

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Київський національний університет будівництва і архітектури

## **Архітектура інформаційних систем**

Методичні вказівки до виконання лабораторних робіт  
для підготовки здобувачів другого магістерського рівня вищої освіти  
спеціальностей 121 «Інженерія програмного забезпечення»,  
122 «Комп'ютерні науки» та  
126 «Інформаційні системи і технології»

Київ 2024

УДК 004.42

А

Укладачі: Т.А. Гончаренко, канд. техн. наук, доцент;  
С.В. Білощицька, доктор техн. наук, професор

Рецензент Ю.В. Рябчун, доктор філософії, доцент

Відповідальна за випуск Т.А. Гончаренко, канд. техн. наук,  
доцент

*Затверджено на засіданні кафедри інформаційних технологій,  
протокол № 11 від 20 травня 2024 року.*

В авторській редакції.

**Архітектура інформаційних систем:** методичні вказівки до виконання лабораторних робіт / уклад.: Гончаренко Т.А., Білощицька С.В. – Київ: КНУБА, 2024. – 48с.

Містять зміст, порядок оформлення і вказівки до виконання окремих лабораторних робіт.

Призначено для здобувачів спеціальностей 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки» та 126 «Інформаційні системи і технології»

© КНУБА, 2024

## ЗМІСТ

Загальні положення	4
Лабораторна робота №1	5
Лабораторна робота №2	10
Лабораторна робота №3	13
Лабораторна робота №4	17
Лабораторна робота №5	18
Лабораторна робота №6	20
Лабораторна робота №7	25
Лабораторна робота №8	28
Лабораторна робота №9	40
Список літератури	47

## Загальні положення

Мета вивчення освітньої компоненти «**Архітектура інформаційних систем**» полягає в підготовці здобувачів рівня «Магістр» до успішної кар'єри у сфері проектування базової архітектури інформаційної технології та розробки програмного забезпечення, надаючи їм необхідні знання, навички та компетентності. Освітня компонента «Архітектура інформаційних систем» вивчає фундаментальні архітектурні принципи і концепції, які лежать в основі розробки програмних систем. Основною метою є ідея зниження складності системи шляхом абстракції і розмежування повноважень.

Освітня компонента спрямована на розвиток у здобувачів здатності аналізувати вимоги до програмної системи та здатності проектувати відповідну архітектуру, враховуючи різні аспекти, такі як модульність, ефективність, масштабованість, безпека тощо. Засвоєння різноманітних методологій розробки програмного забезпечення дасть змогу здобувачам виробити у них навички вибору найбільш ефективних методів для конкретних проектів.

Освітня компонента також сприятиме розвитку комунікаційних навичок, здобувачі мають вміти чітко та зрозуміло спілкуватися з іншими членами команди розробки, включаючи менеджерів, програмістів, тестувальників тощо, щоб ефективно обговорювати архітектурні питання та приймати рішення.

Задача даної компоненти полягає не лише в теоретичному засвоєнні матеріалу, але й у підготовці здобувачів до роботи з реальними проблемами та завданнями, які вони можуть зустріти у процесі професійної діяльності в галузі інформаційних технологій.

Основні програмні результати навчання формують навички використання архітектурних патернів проектування (MVC, MVVM, Client-Server) і їхню взаємодію між компонентами; інформаційні технології та інструментальні засоби для проектування архітектури: архітектурні фреймворки (Spring, .NET), інструменти моделювання (UML, ArchiMate) та архітектурні засоби захисту (наприклад, firewalls, encryption); тенденції та інновації в архітектурі інформаційних систем: Cloud computing та мікросервісна архітектура, Інтернет речей (IoT) та вбудовані системи Big Data архітектури.

