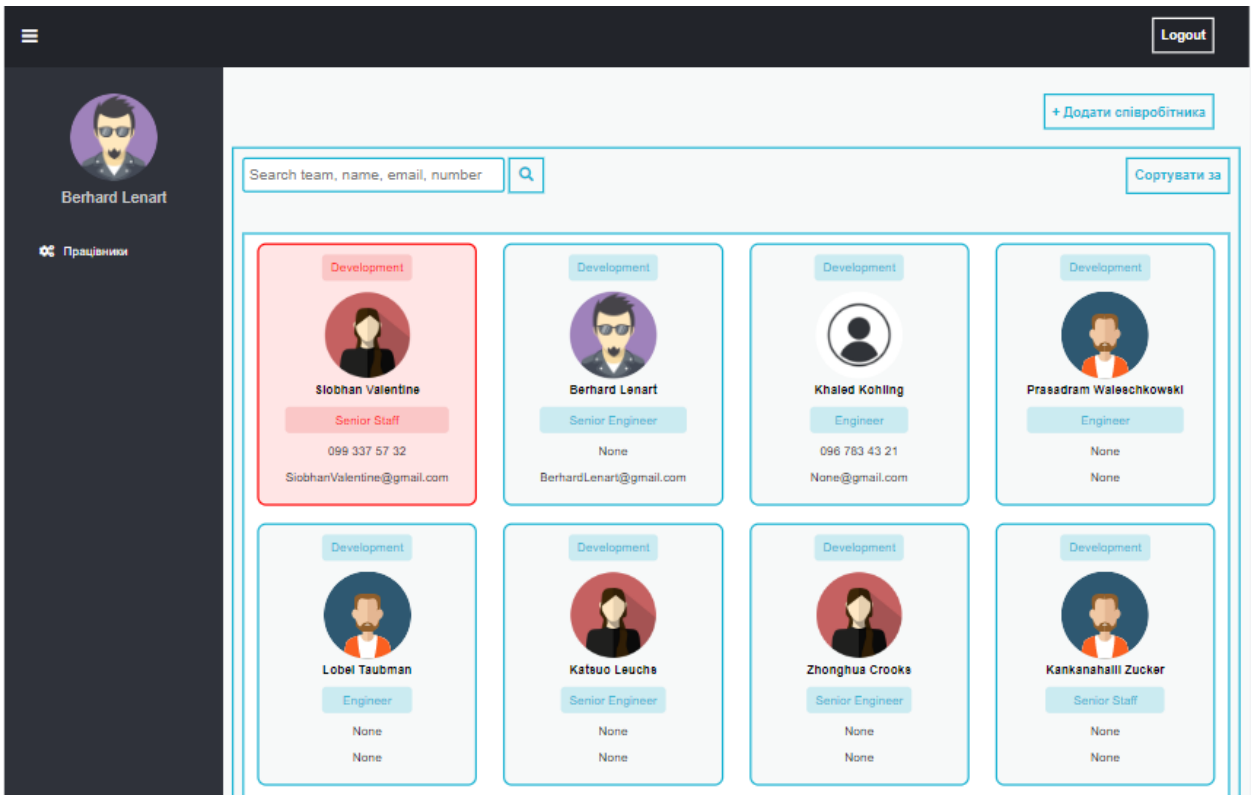

Рис. 15 – Головна сторінка керівника департаменту

Головна сторінка менеджера з персоналу має наступний вигляд:

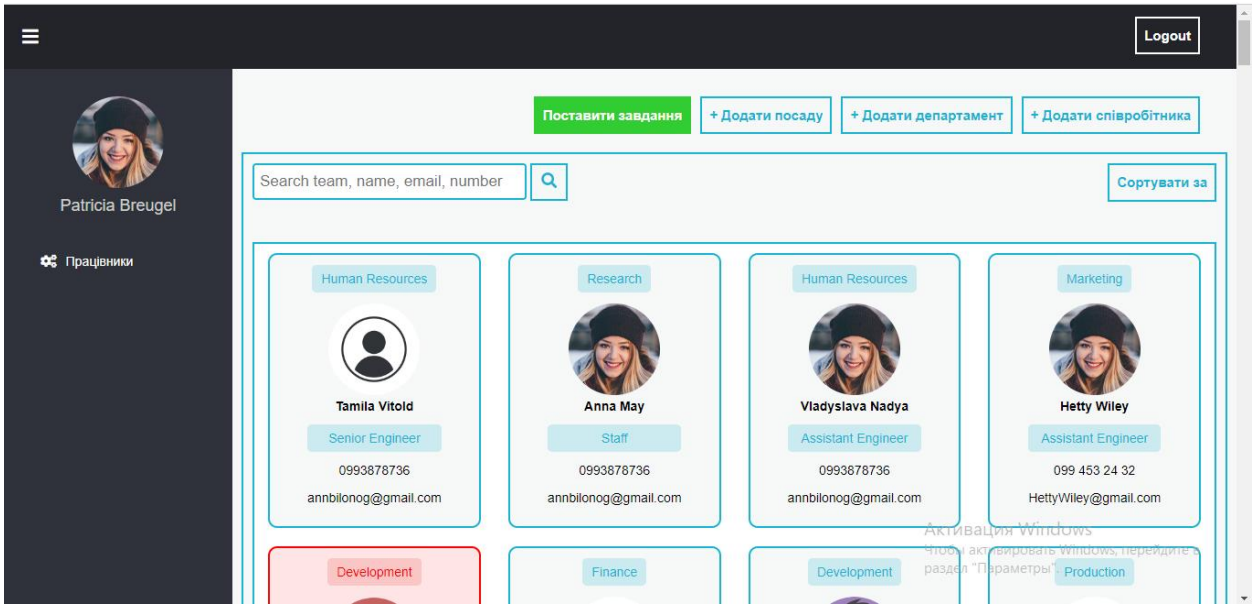

Рис. 16 – Головна сторінка

На головній сторінці є наступні кнопки:

- «Поставити завдання» - передбачається, що кожного дня менеджер з персоналу, заходячи на сторінку, одразу натискатиме її для того, щоб в базу даних вносилась інформація про поточні функції виконувані всіма працівниками. В подальшому на основі цих записів буде рахуватись функціональна стійкість організації.

- «+ Додати посаду» - кнопка при натисканні якої відкривається форма для внесення даних про нову можливу посаду (рис. 17);

- «+ Додати департамент» - кнопка при натисканні якої відкривається форма для внесення даних про новий можливий департамент (рис. 18);

- «+ Додати співробітника» - кнопка при натисканні якої відкривається форма для внесення даних про нового співробітника (рис. 19).

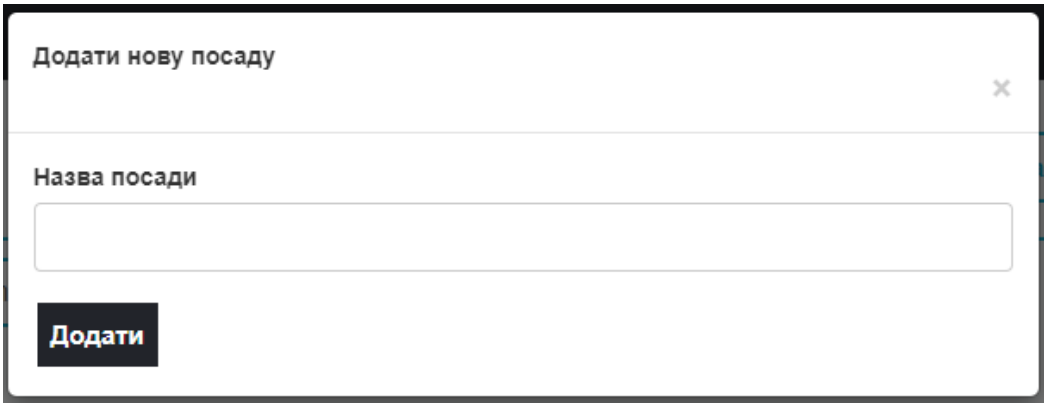

Рис. 17 – Форма додавання нової посади

Рис. 18 – Форма додавання нового департаменту

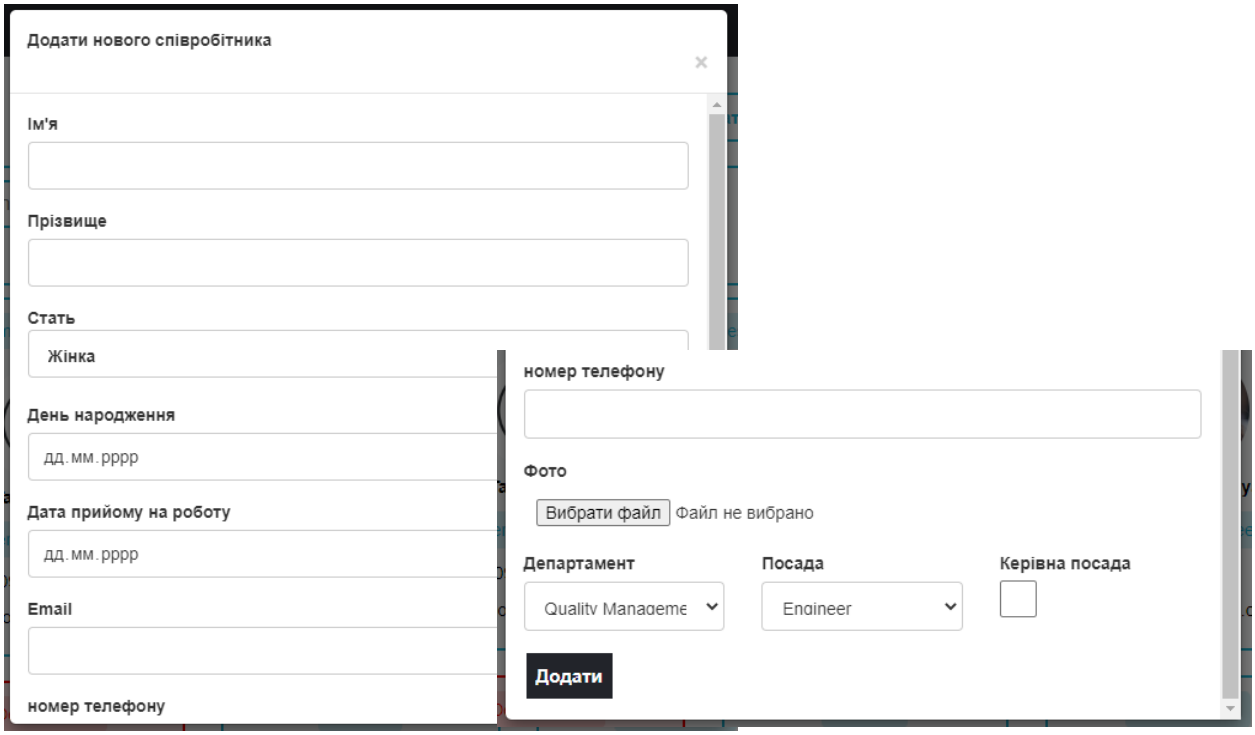

Рис. 19 – Форма додавання нового співробітника

На головній сторінці червоним кольором позначені співробітники, які можуть бути відсутніми поточного дня (до того як їм не знайдуть заміну, вони мають виконувати свої функції), а отже, яким потрібно знайти заміну (рис. 20).