## Лабораторна робота №1

### Місія та цілі підприємства

**Мета роботи:** Навчитись будувати мотиваційну модель архітектури підприємства у середовищі Archimate.

#### Завдання на лабораторну роботу

1. Обрати тему дослідження для подальшої роботи
2. Визначити місію підприємства
3. Визначити цінності підприємства не менше 5
4. Визначити цілі не менше 15
5. Побудувати діаграми бачення місії, цінності та бізнесу підприємства (Mission-Values-Vision View) у середовищі Archimate (<https://www.archimatetool.com/>).
6. Побудувати діаграму бачення стратегії підприємства

#### Теоретичні відомості

*ArchiMate* – це графічна мова моделювання архітектури підприємства.

Вона призначений для опису, аналізу та візуалізації побудови різних складових корпоративної архітектури підприємства, відношень і взаємозв'язків між ними.

В даний час ArchiMate є стандартом міжнародного консорціуму The Open Group. ArchiMate доповнює стандарт архітектури підприємства The Open Group Architecture Framework (TOGAF).

Основними поняттями мови ArchiMate є такі:

- *елементи* – це сутності різного змісту, форми і призначення;
- *зв'язки між елементам* – це з'єднання, що вказують на відношення між елементами.

Існує три класи елементів:

- *структурні елементи*;
- *зовнішнє і внутрішнє бачення системи*;
- *елементи власності*.

Структурні елементи поділяються на активні, пасивні та поведінкові.

*Активний структурний елемент* (active structure element) – це сутність, яка здатна виконувати певні дії. Це можуть бути бізнес-виконавці, компоненти додатків або пристрої, які реально виконують ті чи інші дії.

*Пасивний структурний елемент* (passive structure element) – це певний об'єкт, на якому виконуються дії. Як правило, це інформаційні об'єкти або об'єкти даних, також вони можуть бути використані для подання фізичних об'єктів, над якими виконуються ті чи інші дії.

*Елемент поведінки* (behavior element) – це деяка одиниця дії, що виконується одним або декількома активними структурними елементами.

*Елементами поведінки* можуть бути процеси, функціонали (функції), сервіси та події. Вони призначаються активним структурним елементам, щоб показати, хто або що виконує дії.

*Діаграма бачення місії, цінності та бізнесу підприємства* використовується для представлення місії, бачення та основних цінностей організації.

*Місія підприємства* – це чітко виражена причина існування підприємства на ринку. Місія деталізує статус підприємства та вказує напрям і орієнтири для визначення цілей і стратегій на різних організаційних рівнях.

*Бачення* – найбільш загальний орієнтир в діяльності підприємства, що дозволяє сформулювати гіпотетичну (ідеальну) форму його майбутнього існування і стану, який може бути досягнуто за найсприятливіших обставин.

*Цінності підприємства* – це фундаментальні переконання на яких базується бізнес підприємства.

*Цілі підприємства* – це майбутній бажаний стан, якого має досягнути підприємство в усіх сферах діяльності в кінці стратегічного періоду і на проміжних етапах.

*Діаграма бачення стратегії підприємства* використовується для візуального представлення стратегії підприємства.

*Стратегія підприємства* – це процес формування генерального перспективного напрямку розвитку підприємства на основі визначення якісно нових цілей, узгодження внутрішніх можливостей підприємства з умовами зовнішнього середовища та розробка комплексу заходів, які забезпечують їх досягнення.

На діаграмі бачення стратегії підприємства, як правило, відображають стратегічні ціннісні елементи та відповідні їм заходи розвитку, визначені прямо чи опосередковано.

### Приклад виконання

Необхідно розробити мотиваційну модель архітектури для компанії, що займається розробкою Web сайтів.

1. Створимо нову модель у середовище Archimate
2. Назвемо нашу схему(View) - Mission-Values-Vision-View. У властивостях виберемо Viewpoint – Motivation.
3. Визначимо основні цінності підприємства і зобразимо їх на діаграмі рис.1.