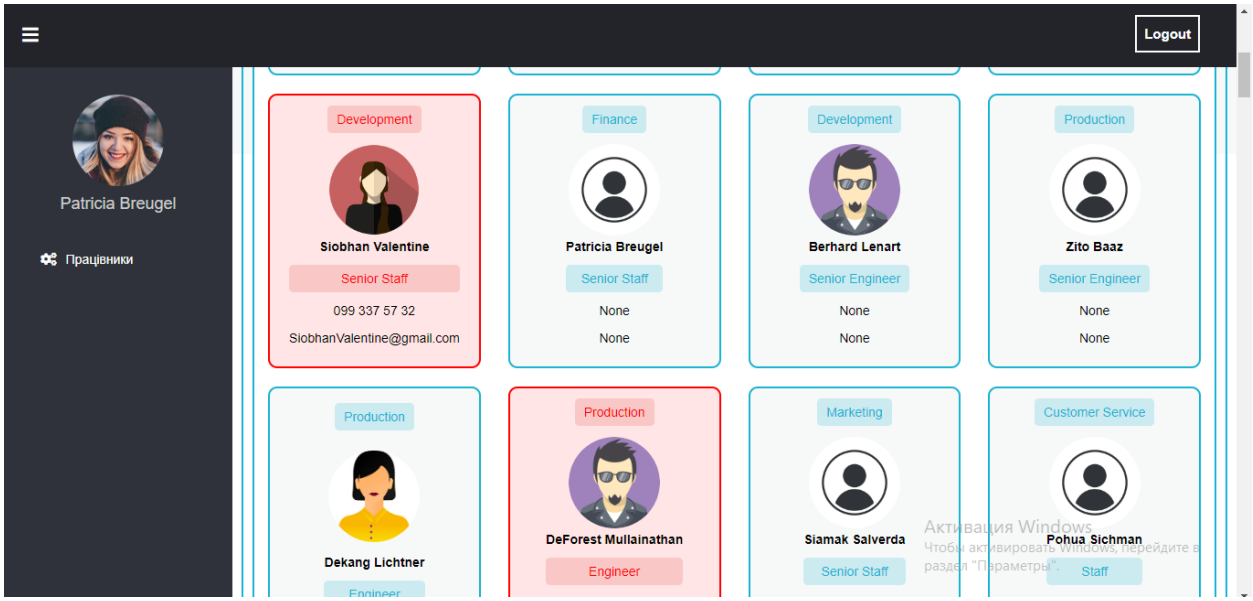

Рис. 20 – Червоним позначені працівники, яким потрібно знайти заміну

Сірим кольором позначені працівники, яким уже знайшли заміну.

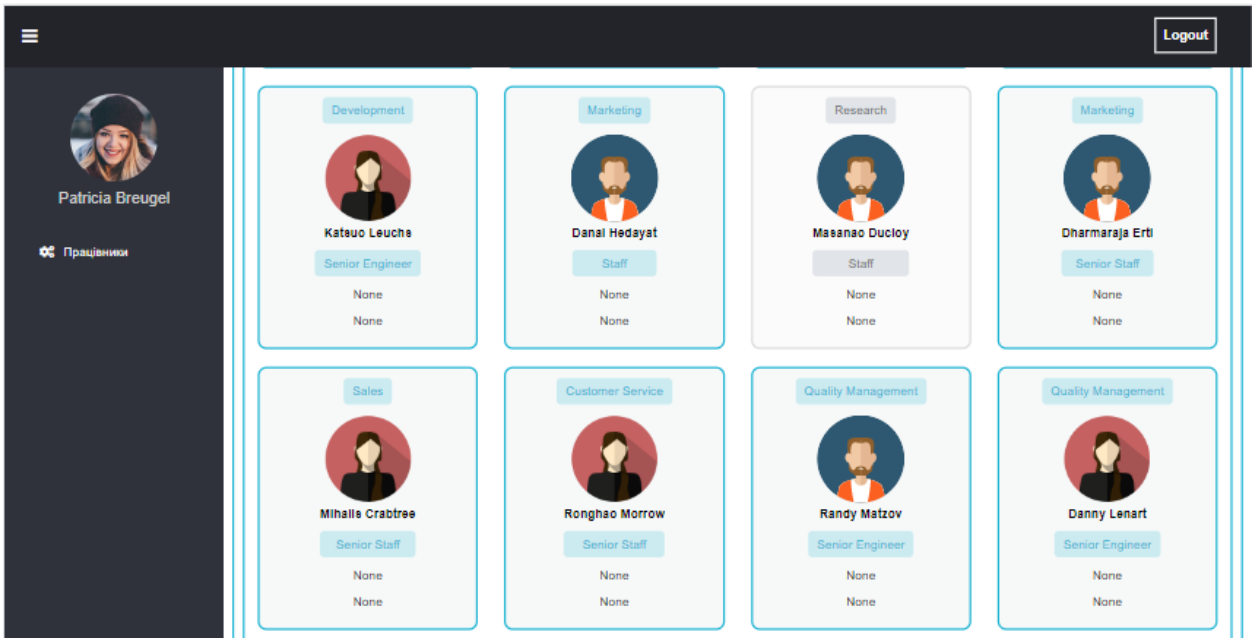

Рис. 21 – Сірим позначені працівники, яким знайшли заміну

При натисканні на співробітника відкривається модальне вікно з детальнішою інформацією та кнопками (рис. 22):

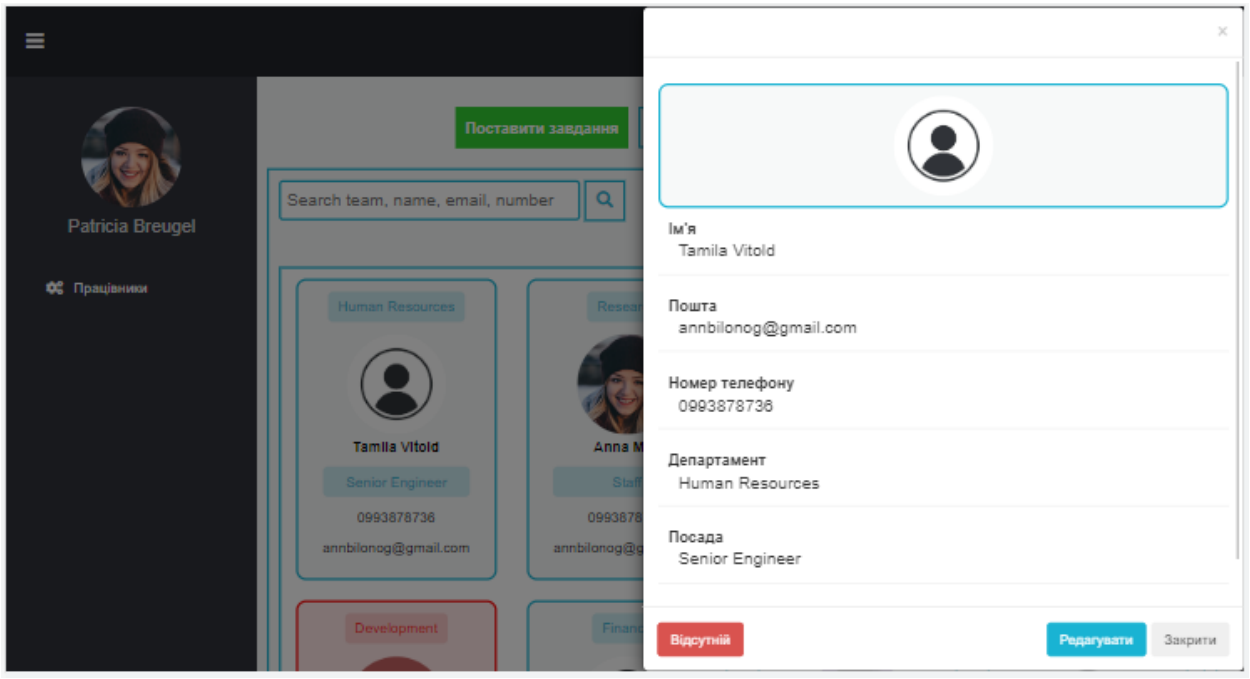

Рис. 22 – Модальне вікно з інформацією про працівника

Воно містить кнопку «Редагувати». Ця кнопка слугує для того, щоб відкривати нове вікно для редагування інформації про співробітника. При натисканні на цю кнопку відривається нова сторінка (рис. 23) – сторінка з детальною інформацією, історією роботи на посадах та департаментах, а також функції, які також можна редагувати:

Рис. 23 – Сторінка редагування

Ця сторінка містить кнопки « Додати посаду», «Додати керівництво» та «Додати департамент». Принцип роботи їх однаковий, тому розглянемо одну з них, наприклад, «Додати посаду». При натисканні відкривається нове модальне вікно з формою (рис. 24). Воно містить два поля «Кінець», оскільки передбачається, що при зміні посади робота на попередній посаді зупиняється. Заповнимо цю форму та натиснемо кнопку «Додати» (рис. 25). Друге поле «Кінець» можна не заповнювати, перше – обов’язкове. Бачимо, що відбулися зміни (рис. 26). Аналогічно працюють інші кнопки.

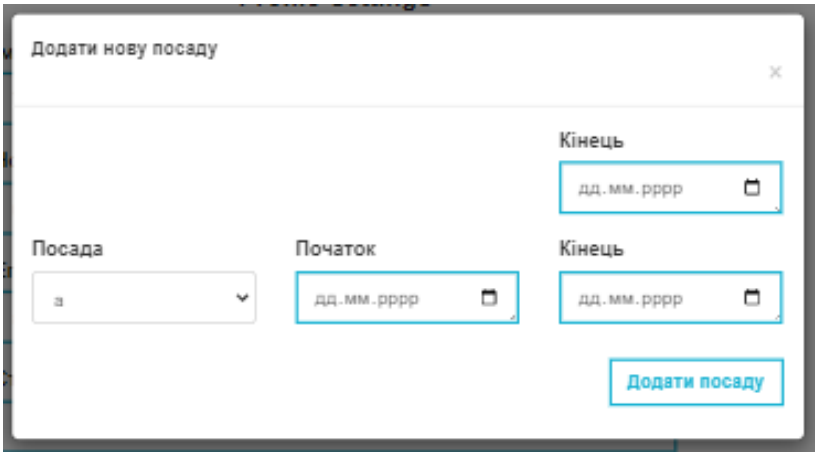

Рис. 24 – Форма для додавання нової посади

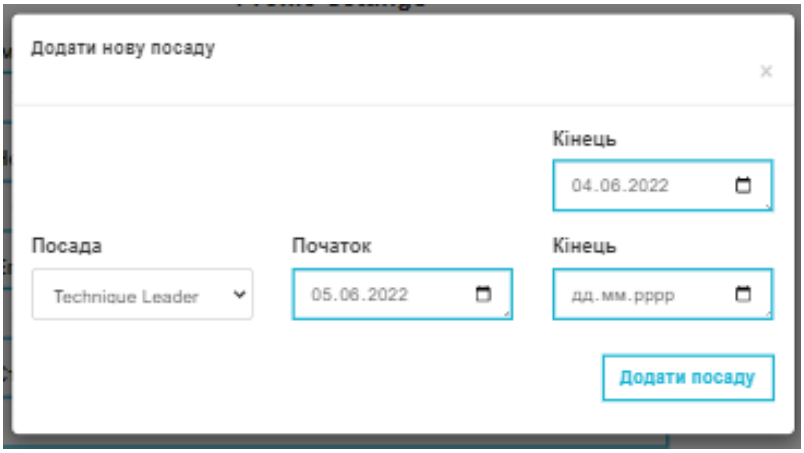

Рис. 25 – Заповнена форма

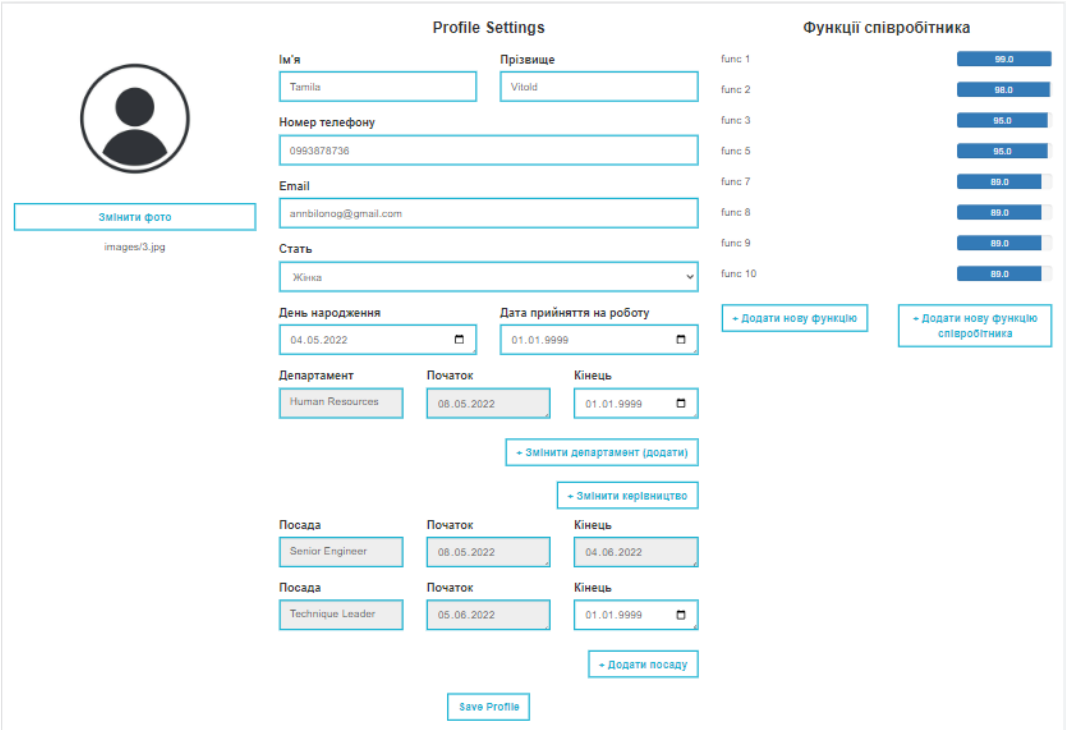

Рис. 26 – Оновлення сторінки

На модальному вікні при натисканні на співробітника також є кнопка «Відсутній» (рис. 22), вона є тільки у співробітників, які присутні поточного дня. Натискаючи на неї, відкривається нове модальне вікно (рис. 27). Тобто менеджер з персоналу, так само як і менеджери департаментів, може вносити інформацію про відсутність. При введенні поточної дати, працівник стане «червоним», тобто відсутнім (рис. 28).

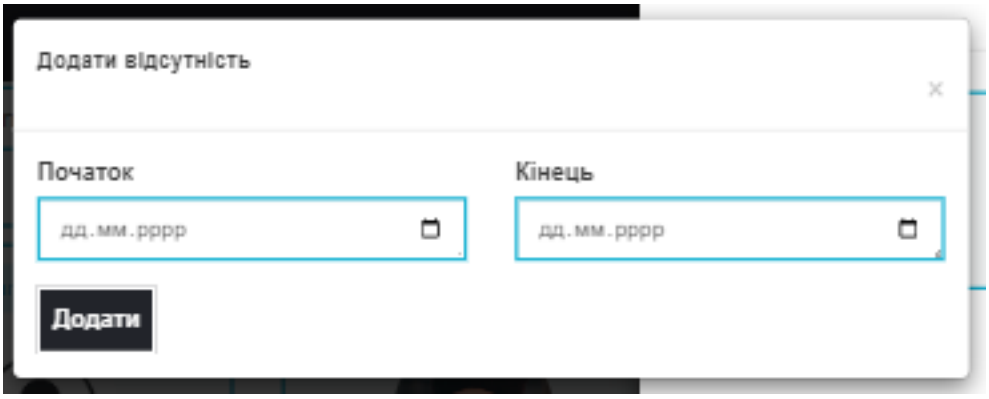

Рис. 27 – Форма відсутності

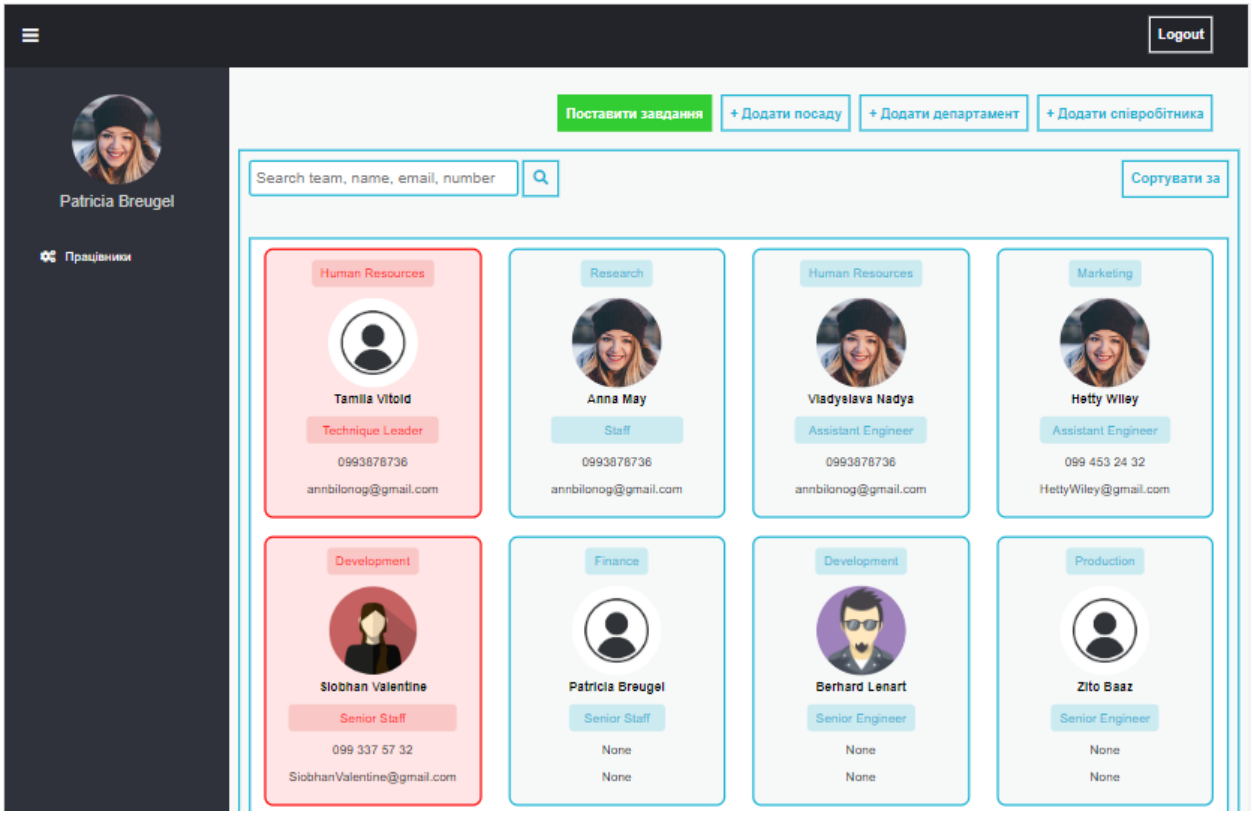

Рис. 28 – Робота кнопки «Відсутній»

У працівників, які будуть відсутніми, замість кнопки «Відсутній» є кнопка «Знайти заміну» (рис. 29).

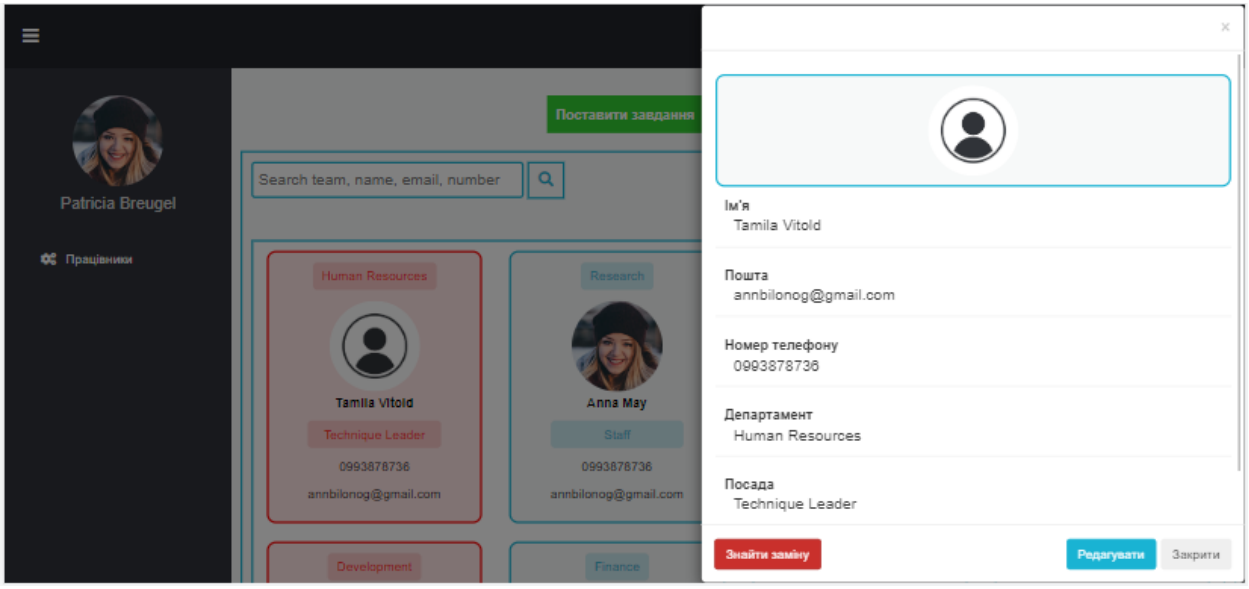

Рис. 29 – Модальне вікно відсутнього працівника

При натисканні на кнопку «Знайти заміну» відкривається нове вікно для знаходження заміни працівнику (рис. 30). Натискання кнопки «Знайти заміну» запускає на виконання розроблений алгоритм підбору працівників для заміни, передаючи функції яким ми б отримали найменше економічних втрат. Отже, в результаті роботи алгоритму ми отримуємо три варіанти з яких можна обрати: не замінювати, замінити на одного, замінити на двох. Оскільки в даному випадку найкращий показник функціональної стійкості мати система, якщо функції відсутнього працівника передати двом працівникам, то варто обрати саме цей варіант. Після прийняття рішення обираємо потрібний варіант, натискаємо кнопку «Замінити», запис з новими функціями зберігається в базу даних в таблицю з поточними функціями працівників.

Рис. 30 – Варіанти заміни працівника

ВИСНОВКИ

У результаті виконання дипломної робот було розроблено систему підтримки прийняття рішень забезпечення функціональної стійкості організаційної системи.

Було проведено аналіз теми дослідження, а саме функціональної стійкості. Аналіз публікацій та наявних застосунків показав, що додатків для забезпечення функціональної стійкості саме організаційної системи поки не існує.

Також було спроектовано модель використання процесів, модель роботи СППР у вигляді «чорного ящика» у нотації IDEF0. Була побудована діаграма процесу прийняття рішення про заміну працівника, а також розроблений алгоритм заміни працівника з урахуванням функціональної стійкості організаційної системи.

Було спроектовано базу даних працівників, яка містить таблиці з посадами, департаментами, функціями організації в цілому та кожного працівника окремо, а також таблицю завдань поточного дня та журнал присутності. Також база даних містила таблицю, яка мала в собі перелік функцій працівника та їх оцінки виконання, визначені експертами наперед.

Було визначено інструменти для розробки додатку, його архітектура. Створено інтерфейс користувача з різними формами для роботи з базою даних та сторінками для редагування даних та знаходженням заміни працівнику. Було реалізовано алгоритм знаходження заміни відсутньому працівнику з урахуванням функціональної стійкості.

Розроблену систему можна використовувати на підприємствах, в організаціях, де є велика кількість працівників і прийняття рішень про заміну складне і дуже важливо. Оскільки від безперервного виконання функцій (бізнес-процесів) залежить конкурентоспроможність та безпека підприємства, а особливо економічна та інформаційна.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Раев В. К. Организационные системы / В. К. Раев. // Информационніе технологии в науке, образовании и управлении. – 2019. – С. 94–100.