Основними цінностями компанії є:

- Лояльність клієнтів
- Якість створених програмних продуктів
- Відкритість
- Чесність та повага до персоналу

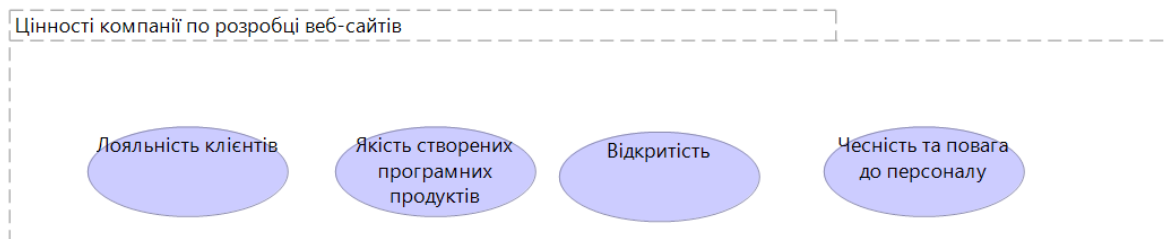


Рис. 1. Цінності компанії

4. Визначимо основну місію та цілі компанії. Місія компанії – Ми зробимо сайт Вашої мрії. Дерево цілей підприємства наведено на рис.2.

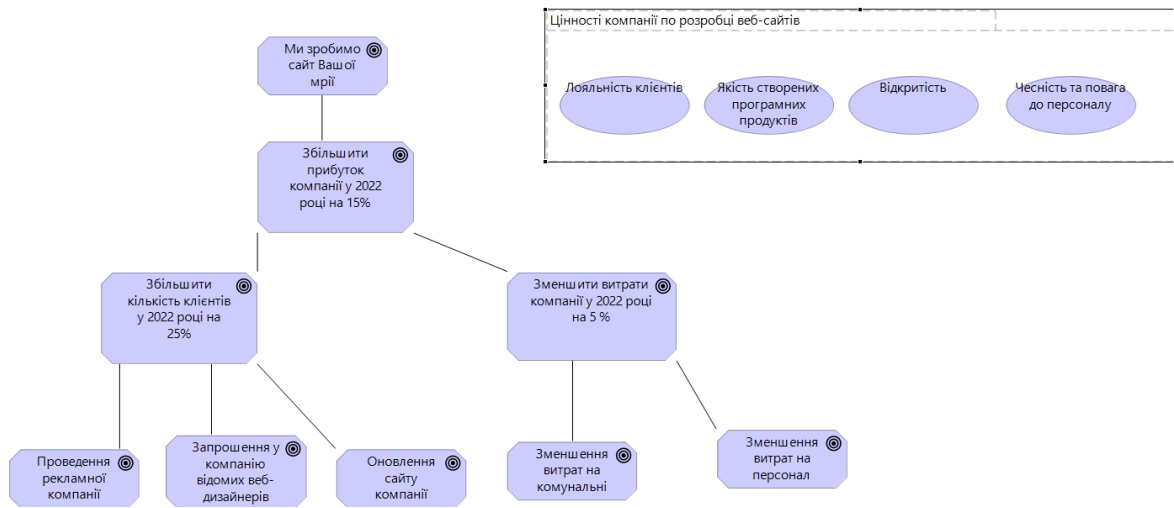


Рис. 2. Цінності та цілі компанії

5. Створимо нову схему(View) та назвемо її (Strategic-Value-Map-View). У властивостях виберемо Viewpoint – Motivation.

6. Поділимо стратегії на декілька шарів, а саме фінансові перспективи, цінності запропоновані клієнтам, внутрішні перспективи, перспективи розвитку, результат наведено на рис. 3.