2. Організаційна структура управління. - [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/Організаційна_структура_управління

2. Бусленко Н.П., Коваленко И.Н. Лекции по теории сложных систем. – К.: Советское радио, 1973. - 439 с.

3. Принципы и структура системного анализа. - [Електронний ресурс]. - Режим доступу:

4. Шуміло О. С. Особливості наукових підходів до визначення поняття «економічна безпека підприємства» // Проблеми економіки. – 2014. - Вип. 4. - С. 339-344

5. Кравченко Ю. В. Сучасний стан та шляхи розвитку теорії функціональної стійкості / Ю. В. Кравченко, С. А. Микусь // Збірник наукових праць Інституту проблем моделювання в енергетиці ім. Г. Є. Пухова. - 2013. - Вип. 69. - С. 40-48

6. Г. М. Гнатієнко, В. Є. Снитюк. Експертні технології прийняття рішень. – К.: Маклаут, 2008. - 444 с.

7. Емерджентність. - [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/ Емерджентність

8. Гапак Н.М. ЕКОНОМІЧНА БЕЗПЕКА ПІДПРИЄМСТВА: СУТНІСТЬ, ЗМІСТ ТА ОСНОВИ.// 2013.

ДОДАТОК А

ЛІСТИНГ ФАЙЛІВ ПРОГРАМНОГО КОДУ

models.py

```
from django.db import models

# Create your models here.
import datetime

from django.db import models
from django.utils import timezone

class Employees(models.Model):
    emp_no = models.IntegerField(primary_key=True)
    birth_date = models.DateField()
    first_name = models.CharField(max_length=14)
    last_name = models.CharField(max_length=16)
    gender = models.CharField(max_length=1)
    hire_date = models.DateField()
    email = models.CharField(max_length=50, blank=True)
    number = models.CharField(max_length=20, blank=True)
    img = models.ImageField(upload_to='images/', default="images/3.jpg")

    class Meta:
        managed = False
        db_table = 'employees'

    def __str__(self):
        return self.first_name
class TitlesName(models.Model):
    title_no = models.IntegerField(primary_key=True)
    title_name = models.CharField(unique=True, max_length=40)

    class Meta:
        managed = True
        db_table = 'titles_name'

    def __str__(self):
```

```
            return self.title_name

        class Titles(models.Model):
            emp_no = models.OneToOneField(Employees, models.DO_NOTHING, db_column='emp_no',
primary_key=True, unique=False)
            title_no = models.ForeignKey(TitlesName, models.DO_NOTHING, db_column='title_no',
unique=False)
            from_date = models.DateField(unique=False)
            to_date = models.DateField(default='9999-01-01')
            class Meta:
                managed = True
                db_table = 'titles'
                unique_together = (('emp_no', 'title_no', 'from_date'),)
        class Departments(models.Model):
            dept_no = models.CharField(primary_key=True, max_length=4)
            dept_name = models.CharField(unique=True, max_length=40)
            class Meta:
                managed = True
                db_table = 'departments'

            def __str__(self):
                return self.dept_name

        class DeptEmp(models.Model):
            emp_no = models.OneToOneField('Employees', models.DO_NOTHING, db_column='emp_no',
primary_key=True, unique=False)
            dept_no = models.ForeignKey(Departments, models.DO_NOTHING, db_column='dept_no')
            from_date = models.DateField()
            to_date = models.DateField(default='9999-01-01')

            class Meta:
                managed = False
                db_table = 'dept_emp'
                unique_together = (('emp_no', 'dept_no'),)


        class DeptManager(models.Model):
            emp_no = models.OneToOneField('Employees', models.DO_NOTHING, db_column='emp_no',
primary_key=True, unique=False)
            dept_no = models.ForeignKey(Departments, models.DO_NOTHING, db_column='dept_no')
            from_date = models.DateField()
            to_date = models.DateField(blank=True, default='9999-01-01')
```

```
        class Meta:
            managed = False
            db_table = 'dept_manager'
            unique_together = (('emp_no', 'dept_no'),)


    class Functions(models.Model):
        func_no = models.IntegerField(primary_key=True)
        func_name = models.CharField(unique=True, max_length=200)

        class Meta:
            managed = True
            db_table = 'functions'

        def __str__(self):
            return self.func_name

    class EmployeeFunctions(models.Model):
        emp_no = models.OneToOneField('Employees', models.DO_NOTHING, db_column='emp_no',
primary_key=True, unique=False)
        func_no = models.ForeignKey(Functions, models.DO_NOTHING, db_column='func_no')
        func_assessment = models.FloatField()

        class Meta:
            managed = True
            db_table = 'emp_functions'
            unique_together = (('emp_no', 'func_no'),)


    class PresenceLog(models.Model):
        emp_no = models.OneToOneField('Employees', models.DO_NOTHING, db_column='emp_no',
primary_key=True, unique=False)
        from_date = models.DateField()
        to_date = models.DateField(default='9999-01-01')
        class Meta:
            managed = True
            db_table = 'presence_log'
            unique_together = (('emp_no', 'from_date'),)


    class EmpTaskList(models.Model):
```

```
        emp_no = models.OneToOneField('Employees', models.DO_NOTHING, db_column='emp_no',
primary_key=True, unique=False)
        func_no = models.ForeignKey(Functions, models.DO_NOTHING, db_column='func_no')
        new_func_assessment = models.FloatField()
        date = models.DateField()
        to_date = models.DateField()
        add_func = models.IntegerField(default=0)
        class Meta:
            managed = True
            db_table = 'emp_task_list'
            unique_together = (('emp_no', 'func_no', 'new_func_assessment', 'date', 'to_date', 'add_func'),)

class ManAcc(models.Model):
    emp_no = models.OneToOneField('Employees', models.DO_NOTHING, db_column='emp_no',
primary_key=True)
    password = models.CharField(max_length=45)
    class Meta:
        managed = True
        db_table = 'manager_account'
```