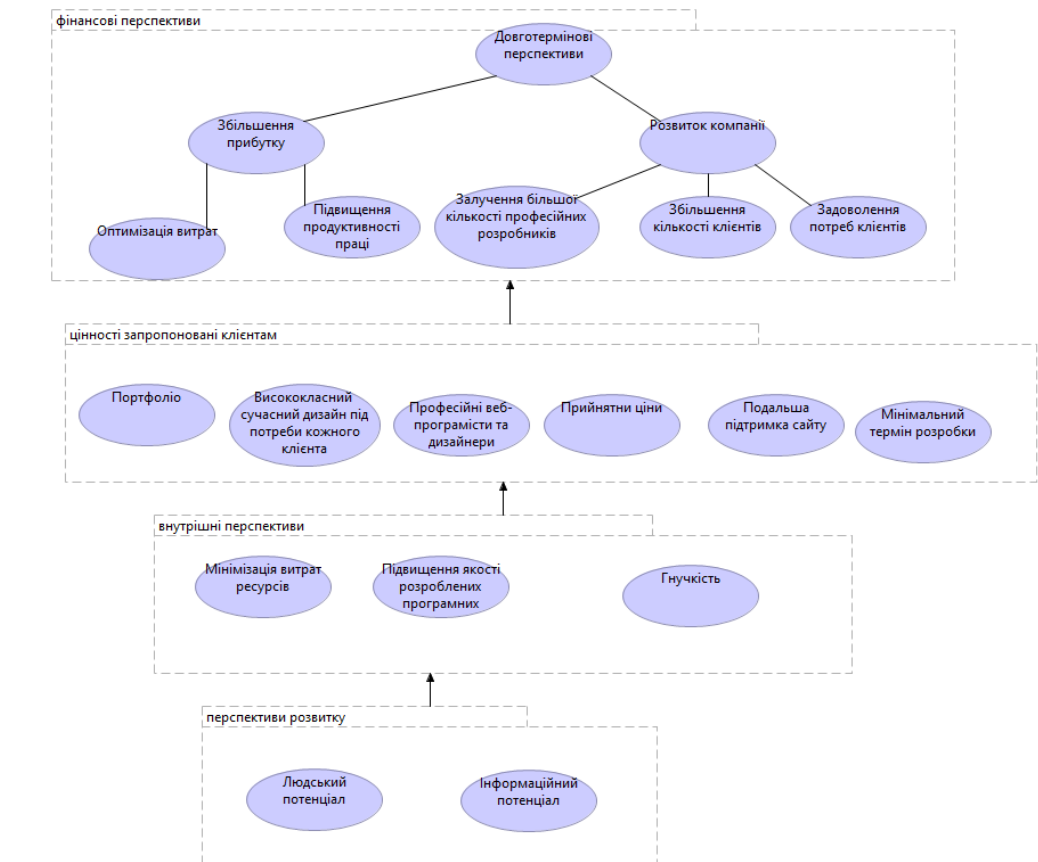


Рис. 3. Strategic-Value-Map-View



## Теми дослідження

Можна використовувати теми обрані в рамках дисципліни «Проектування ІС».

1. Дослідження роботи лабораторії харчового підприємства.
2. Дослідження функцій виробничого підрозділу підприємства.
3. Дослідження функцій логістики транспортного підрозділу підприємства.
4. Дослідження роботи відділу збуту готової продукції харчового підприємства.
5. Дослідження функцій контролю якості продукції на харчовому підприємстві.
6. Дослідження функцій адміністрування системи дистанційного надання освітніх послуг.
7. Дослідження алгоритмічної моделі для системи масового обслуговування підприємства.
8. Дослідження функцій управління запасами підприємства.
9. Дослідження роботи залізничного вузла.
10. Дослідження функцій електромонтажного підприємства.
11. Дослідження функцій туристичної фірми.
12. Дослідження функцій роботи підприємства по вирощуванню квітів.
13. Дослідження функцій аеропорту.
14. Дослідження функцій косметичного салону.
15. Дослідження функцій меблевого салону.
16. Дослідження функцій системи морських вантажних перевезень.
17. Дослідження функцій інформаційної системи з працевлаштування.
18. Дослідження функцій інформаційної системи надання кредитів.
19. Дослідження функцій інформаційної системи обліку кадрів підприємства.
20. Дослідження роботи фотостудії.
21. Дослідження функцій роботи букмекерської контори.
22. Дослідження функцій інформаційної системи аптеки.
23. Дослідження функцій організації руху таксі.
24. Дослідження функцій продажу квитків залізничного вокзалу.