views.py

```
import os
import re
from datetime import date
from MySQLdb.constants.FIELD_TYPE import NULL
from django.contrib import messages
from django.shortcuts import render, get_object_or_404
from django.contrib import messages
# Create your views here.
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from django.http import Http404
from django.shortcuts import render, redirect
import time
from collections import Counter
from django.urls import reverse

from .models import *
import mysql.connector
from collections import namedtuple
from .forms import *
```

```
    conn = mysql.connector.connect(
        user='root', password='qwerty8*', host='localhost', database='employees'
      )
      # Creating a cursor object using the cursor() method
    cur = conn.cursor()

    '''
    def index(request):
      latest_question_list = Question.objects.order_by('-pub_date')[:5]
      template = loader.get_template('app/index.html')
      context = {
        'latest_question_list': latest_question_list,
      }
      return HttpResponse(template.render(context, request))
    '''


    def namedtuplefetchall(cursor):
      "Return all rows from a cursor as a namedtuple"
      desc = cursor.description
      nt_result = namedtuple('Result', [col[0] for col in desc])
      return [nt_result(*row) for row in cursor.fetchall()]


    def manager_page(request, employee_id):
      conn = mysql.connector.connect(
        user='root', password='qwerty8*', host='localhost', database='employees'
      )
      # Creating a cursor object using the cursor() method
      cur = conn.cursor()
      print("employee_id - ", employee_id)
      print("employee_id - ", type(employee_id))
      emp_id = Employees.objects.get(emp_no=int(employee_id))
      emp_dept = DeptEmp.objects.filter(to_date='9999-01-01').get(emp_no=emp_id.emp_no)
      # emp_dept = DeptEmp.objects.filter(to_date='9999-01-01').filter(dept_no = emp_dept.dept_no)[:50]

      cur.execute(
        "SELECT employees.gender, employees.emp_no, employees.first_name, employees.last_name,
employees.email,"
        "employees.number, employees.img, titles_name.title_name, departments.dept_name "
        "FROM employees "
```

```
    "INNER JOIN titles ON employees.emp_no = titles.emp_no "
    "INNER JOIN dept_emp ON employees.emp_no = dept_emp.emp_no "
    "INNER JOIN departments ON dept_emp.dept_no = departments.dept_no "
    "INNER JOIN titles_name ON titles.title_no = titles_name.title_no "
    "WHERE titles.to_date='9999-01-01' and dept_emp.to_date='9999-01-01' and "
    "dept_emp.dept_no = %s "
    "ORDER BY employees.emp_no DESC LIMIT 50", (emp_dept.dept_no_id,))
# managerseee = [item for item in cur.fetchall()]
results = namedtuplefetchall(cur)

employyes_info = Employees.objects.all().values()[:8]
departments = Departments.objects.all()
today = date.today()
today = today.strftime('%Y-%m-%d')
presencelog = PresenceLog.objects.filter(to_date__gte=today).filter(replaced=0).values()
presencelog_emp = [emp_no["emp_no_id"] for emp_no in presencelog]
presencelog_emp_replaced = [emp_no["replaced"] for emp_no in presencelog]
print(presencelog_emp)
# results = cur
cur.execute("SELECT employees.emp_no "
        "FROM employees "
        "INNER JOIN dept_manager ON employees.emp_no = dept_manager.emp_no "
        "INNER JOIN departments ON dept_manager.dept_no = departments.dept_no "
        "WHERE dept_manager.to_date='9999-01-01'")

# managers = cur.fetchall()
managers = [item[0] for item in cur.fetchall()]
# managers = cur)
titles = TitlesName.objects.all()

cur.execute("SELECT dept_no FROM departments ORDER BY dept_no DESC LIMIT 1")

dept_no = namedtuplefetchall(cur)
cur.execute("SELECT title_no FROM titles_name ORDER BY title_no DESC LIMIT 1")

title_no = namedtuplefetchall(cur)
cur.execute("SELECT employees.img "
        "FROM employees "
        "ORDER BY employees.emp_no DESC LIMIT 15")
img = namedtuplefetchall(cur)

# Журнал присутності
```

```
        """
        cur.execute("SELECT from_date "
                    "FROM presence_log "
                    "ORDER BY from_date DESC LIMIT 1")
        """
        # from_date = namedtuplefetchall(cur)

        new_department = NewDepartmentsForm()
        new_employee = NewEmployeeForm()
        new_title = EmployeeTitleForm()
        dept_manager = EmployeeManagerForm()

        today = date.today()
        today = today.strftime('%Y-%m-%d')
        cur.execute("SELECT emp_no FROM presence_log WHERE from_date <= %s AND to_date >= %s",
(today, today))
        presence_emp = [item[0] for item in cur.fetchall()]
        if EmpTaskList.objects.filter(date=today).exists():
            day_exist = 0
            #day_exist = 1
        else:
            day_exist = 1

        if request.method == "POST" and 'empPresences' in request.POST:
            from_date = request.POST.get("presence_from_date")
            to_date = request.POST.get("presence_to_date")
            emp_no = request.POST.get("emp_no")
            emp_no = Employees.objects.get(pk=emp_no)
            PresenceLog.objects.create(emp_no=emp_no,
                                       from_date=from_date,
                                       to_date=to_date,
                                       replaced = 1)
            print(request)

            return HttpResponseRedirect(request.path_info)

        if request.method == "POST" and 'addNewEmployee' in request.POST:
            cur.execute("SELECT emp_no FROM employees ORDER BY emp_no DESC LIMIT 1")

            emp_no = namedtuplefetchall(cur)
            emp_no = int(emp_no[0][0]) + 1
            form = NewEmployeeForm(request.POST, request.FILES)
```

```
    # department_name = NewDepartmentsForm(request.POST)
    # department = EmployeeDepartmentForm(request.POST)
    department = request.POST.get("departament_addNewEmployee")
    department = Departments.objects.get(pk=department)
    title = request.POST.get("title")
    title = TitlesName.objects.get(pk=title)
    # dept_manager = EmployeeManagerForm()
    dept_manager = request.POST.get("dept_manager")

    # if form.is_valid() and title.is_valid():
    if form.is_valid():
        employee = Employees.objects.create(birth_date=form.cleaned_data["birth_date"],
                                            first_name=form.cleaned_data["first_name"],
                                            last_name=form.cleaned_data["last_name"],
                                            gender=form.cleaned_data["gender"],
                                            hire_date=form.cleaned_data["hire_date"],
                                            email=form.cleaned_data["email"],
                                            number=form.cleaned_data["number"],
                                            img=form.cleaned_data["img"],
                                            emp_no=emp_no)

        Titles.objects.create(emp_no=employee,
                              title_no=title,
                              from_date=form.cleaned_data["hire_date"])
        # department = department_name.cleaned_data["dept_name"]
        # author = Departments.objects.get(dept_name=request.POST["dept_name"])

        DeptEmp.objects.create(emp_no=employee,
                               dept_no=department,
                               from_date=form.cleaned_data["hire_date"])
        if request.POST.get("manager_check") == "clicked":
            DeptManager.objects.create(emp_no=employee,
                                       dept_no=department,
                                       from_date=form.cleaned_data["hire_date"])
        # form.save()

        return HttpResponseRedirect(request.path_info)
        # return render(request, 'error.html', department)
    # return render(request, 'index.html')
else:
    print("presence_emp - ", presence_emp)
    return render(request, 'manager_page.html',
```

```
                {"employee_id": employee_id,
                 'emp': results, 'departments': departments, 'titles': titles,
                 'new_department': new_department, 'new_employee': new_employee,
                 'new_title': new_title,
                 'dept_manager': dept_manager, 'managers': managers,
                 'presence': presence_emp, 'presencelog_emp': presencelog_emp,
                 'presencelog_emp_replaced': presencelog_emp_replaced,
                 'day_exist': day_exist, 'emp_manager': emp_id})

def login(request):
    conn = mysql.connector.connect(
        user='root', password='qwerty8*', host='localhost', database='employees'
    )
    # Creating a cursor object using the cursor() method
    cur = conn.cursor()
    today = date.today()
    today = today.strftime('%Y-%m-%d')
    '''
    account = ManagerAccountForm()
    if request.method == "POST" and 'login' in request.POST:

        form = ManagerAccountForm(request.POST)
        if form.is_valid():
            email = form.cleaned_data["email"]
            password = form.cleaned_data["password"]



            # form.save()
            return redirect('index')

    return render(request, 'login.html', {'account': account})
    '''

    if request.method == "POST" and 'login_man' in request.POST:
        email = request.POST.get("email")
        password = request.POST.get("password")
        if email == "login" and password == "login":
            return HttpResponseRedirect('index')



        else:
```

```
'''
cur.execute(
   "SELECT employees.emp_no, manager_account.password "
   "FROM employees "
   "INNER JOIN manager_account ON manager_account.emp_no = employees.emp_no "
   "WHERE employees.emp_no = %s ", (emp.emp_no,))
# managerseee = [item for item in cur.fetchall()]
password_check = namedtuplefetchall(cur)[0]
return HttpResponseRedirect(reverse('manager_page', args=(),
                            kwargs={'employee_id': emp.emp_no}))
return HttpResponseRedirect('manager_page/',headers={"employee_id": emp})
return manager_page(request, emp)
'''
#return render(request, 'manager_page.html', {'emp': results, 'emp_manager': emp})

try:
   emp = Employees.objects.get(email=email)
   emp_dept = DeptEmp.objects.filter(to_date='9999-01-01').get(emp_no=emp.emp_no)
   try:

      cur.execute(
         "SELECT employees.emp_no, manager_account.password "
         "FROM employees "
         "INNER JOIN manager_account ON manager_account.emp_no = employees.emp_no "
         "WHERE employees.emp_no = %s ", (emp.emp_no,))
      # managerseee = [item for item in cur.fetchall()]
      password_check = namedtuplefetchall(cur)[0]
      if password_check.password == password:

         return HttpResponseRedirect(reverse('manager_page',
                           args=(),
                           kwargs={'employee_id': emp.emp_no}))
      else:

         messages.warning(request,'Невірний пароль!')
   except:
      messages.warning(request, 'Аккауту не існує!')

except:
   messages.warning(request, 'Невірний логін!')
```

```
        return render(request, 'login.html')


    def index(request):
        employyes_info = Employees.objects.all().values()[:8]
        departments = Departments.objects.all()

        conn = mysql.connector.connect(
            user='root', password='qwerty8*', host='localhost', database='employees'
        )
        # Creating a cursor object using the cursor() method
        cur = conn.cursor()

        cur.execute("SELECT employees.gender, employees.emp_no, employees.first_name,
employees.last_name, employees.email,"
                    "employees.number, employees.img, titles_name.title_name, departments.dept_name "
                    "FROM employees "
                    "INNER JOIN titles ON employees.emp_no = titles.emp_no "
                    "INNER JOIN dept_emp ON employees.emp_no = dept_emp.emp_no "
                    "INNER JOIN departments ON dept_emp.dept_no = departments.dept_no "
                    "INNER JOIN titles_name ON titles.title_no = titles_name.title_no "
                    "WHERE titles.to_date='9999-01-01' and dept_emp.to_date='9999-01-01' "
                    "ORDER BY employees.emp_no DESC LIMIT 100  ")
        #managerseee = [item for item in cur.fetchall()]
        results = namedtuplefetchall(cur)
        today = date.today()
        today = today.strftime('%Y-%m-%d')
        presencelog = PresenceLog.objects.filter(to_date__gte=today).filter(replaced=0).values()
        presencelog_emp = [ emp_no["emp_no_id"] for emp_no in presencelog]
        presencelog_emp_replaced = [ emp_no["replaced"] for emp_no in presencelog]
        #print(presencelog_emp)
        #results = cur
        cur.execute("SELECT employees.emp_no "
                    "FROM employees "
                    "INNER JOIN dept_manager ON employees.emp_no = dept_manager.emp_no "
                    "INNER JOIN departments ON dept_manager.dept_no = departments.dept_no "
                    "WHERE dept_manager.to_date='9999-01-01'")

        #managers = cur.fetchall()
        managers = [item[0] for item in cur.fetchall()]
```

```
#managers = cur)
titles = TitlesName.objects.all()

cur.execute("SELECT dept_no FROM departments ORDER BY dept_no DESC LIMIT 1")

dept_no = namedtuplefetchall(cur)
cur.execute("SELECT title_no FROM titles_name ORDER BY title_no DESC LIMIT 1")

title_no = namedtuplefetchall(cur)
cur.execute("SELECT employees.img "
        "FROM employees "
        "ORDER BY employees.emp_no DESC LIMIT 15")
img = namedtuplefetchall(cur)

# Журнал присутності
"""
cur.execute("SELECT from_date "
        "FROM presence_log "
        "ORDER BY from_date DESC LIMIT 1")
"""
#from_date = namedtuplefetchall(cur)

if request.method == "POST" and 'empPresence' in request.POST:
   from_date = request.POST.get("presence_from_date")
   to_date = request.POST.get("presence_to_date")
   emp_no = request.POST.get("emp_no")
   emp_no = Employees.objects.get(pk=emp_no)
   PresenceLog.objects.create(emp_no=emp_no,
                        from_date=from_date,
                        to_date=to_date)
   return redirect('index')
if request.method == "POST" and 'addNewTitle' in request.POST:
   title_no = title_no[0][0]+1
   form = EmployeeTitleForm(request.POST)
   if form.is_valid():
      #department = Departments(dept_name=form.cleaned_data["dept_name"],dept_no=dept_no)

      #department.save()
      TitlesName.objects.create(title_name=form.cleaned_data["title_name"],title_no=title_no)
      #form.save()
      return redirect('index')
if request.method == "POST" and 'addNewDepartment' in request.POST:
```

```
        dept_no = 'd0'+str(int(dept_no[0][0][1:])+1)
        form = NewDepartmentsForm(request.POST)
        if form.is_valid():
            #department = Departments(dept_name=form.cleaned_data["dept_name"],dept_no=dept_no)

            #department.save()
            Departments.objects.create(dept_name=form.cleaned_data["dept_name"],dept_no=dept_no)
            #form.save()
            return redirect('index')
    if request.method == "POST" and 'addNewEmployee' in request.POST:
        cur.execute("SELECT emp_no FROM employees ORDER BY emp_no DESC LIMIT 1")

        emp_no = namedtuplefetchall(cur)
        emp_no = int(emp_no[0][0])+1
        form = NewEmployeeForm(request.POST, request.FILES)
        #department_name = NewDepartmentsForm(request.POST)
        #department = EmployeeDepartmentForm(request.POST)
        department = request.POST.get("departament_addNewEmployee")
        department = Departments.objects.get(pk = department)
        title = request.POST.get("title")
        title = TitlesName.objects.get(pk=title)
        #dept_manager = EmployeeManagerForm()
        dept_manager = request.POST.get("dept_manager")

        #if form.is_valid() and title.is_valid():
        if form.is_valid():
            #department = Departments(dept_name=form.cleaned_data["dept_name"],dept_no=dept_no)

            #department.save()
            '''
            img = form.cleaned_data["img"]

            if type(img) is None:
                img = None
            '''

            employee = Employees.objects.create(birth_date=form.cleaned_data["birth_date"],
                                first_name=form.cleaned_data["first_name"],
                                last_name=form.cleaned_data["last_name"],
                                gender=form.cleaned_data["gender"],
                                hire_date=form.cleaned_data["hire_date"],
                                email=form.cleaned_data["email"],
```

```
                    number=form.cleaned_data["number"],
                    img=form.cleaned_data["img"],
                    emp_no=emp_no)

        Titles.objects.create(emp_no=employee,
                    title_no=title,
                    from_date=form.cleaned_data["hire_date"])
        #department = department_name.cleaned_data["dept_name"]
        #author = Departments.objects.get(dept_name=request.POST["dept_name"])

        DeptEmp.objects.create(emp_no = employee,
                    dept_no = department,
                    from_date=form.cleaned_data["hire_date"])
        if request.POST.get ("manager_check") == "clicked":
            DeptManager.objects.create(emp_no=employee,
                    dept_no=department,
                    from_date=form.cleaned_data["hire_date"])
        #form.save()

        return redirect('index')
        #return render(request, 'error.html', department)
    #return render(request, 'index.html')
if request.method == "POST" and 'setTasks' in request.POST:
    today = date.today()
    today = today.strftime('%Y-%m-%d')
    if EmpTaskList.objects.filter(date=today).exists():
        pass
        #print("tttttttt")
        EmpTaskList.objects.filter(date=today).delete()
    #print("YYYYYY")
    cur.execute("SELECT date FROM emp_task_list ORDER BY date DESC LIMIT 1")
    day = namedtuplefetchall(cur)
    #print(day)

    if day != []:
        if day[0].date!= today:
            emp_yesterdays_tasks = {}
            yesterdays_tasks = EmpTaskList.objects.filter(to_date__gte=today)

            for emp in yesterdays_tasks:
                #if PresenceLog.objects.filter(emp_no = emp.emp_no).get(to_date__gte=today) == 0:
                print("emp - ", emp.func_no, emp.emp_no)
```

```
                    if emp.emp_no in emp_yesterdays_tasks.keys():
                        print("emp.func_no - ", emp.func_no)
                        if emp.func_no in emp_yesterdays_tasks[emp.emp_no].keys():
                            emp_yesterdays_tasks[emp.emp_no][emp.func_no].append(emp.add_func)
                        else:
                            emp_yesterdays_tasks[emp.emp_no][emp.func_no] = [emp.new_func_assessment,
emp.to_date, emp.add_func]

                    else:
                        emp_yesterdays_tasks[emp.emp_no] = {}
                        if emp.func_no in emp_yesterdays_tasks[emp.emp_no].keys():
                            emp_yesterdays_tasks[emp.emp_no][emp.func_no].append(emp.add_func)
                        else:
                            emp_yesterdays_tasks[emp.emp_no][emp.func_no] = [emp.new_func_assessment,
emp.to_date,
                                                                             emp.add_func]

                    #emp_yesterdays_tasks[emp.emp_no][emp.func_no] = [emp.new_func_assessment,
emp.to_date, emp.add_func]
                print("emp_yesterdays_tasks - ", emp_yesterdays_tasks)

                #cur.execute("SELECT DISTINCT emp_no FROM emp_functions")

                cur.execute("SELECT employees.emp_no FROM employees "
                            "INNER JOIN titles ON employees.emp_no = titles.emp_no "
                            "INNER JOIN dept_emp ON employees.emp_no = dept_emp.emp_no "
                            "WHERE titles.to_date='9999-01-01' and dept_emp.to_date='9999-01-01' "
                            "ORDER BY employees.emp_no DESC LIMIT 100  ")
                # які мають завдання
                employee_listt = namedtuplefetchall(cur)
                print("employee_listt", set(employee_listt))
                employee_list = []
                for emp in employee_listt:
                    try:
                        if EmployeeFunctions.objects.filter(emp_no=emp.emp_no) not in employee_list:
                            employee_list.append(EmployeeFunctions.objects.filter(emp_no=emp.emp_no)[0])
                    except:
                        pass
                print("employee_list - ", employee_list)
                employee_list_instance = []
                for i in employee_list:
```

```
                    cur.execute("SELECT emp_no, func_no, func_assessment FROM emp_functions "
                            "where emp_no = %s", (i.emp_no_id, ))
                    employee_function_list = namedtuplefetchall(cur)
                    employee = Employees.objects.get(pk=i.emp_no_id)

                    for j in employee_function_list:
                        func = Functions.objects.get(pk=j.func_no)
                        if employee in emp_yesterdays_tasks.keys():
                            if func in emp_yesterdays_tasks[employee].keys():
                                if 0 in emp_yesterdays_tasks[employee][func] and 1 in
emp_yesterdays_tasks[employee][func]:
                                    EmpTaskList.objects.create(emp_no=employee,
                                                               func_no=func,
                                                               new_func_assessment=emp_yesterdays_tasks[employee][func][0],
                                                               date=today,
                                                               to_date=emp_yesterdays_tasks[employee][func][1],
                                                               add_func=emp_yesterdays_tasks[employee][func][2])
                                    EmpTaskList.objects.create(emp_no=employee,
                                                               func_no=func,
                                                               new_func_assessment=emp_yesterdays_tasks[employee][func][0],
                                                               date=today,
                                                               to_date=emp_yesterdays_tasks[employee][func][1],
                                                               add_func=emp_yesterdays_tasks[employee][func][3])
                                else:
                                    EmpTaskList.objects.create(emp_no=employee,
                                                               func_no=func,
                                                               new_func_assessment=emp_yesterdays_tasks[employee][func][
                                                                   0],
                                                               date=today,
                                                               to_date=emp_yesterdays_tasks[employee][func][1],
                                                               add_func=emp_yesterdays_tasks[employee][func][2])
                            else:
                                EmpTaskList.objects.create(emp_no=employee,
                                                           func_no=func,
                                                           new_func_assessment=j.func_assessment,
                                                           date=today,
                                                           to_date=today,
                                                           add_func=0)
                        else:
                            EmpTaskList.objects.create(emp_no=employee,
                                                       func_no=func,
                                                       new_func_assessment=j.func_assessment,
```

```
                                    date=today,
                                    to_date=today,
                                    add_func=0)
            # managers = cur.fetchall()
            #managers = [item[0] for item in cur.fetchall()]
            #tasks_employees = namedtuplefetchall(cur)
            #print('!!!!', employee_list_instance)
        else:
            today = date.today()
            today = today.strftime('%Y-%m-%d')
            cur.execute("SELECT DISTINCT emp_functions.emp_no FROM emp_functions "
                    "INNER JOIN titles ON emp_functions.emp_no = titles.emp_no "
                    "INNER JOIN dept_emp ON emp_functions.emp_no = dept_emp.emp_no "
                    "WHERE titles.to_date='9999-01-01' and dept_emp.to_date='9999-01-01'")
            # які мають завдання
            employee_list = namedtuplefetchall(cur)
            employee_list_instance = []
            for i in employee_list:

                cur.execute("SELECT emp_no, func_no, func_assessment FROM emp_functions "
                        "where emp_no = %s", i)
                employee_function_list = namedtuplefetchall(cur)

                for j in employee_function_list:
                    EmpTaskList.objects.create(emp_no=Employees.objects.get(pk=i.emp_no),
                                    func_no=Functions.objects.get(pk=j.func_no),
                                    new_func_assessment=j.func_assessment,
                                    date=today,
                                    to_date=today,
                                    add_func = 0)

            # managers = cur.fetchall()
            # managers = [item[0] for item in cur.fetchall()]
            #tasks_employees = namedtuplefetchall(cur)
            #print('!!!!', employee_list_instance)

        return redirect('index')
        #return HttpResponseRedirect(request.path_info)

    else:
        new_department = NewDepartmentsForm()
        new_employee = NewEmployeeForm()
```

```
            new_title = EmployeeTitleForm()
            dept_manager = EmployeeManagerForm()

            today = date.today()
            today = today.strftime('%Y-%m-%d')
            cur.execute("SELECT emp_no FROM presence_log WHERE from_date <= %s AND to_date >= %s",
(today,today))
            presence_emp = [item[0] for item in cur.fetchall()]
            print("prsencelog - ", presencelog)
            if EmpTaskList.objects.filter(date=today).exists():
                day_exist = 0
                #day_exist = 1
            else:
                day_exist = 1

            print(type(day_exist))
            print(day_exist)
            return render(request, 'index.html',
                          {'employyes_info': results, 'departments': departments, 'titles': titles,
                           'new_department': new_department, 'new_employee': new_employee, 'new_title': new_title,
                           'dept_manager': dept_manager, 'managers': managers,
                           'presence': presence_emp, 'presencelog_emp': presencelog_emp,
                           'presencelog_emp_replaced': presencelog_emp_replaced,
                           'day_exist': day_exist})


        def edit(request, employee_id):
            conn = mysql.connector.connect(
                user='root', password='qwerty8*', host='localhost', database='employees'
            )
            # Creating a cursor object using the cursor() method
            cur = conn.cursor()

            employee = Employees.objects.get(pk=employee_id)
            employee_title = Titles.objects.filter(pk=employee_id)
            #print(employee_title)
            employee_department = DeptEmp.objects.filter(pk=employee_id)
            employee_manager = DeptManager.objects.filter(pk=employee_id)
            image = employee.img
            department = Departments.objects.all()
            titles_name = TitlesName.objects.all()
            functions = Functions.objects.all()
```

```
cur.execute("SELECT DISTINCT emp_no FROM dept_manager")

managers = [item[0] for item in cur.fetchall()]
if int(employee_id) in managers:
   dept_manager = DeptManager.objects.filter(pk=employee_id)
else:
   dept_manager = None
emp_function = EmployeeFunctions.objects.filter(pk=employee_id)
#print('emp_function', emp_function)

if request.method == "POST" and 'addManager' in request.POST:

   if request.POST.get("to_date_manager"):
      emp_to_date = dept_manager.get(to_date='9999-01-01')
      emp_to_date.to_date=request.POST.get("to_date_manager")
      emp_to_date.save()



   dept_no = request.POST.get("manager_option")
   dept_no = Departments.objects.get(dept_no=dept_no)



   DeptManager.objects.create(emp_no=employee,
                  dept_no=dept_no,
                  from_date=request.POST.get("new_manager-from_date"))

   return HttpResponseRedirect(request.path_info)
if request.method == "POST" and 'deleteManager' in request.POST:

   dept_name = request.POST.get("delete_manager")
   dept_no = Departments.objects.get(dept_name=dept_name)
   employee_departments = DeptManager.objects.filter(pk=employee_id)
   employee_departments = employee_departments.filter(dept_no=dept_no)
   employee_departments.delete()
   return HttpResponseRedirect(request.path_info)

if request.method == "POST" and 'deleteTitle' in request.POST:

   title_name = request.POST.get("delete_title")
   title_from_date = request.POST.get("delete_title_from_date")
   title_no = TitlesName.objects.get(title_name=title_name)
```

```
    employee_titles = Titles.objects.filter(pk=employee_id)
    employee_titles = employee_titles.filter(title_no=title_no)
    employee_titles = employee_titles.filter(from_date=title_from_date)
    employee_titles.delete()
    return HttpResponseRedirect(request.path_info)

if request.method == "POST" and 'deleteDepartment' in request.POST:

    dept_name = request.POST.get("delete_departmentname")
    dept_no = Departments.objects.get(dept_name=dept_name)
    employee_departments = DeptEmp.objects.filter(pk=employee_id)
    employee_departments = employee_departments.filter(dept_no=dept_no)
    employee_departments.delete()
    return HttpResponseRedirect(request.path_info)

if request.method == "POST" and 'addTitle' in request.POST:

    if request.POST.get("title_to_date"):
        title_to_date = employee_title.get(to_date='9999-01-01')
        title_to_date.to_date=request.POST.get("title_to_date")
        title_to_date.save(update_fields=['to_date'])

    if request.POST.get("new_title-to-date"):