25. Дослідження функцій інформаційної системи управління аптекою.
26. Дослідження функцій роботи автозаправного комплексу.
27. Дослідження функцій роботи майстерні з ремонту побутової техніки.
28. Дослідження функцій організації екскурсій.
29. Дослідження робіт по обліку обладнання організації та проведення інвентаризації.
30. Дослідження роботи шлюбної агенції.

## **Лабораторна робота №2**

### **Моделювання бізнес архітектури підприємства**

**Мета роботи:** навчитися будувати бізнес архітектуру підприємства у редакторі Archimate.

#### **Завдання на лабораторну роботу**

1. Побудувати організаційну структуру та визначити основні ролі.
2. Виокремити окремий підрозділ, який в подальшому буде досліджуватися.
3. Визначити бізнес-виконавців бізнес процесів, які виконують певні ролі.
4. Визначити основні бізнес процеси підприємства(не менше 5)
5. Побудувати бізнес архітектуру підприємства.

#### **Теоретичні відомості**

*Business actor* – відповідальний. В Archimate це елемент оргструктури: від однієї людини до всієї юрби холдингу, включаючи підрозділи, клієнтів та інших людей групами та поодиночі. Не плутати з конкретною людиною чи групою людей (архітектура "у типах", у ній немає конкретних об'єктів): це саме місце в оргструктурі. Називається іменником.



*Business role* – роль. Тип навмисно проміжний між "роботами" та "виконавцями", правильно розуміти як temporal part of actor (тимчасову

частину відповідального) під час занять відповідального якоюсь роботою. Один відповідальний може бути призначений на кілька ролей і кілька відповідальних можуть грати якусь роль, але тільки одна роль може бути призначена на якусь роботу.

*Business process* – процес (зокрема одна операція). Ім'я — дієслово в невизначеній формі, щоб краще розумівся послідовний ланцюжок операцій, кожна з яких виконується якоюсь роллю людей: "процес отримати страховий поліс — включає операції отримати запит, обробити заявку, прийняти платіж".

### *Зв'язки між об'єктами.*

Група відносин	Найменування відносини	Приклад графічного позначення в ArchiMate	опис відносини
структурні відносини	Реалізація (Realization)		Ставлення «реалізація» пов'язує логічну сутність з більш конкретною сутністю, яка її реалізує
	призначення (Assignment)		Ставлення «призначення» пов'язує об'єкти діяльності з активними елементами, які виконують цю поведінку (наприклад, з ролями і компонентами додатків)
	об'єднання (Aggregation)		Ставлення «об'єднання» показує, що сутність (елемент) утворює групу з ряду інших сутностей (елементів)
	Асоціація (Association)		Ставлення «асоціація» моделює відношення між об'єктами, яке не охоплюється іншими, більш визначеними відносинами
відносини залежності	вплив (Influence)		Ставлення «вплив» моделює ситуацію, в якій певний елемент має позитивний або негативний вплив на реалізацію іншого елемента
	доступ (Access)		Ставлення «доступ» моделює доступ елементів діяльності до об'єктів діяльності або об'єктів даних
	обслуговування (Serving)		Ставлення «обслуговування» моделює, що один елемент надає свою функціональність іншому
динамічні відносини	запуск (Triggering)		Ставлення «запуск» описує часові та причинні відносини між процесами, функціями і подіями
	Потік (Flow)		Ставлення «потік» описує обмін або передачу інформації між процесами і / або функціями

Група ставлення	Найменування відносини	Приклад графічного позначення в ArchiMate	опис відносини
інші відносини	спеціалізація (Specialization)		Ставлення «Спеціалізація» показує, що об'єкт є різновидом або конкретизацією іншого об'єкта
	з'єднання (Junction)		Ставлення «з'єднання» використовується для зв'язування відносин одного типу

### Приклад виконання

Необхідно розробити бізнес архітектуру підприємства для компанії, що займається розробкою Web сайтів.

1. Побудуємо організаційну структуру компанії (рис. 1).