        to_date = request.POST.get("new_title-to-date")
    else:
        to_date = '9999-01-01'
    title_name = request.POST.get("new_title")
    title_name = TitlesName.objects.get(pk=title_name)

    Titles.objects.create(emp_no=employee,
                          title_no=title_name,
                          from_date=request.POST.get("new_title-from_date"),
                          to_date=to_date)

    return HttpResponseRedirect(request.path_info)

if request.method == "POST" and 'addDepartment' in request.POST:

    if request.POST.get("department-to_date"):
        emp_to_date = employee_department.get(to_date='9999-01-01')
        emp_to_date.to_date=request.POST.get("department-to_date")
        emp_to_date.save()
```

```
    dept_name = request.POST.get("department_option")
    department_name = Departments.objects.get(pk=dept_name)

    #from_date = date_str(request.POST.get("new_department-from_date"))
    if request.POST.get("new_department-to_date"):
        to_date = request.POST.get("new_department-to_date")
    else:
        to_date = '9999-01-01'

    DeptEmp.objects.create(emp_no=employee,
                           dept_no=department_name,
                           from_date= request.POST.get("new_department-from_date"),
                           to_date = to_date)
    return HttpResponseRedirect(request.path_info)
if request.method == "POST" and 'addNewFunction' in request.POST:
    cur.execute("SELECT func_no FROM functions ORDER BY func_no DESC LIMIT 1")

    func_no = namedtuplefetchall(cur)

    Functions.objects.create(func_no=func_no[0][0] + 1,
                           func_name=request.POST.get("new_function"))
    return HttpResponseRedirect(request.path_info)
if request.method == "POST" and 'addNewEmpFunction' in request.POST:
    func_name = request.POST.get("func_option")
    func_no = Functions.objects.get(pk=func_name)

    EmployeeFunctions.objects.create(emp_no=employee,
                           func_no=func_no,
                           func_assessment=request.POST.get("new_function_assessment"))
    return HttpResponseRedirect(request.path_info)
if request.method == "POST" and 'editEmployee' in request.POST:
   # item = Employees.objects.get(pk=employee_id)
    #form = EditEmployeeForm(request.POST or None, request.FILES or None, instance=item)
    #if form.is_valid():

     #   form.save()
      #  messages.success(request, ('The item has been edited successfully!'))
       # return redirect('index')
    #return render(request, 'index.html')
   employee.emp_no  = request.POST.get("emp_no")
   employee.birth_date = request.POST.get("birth_date")
```

```
    employee.first_name = request.POST.get("first_name")
    employee.last_name = request.POST.get("last_name")
    employee.gender = request.POST.get("gender")
    employee.hire_date = request.POST.get("hire_date")
    employee.email = request.POST.get("email")
    employee.number = request.POST.get("number")
    if request.FILES:
        employee.img = request.FILES.get("img")
    else:
        employee.img = image
    employee.save()

    # return render(request, 'index.html', {'employee': employee, 'titles': titles})
    #return redirect('index')
    next = request.POST.get('next', '/')
    print('next - ', next)
    return HttpResponseRedirect(next)

 if request.method == "POST" and 'editPassword' in request.POST:

    password = request.POST.get("password")
    try:
        ManAcc.objects.get(emp_no=employee_id).delete()
        ManAcc.objects.create(emp_no=Employees.objects.get(emp_no=employee_id),
                              password=password)
    except:
         ManAcc.objects.create(emp_no=Employees.objects.get(emp_no=employee_id),
                               password=password)


#return render(request, 'index.html', {'employee': employee, 'titles': titles})
    return HttpResponseRedirect(request.path_info)
 else:
     account = None
     if dept_manager != None:
         print("password - ", account)

         try:
             DeptManager.objects.filter(emp_no=employee_id).get(to_date='9999-01-01')
             print("password - ", account)
             try:
                 ManAcc.objects.get(emp_no=employee_id)
```

```
                print("password - ", account)
                account = ManAcc.objects.get(emp_no=employee_id)
            except:
                account = None
        except:
            account = None
    print("password - ", account)
    birth_date = datee(employee.birth_date)
    hire_date = datee(employee.hire_date)
    return render(request, 'edit.html', {'employee': employee, "employee_title": employee_title,
                                         'employee_department': employee_department,
                                         'dept_manager': dept_manager,
                                         'departments': department,
                                         'titles': titles_name,
                                         'birth_date': birth_date,
                                         'hire_date': hire_date,
                                         'date': date,
                                         'functions': emp_function,
                                         'functions_all':functions,
                                         'account': account})

def func_stability(emp_list, emp_func_stability, to_date, employee_id):
    today = date.today()
    today = today.strftime('%Y-%m-%d')
    v = {}
    func_stab = {}
    # просто індекс
    id = 0
    start = time.time()

    for i in range(len(emp_list)-1):
        # перебираєм перший
        for j in emp_func_stability[Employees.objects.get(pk=emp_list[i])]:
            for k in range(i+1, len(emp_func_stability)):
                #print(j[0])
                # перебираєм другий
                next_emp_func = emp_func_stability[Employees.objects.get(pk=emp_list[k])]
                for next_emp_func_i in next_emp_func:
                    if type(j)==tuple and len(j) != 1:
                        if len(list(filter(lambda i: i in j, next_emp_func_i)))==0:
                            v[(id, emp_list[i], emp_list[k])] = {}
                            v[(id, emp_list[i],emp_list[k])] = (j, next_emp_func_i)
```

```
                        id += 1
                        #print(emp_list[i],emp_list[k], j, next_emp_func_i)

                    else:
                        if j != next_emp_func_i:
                            if j[0] not in next_emp_func_i:
                                v[(id, emp_list[i], emp_list[k])] = {}
                                #print(emp_list[i], emp_list[k], j, next_emp_func_i)
                                v[(id, emp_list[i], emp_list[k])] = (j, next_emp_func_i)
                                id += 1
            end = time.time()
            print("Two employee list time - ", str(end - start))

            start = time.time()
            emp_absent = Employees.objects.get(pk=employee_id)
            end = time.time()
            print("1 - ", str(end - start))
            start = time.time()
            for ind in list(v.keys()):
                help_dict = {}

                emp = ind

                for num_emp_key in range(1,len(emp)):

                    for emp_func in v[emp][num_emp_key-1]:
                        #print(emp_func)
                        #print(emp[num_emp_key])
                        u =
EmployeeFunctions.objects.filter(emp_no=Employees.objects.get(pk=emp[num_emp_key])).get(func_no=emp_fun
c.func_no)
                        add = 0
                        assessment = change(u, emp_absent, today)
                        help_dict[(Employees.objects.get(pk=emp[num_emp_key]),
Functions.objects.get(func_no=emp_func.func_no),
                                today, to_date, add)] = assessment
                        add = 1
                        help_dict[(Employees.objects.get(pk=emp[num_emp_key]),
Functions.objects.get(func_no=emp_func.func_no),
                                today, to_date, add)] = assessment
```

```
allemp = EmpTaskList.objects.filter(date=today)
tup = ()

for t in v[emp]:
    tup += t

for cfunc in allemp:
    assessment = cfunc.new_func_assessment
    if cfunc.emp_no != emp_absent:
        if cfunc.emp_no_id not in emp:
            add = 0
            help_dict[
                (cfunc.emp_no, cfunc.func_no_id,
                 today, cfunc.to_date, add)] = assessment

        else:
            if cfunc.func_no not in v[emp][emp.index(cfunc.emp_no_id)-1]:
                add = 0
                help_dict[
                    (cfunc.emp_no, cfunc.func_no_id,
                     today, cfunc.to_date, add)] = assessment

    else:
        c = Functions.objects.get(func_no=cfunc.func_no_id)
        if c not in tup:
            add = 0
            help_dict[
                (cfunc.emp_no, cfunc.func_no_id,
                 today, cfunc.to_date, add)] = 0


#distinct_assessments = help.objects.values('assessment').distinct()
distinct_assessments = list(help_dict.values())
count_distinct_assessments = Counter(distinct_assessments)
distinct_assessments = list(set(distinct_assessments))
a = {}
wam = 0
sum = 0
for i in distinct_assessments:
    a[i] = count_distinct_assessments[i]
    sum += a[i]
    wam += a[i] * i
```

```
                wam /= sum
                # print("WAAM",wam)
                # func_emp = (j.emp_no,(j.func_no, j.emp_no))
                func_emp = tuple(emp)+tuple(v[emp])
                # func_stab[func_emp] = wam

                func_stab[func_emp] = wam
            print('func_stab - ', func_stab)
            end = time.time()
            print("4 - ", str(end-start))

            max_key = max(func_stab, key=func_stab.get)

            max_func_stab = {}
            print("max_key", max_key)
            k = (max_key[0], Employees.objects.get(emp_no=max_key[1]),
Employees.objects.get(emp_no=max_key[2]),  max_key[3:])
            print("k - ", k)
            max_func_stab[max_key] = func_stab[max_key]
            print("max_func_stab - ", max_func_stab)
            return max_func_stab

        def employee_replacement(employee_id, emp_functions, all_employee_functions):
            func_stab = {}
            func_stab2 = {}
            today = date.today()
            today = today.strftime('%Y-%m-%d')
            # нікому не передавать функції
            task_list = EmpTaskList.objects.filter(emp_no=employee_id).filter(date=today)
            presence = PresenceLog.objects.filter(emp_no=employee_id)
            presence = presence.filter(to_date__gte=today)[0]

            # розділяємо функції

            func_list = Functions.objects.all()
            same_functions = {}

            emp_same_func = []
            for i in task_list:
                same_functions[i.func_no] = []

                emp_same_task = EmpTaskList.objects.filter(func_no=i.func_no).filter(date=today)
```

```
                emp_same_task_list = []
                for same_task in list(emp_same_task):
                    if list(emp_same_task).count(same_task) < 2:
                        emp_same_task_list.append(same_task)
                print("emp_same_task - ", emp_same_task)
                print("emp_same_task_list - ", emp_same_task_list)
                emp_same_task = emp_same_task_list
                for j in emp_same_task:
                    #print(Employees.objects.get(emp_no=j.emp_no).value())
                    if j.emp_no != i.emp_no:
                        if j.new_func_assessment != 0.0 and
len(EmpTaskList.objects.filter(func_no=i.func_no).filter(date=today).filter(emp_no=j.emp_no_id)) < 2:
                            emp_no = Employees.objects.get(emp_no=j.emp_no_id).emp_no
                            #print(str(j.emp_no))
                            #EmpTaskList.objects.filter(emp_no=j.emp_no)
                            #emp = Employees.objects.get(emp_no=j.emp_no)
                            same_functions[i.func_no].append(j)
                            #emp_no = getattr(j,'emp_no')
                            if emp_no not in emp_same_func:
                                emp_same_func.append(emp_no)

            help.objects.all().delete()

            func_stab = {}
            for i in same_functions:
                for j in same_functions[i]:

                    func_emp = (j.func_no)
                    func_stab2 = {}
                    func_stab2[j.func_no] = 0
                    try:
                        if not func_stab[j.emp_no]:
                            pass
                    except:
                        func_stab[j.emp_no] = {}
                    func_stab[j.emp_no][func_emp] = func_stab2[func_emp]

            v = []
            start = time.time()
            for i in emp_same_func:
```

```
v.append({(Employees.objects.get(pk=i)):subsets(list(func_stab[(Employees.objects.get(pk=i))].keys()))})

            end = time.time()
            print("recursion - ", str(end - start))
            func_stab = {}
            # всі функції 0
            all_emp = EmpTaskList.objects.filter(date=today)
            help_dict = {}
            for emp in all_emp:
                if emp.emp_no != Employees.objects.get(emp_no = employee_id):

                    help_dict[(emp.emp_no, emp.func_no, today, presence.to_date, emp.add_func)] =
emp.new_func_assessment



                else:

                    help_dict[(emp.emp_no, emp.func_no, today, presence.to_date, emp.add_func)] = 0

            distinct_assessments = list(help_dict.values())
            count_distinct_assessments = Counter(distinct_assessments)
            distinct_assessments = list(set(distinct_assessments))
            a = {}
            wam = 0
            sum = 0
            for i in distinct_assessments:
                a[i] = count_distinct_assessments[i]
                sum += a[i]
                wam += a[i] * i
            wam /= sum

            func_zero = {}
            func_zero[employee_id] = wam
            #one_emp_func_stab(v, today, presence, func_stab)
            start = time.time()
            help_dict = {}
            emp_absent = Employees.objects.get(pk=employee_id)
            for i in v:
                #print('i  ', i)
                #print(i[Employees.objects.filter(emp_no=i.emp_no)])
                for j in i:
```

```
                #print("j  ", j)
                #for k in i[j]:
                help_dict = {}
                for k in i[j]:
                    help_dict = {}
                    #print('k  ', k)
                    for e in k:
                        #print('e  ', e)
                        u = EmployeeFunctions.objects.filter(emp_no=j.emp_no).get(func_no=e.func_no)
                        add = 0
                        assessment = change(u, emp_absent, today)
                        help_dict[(j,e,today,presence.to_date, add)] = assessment

                        add = 1
                        help_dict[(j, e, today, presence.to_date, add)] = assessment

                    allemp = EmpTaskList.objects.filter(date=today)
                    if k!=[]:
                        for cfunc in allemp:

                            if cfunc.emp_no != emp_absent:
                                if cfunc.emp_no != j:
                                    add = 0
                                    help_dict[(cfunc.emp_no, cfunc.func_no, today, cfunc.to_date, add)] =
cfunc.new_func_assessment

                                else:
                                    if cfunc.func_no not in k:
                                        add = 0
                                        help_dict[(
                                        cfunc.emp_no, cfunc.func_no, today, cfunc.to_date, add)] =
cfunc.new_func_assessment

                            else:
                                if cfunc.func_no not in k:
                                    add = 0
                                    help_dict[(
                                    cfunc.emp_no, cfunc.func_no, today, cfunc.to_date, add)] = 0

                        distinct_assessments = list(help_dict.values())
                        count_distinct_assessments = Counter(distinct_assessments)
                        distinct_assessments = list(set(distinct_assessments))
```

```
            a = {}

            wam = 0
            sum = 0
            for i in distinct_assessments:
                a[i] = count_distinct_assessments[i]
                sum += a[i]
                wam += a[i] * i
            wam /= sum



            func_emp = tuple(k)
            # func_stab[func_emp] = wam
            func_stab2 = {}
            func_stab2[tuple(k)] = wam
            try:
                if not func_stab[j]:
                    pass
            except:
                func_stab[j] = {}
            func_stab[j][func_emp] = func_stab2[func_emp]
print("func_stab - ", func_stab)

end = time.time()
print("one employee time - ", str(end - start))
max = 0
max_one_emp = {}
for i in list(func_stab.keys()):
    for j in list(func_stab[i].keys()):
        #print(func_stab[i][j])
        if func_stab[i][j] > max:
            max = func_stab[i][j]
            max_one_emp = {}
            max_one_emp[(i, j)] = max

print("max_one_emp - ", max_one_emp)
start = time.time()
max_func_stab = func_stability(emp_same_func, func_stab, presence.to_date, employee_id)
end = time.time()
print("max_func_stab - ", max_func_stab)
print("Two employee time - ", str(end - start))
```

```
    return task_list, wam, max_func_stab, max_one_emp, func_zero

def replacement(request, employee_id):
    today = date.today()
    today = today.strftime('%Y-%m-%d')
    presence = PresenceLog.objects.filter(emp_no=employee_id)
    presence = presence.filter(to_date__gte=today)[0]
    emp_absent = Employees.objects.get(pk=employee_id)
    if request.method == "POST" and 'zero_replacement' in request.POST:
        presence = PresenceLog.objects.filter(emp_no=employee_id)
        presence = presence.filter(to_date__gte=today)[0]
        EmpTaskList.objects.filter(emp_no=employee_id).filter(date=today).delete()
        for func in EmployeeFunctions.objects.filter(emp_no=Employees.objects.get(emp_no=employee_id)):
            EmpTaskList.objects.create(
                emp_no = Employees.objects.get(emp_no=employee_id),
                func_no = func.func_no,
                new_func_assessment = 0,
                date = today,
                to_date = presence.to_date,
                add_func = 0
            )

    if request.method == "POST" and 'one_replacement' in request.POST:
        one_emp = request.POST.get("one_emp")
        print(one_emp)

        counter = request.POST.get("one_emp_func")
        func_nos = re.findall('[0-9]+', counter)
        func_nos = [int(num) for num in func_nos]
        emp = Employees.objects.get(emp_no=int(one_emp))
        for func in func_nos:
            f = Functions.objects.get(func_no=func)
            EmpTaskList.objects.filter(emp_no=employee_id).filter(date=today).filter(func_no=f).delete()
            one_emp_func = EmpTaskList.objects.filter(emp_no=emp).filter(date=today).get(func_no=f)
            #one_emp_func = one_emp_func.get(add_func=0)
            a = EmployeeFunctions.objects.filter(emp_no=emp).get(func_no=f).func_assessment
            #print(a)
            b = change( EmployeeFunctions.objects.filter(emp_no=emp).get(func_no=f), emp_absent, today)
            print(type(b))


            EmpTaskList.objects.filter(emp_no=emp).filter(date=today).filter(func_no=f).delete()
```

```
        EmpTaskList.objects.create(emp_no=emp,
                                   func_no=f,
                                   new_func_assessment=b,
                                   date=today,
                                   to_date=presence.to_date,
                                   add_func=1
                                   )
        EmpTaskList.objects.create(emp_no=emp,
                                   func_no=f,
                                   new_func_assessment=b,
                                   date=today,
                                   to_date=presence.to_date,
                                   add_func=0
                                   )

    print(func_nos)
    zero_func = EmpTaskList.objects.filter(emp_no=employee_id).filter(date=today)
    emp = Employees.objects.get(emp_no=employee_id)
    for func in zero_func:
        EmpTaskList.objects.filter(emp_no=emp).filter(date=today).filter(func_no=func.func_no).delete()
        EmpTaskList.objects.create(emp_no=emp,
                                   func_no=func.func_no,
                                   new_func_assessment=0,
                                   date=today,
                                   to_date=presence.to_date,
                                   add_func=0)


if request.method == "POST" and 'two_replacement' in request.POST:
    emp1 = request.POST.get("first_emp")
    print("emp1  --- ", emp1)
    print("emp1  --- ", type(emp1))
    emp1 = Employees.objects.get(emp_no=int(emp1))

    counter1 = request.POST.get("first_emp_func")
    func_nos1 = re.findall('[0-9]+', counter1)
    func_nos1 = [int(num) for num in func_nos1]
    emp2 = request.POST.get("second_emp")
    emp2 = Employees.objects.get(emp_no=int(emp2))
    counter2 = request.POST.get("second_emp_func")
    func_nos2 = re.findall('[0-9]+', counter2)
    func_nos2 = [int(num) for num in func_nos2]
```

```
            emp_list = [emp1, emp2]
            func_list = [func_nos1, func_nos2]
            for emp in range(len(emp_list)):
                for func in func_list[emp]:
                    f = Functions.objects.get(func_no=func)
                    EmpTaskList.objects.filter(emp_no=employee_id).filter(date=today).filter(func_no=f).delete()
                    one_emp_func =
EmpTaskList.objects.filter(emp_no=emp_list[emp]).filter(date=today).get(func_no=f)
                    #one_emp_func = one_emp_func.get(add_func=0)
                    a = EmployeeFunctions.objects.filter(emp_no=emp_list[emp]).get(func_no=f).func_assessment
                    #print(a)
                    b = change( EmployeeFunctions.objects.filter(emp_no=emp_list[emp]).get(func_no=f),
emp_absent, today)
                    print(type(b))


                    EmpTaskList.objects.filter(emp_no=emp_list[emp]).filter(date=today).filter(func_no=f).delete()
                    EmpTaskList.objects.create(emp_no=emp_list[emp],
                                               func_no=f,
                                               new_func_assessment=b,
                                               date=today,
                                               to_date=presence.to_date,
                                               add_func=1
                                               )
                    EmpTaskList.objects.create(emp_no=emp_list[emp],
                                               func_no=f,
                                               new_func_assessment=b,
                                               date=today,
                                               to_date=presence.to_date,
                                               add_func=0
                                               )


                print(func_nos1)
            zero_func = EmpTaskList.objects.filter(emp_no=employee_id).filter(date=today)
            emp = Employees.objects.get(emp_no=employee_id)
            for func in zero_func:
                EmpTaskList.objects.filter(emp_no=emp).filter(date=today).filter(func_no = func.func_no).delete()
                EmpTaskList.objects.create(emp_no=emp,
                                           func_no=func.func_no,
                                           new_func_assessment=0,
                                           date=today,
                                           to_date=presence.to_date,
```

```
                                add_func=0)

            if request.method == "POST" and 'one_replacement' in request.POST or \
                request.method == "POST" and 'two_replacement' in request.POST or \
                request.method == "POST" and 'zero_replacement' in request.POST:
              emp = Employees.objects.get(emp_no=employee_id)
              p = PresenceLog.objects.filter(emp_no=emp).get(to_date__gte=today)
              PresenceLog.objects.filter(emp_no=emp).filter(to_date__gte=today).delete()
              print("p", p)
              PresenceLog.objects.create(emp_no=emp,
                                from_date = p.from_date,
                                to_date = p.to_date,
                                replaced = 0)




              return redirect('index')
            else:
              employee = Employees.objects.get(pk=employee_id)
              emp_function = EmployeeFunctions.objects.filter(pk=employee_id)
              employee_department = DeptEmp.objects.filter(pk=employee_id)
              employee_department = employee_department.get(to_date='9999-01-01')
              employee_title = Titles.objects.filter(pk=employee_id)
              employee_title = employee_title.get(to_date='9999-01-01')
              all_employee_functions = EmployeeFunctions.objects.all()

              result, wam, max_func_stab, max_one_emp, func_zero = employee_replacement(employee_id,
emp_function,
                                                                    all_employee_functions)
              #result, wam, max_one_emp, func_zero = employee_replacement(employee_id,
emp_function,all_employee_functions)
              # Записать в журнал присутності
              #print('list(max_func_stab.keys())', list(max_func_stab.keys()))
              # функції не виконуються
              zero_emp = int(list(func_zero.keys())[0])
              zero_func_stab = {'emp': Employees.objects.get(emp_no=zero_emp),
                        'func_stab': func_zero[list(func_zero.keys())[0]]
                        }
              # функції одному працівнику
              one_emp = list(max_one_emp.keys())[0][0]
              print(one_emp)
```

```
            print(type(one_emp))
            one_emp_func = list(max_one_emp.keys())[0][1]
            one_employee_func_stab = {
               'emp': one_emp,
               #'funcs': [Functions.objects.get(func_name = one_emp_func[i].func_name).func_no for i in
range(len(one_emp_func))],
               'funcs': one_emp_func,
               'func_stab': max_one_emp[ list(max_one_emp.keys())[0] ]
            }

            # функції на двох
            first_emp = list(max_func_stab.keys())[0][1]
            print("first_emp", first_emp)
            first_emp_func = list(max_func_stab.keys())[0][3]
            second_emp = list(max_func_stab.keys())[0][2]
            second_emp_func = list(max_func_stab.keys())[0][4]
            two_employee_func_stab = {'emp1': Employees.objects.get(emp_no=first_emp),
                        #'emp1_func': [first_emp_func[i].func_name for i in range(len(first_emp_func))],
                        'emp1_func': first_emp_func,
                        'emp2': Employees.objects.get(emp_no=second_emp),
                        'emp2_func': second_emp_func,
                        'func_stub': max_func_stab[list(max_func_stab.keys())[0]],
                        }

            return render(request, 'replacement.html',{'employee': employee,
                                'functions': emp_function,
                                'employee_department': employee_department,
                                'employee_title': employee_title,
                                'result': result,
                                'max_func_stab': max_func_stab,
                                'max_one_emp': max_one_emp,
                                'func_zero': func_zero,
                                'two_employee_func_stab': two_employee_func_stab,
                                'zero_func_stab': zero_func_stab,
                                'one_employee_func_stab': one_employee_func_stab})


        def change(emp, emp_absent, day):
            print("emp - ", emp)
            emp_a_dept = DeptEmp.objects.filter(emp_no = emp_absent.emp_no).get(to_date='9999-01-01')
            e_dept = DeptEmp.objects.filter(emp_no = emp.emp_no).get(to_date='9999-01-01')
            emp_a_title = Titles.objects.filter(emp_no = emp_absent.emp_no).get(to_date='9999-01-01')
```

```
    e_title = Titles.objects.filter(emp_no = emp.emp_no).get(to_date='9999-01-01')

    print("emp_a.dept_no - ", e_dept.dept_no)
    print("emp_a.dept_no - ", emp_a_dept.dept_no)
    new_assesment = emp.func_assessment - emp.func_assessment*0.05
    if  e_dept.dept_no != emp_a_dept.dept_no:
        new_assesment = new_assesment - new_assesment*0.1
    if e_title.title_no != emp_a_title.title_no:
        new_assesment = new_assesment - new_assesment * 0.07




    return new_assesment


def datee(date):
    if date.month < 10:
        month = '0' + str(date.month)
    else:
        month = str(date.month)
    if date.day < 10:
        day = '0' + str(date.day)
    else:
        day = str(date.day)

    new_date = str(date.year) + '-' + month + '-' + day

    return new_date

def date_str(date):
    x = date.split('.')
    return str(x[2])+'-'+str(x[1])+'-'+str(x[0])
def posts_edit(request, id=None):
    instance = get_object_or_404(DeptEmp, id=id)
    context={
        'instance': instance
    }
    return render(request, 'modal.html', context)

def subsets(nums):
    rslt = []
```

```
    def dfs(temp, idx):
        rslt.append(temp[:])
        for i in range(idx, len(nums)):
            temp.append(nums[i])
            # backtrack
            dfs(temp, i + 1)
            temp.pop()

    dfs([], 0)
    return rslt
```

urls.py

```
from django.conf.urls.static import static
from django.urls import path
from .views import *
from django.conf import settings
urlpatterns = [
    # ex: /polls/
    path('', login, name='login'),
    path('replacement/<employee_id>', replacement, name='replacement'),
    path('manager_page/<employee_id>/edit', edit, name='edit'),
    path('manager_page/<employee_id>', manager_page, name='manager_page'),
    path('manager_page/edit/', edit, name='edit'),
    path('edit/<employee_id>', edit, name='edit'),
    path('edit/', edit, name='edit'),
    path('index', index, name='index')
]
```