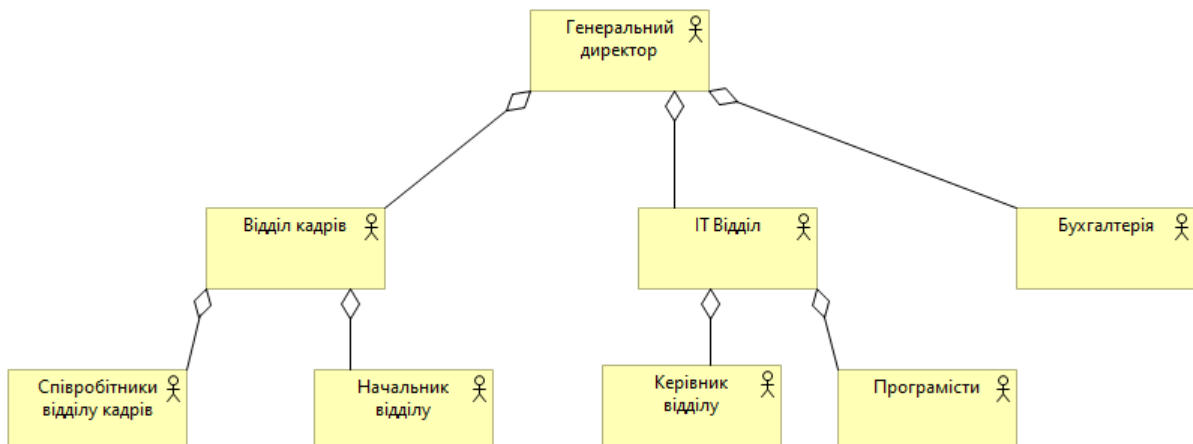


Рис. 1. Організаційна структура

2. Зосередимося на діяльності відділу кадрів.
3. Виокреми основні бізнес процеси у роботі відділу кадрів та бізнес ролі, тих хто їх виконує:

- Підбір кадрів:
  - аналіз професійних якостей – спеціаліст з IT;
  - аналіз психологічного портрету – Психолог.
- Робота з співробітниками - HR:
  - ведення списку співробітників;
  - табулювання співробітників;
  - формування графіку відпусток.
- Звільнення співробітників - HR

Додаємо ролі до організаційної діаграми (рис. 2), в якості психолога також буде виступати співробітник відділу кадрів.

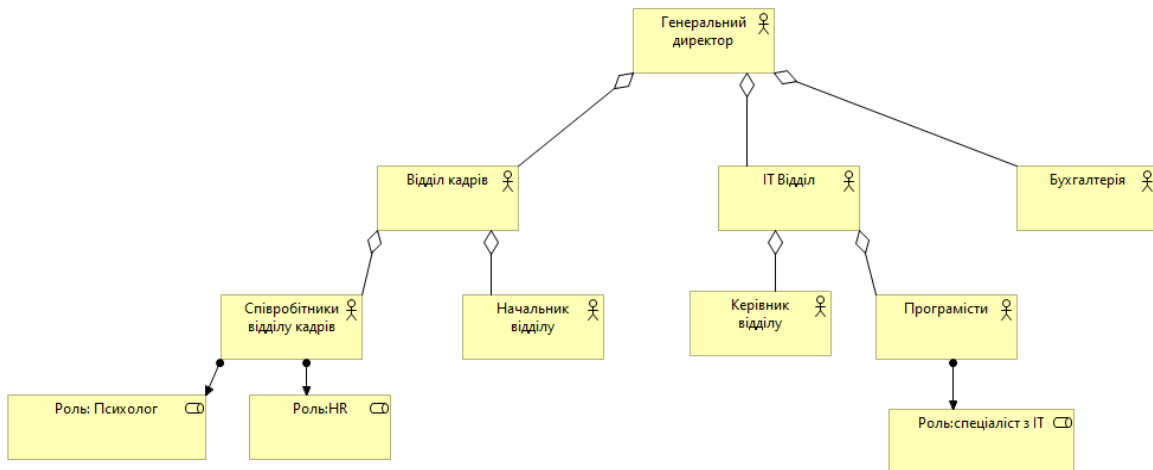


Рис. 2. Організаційна структура та ролі

#### 4. Побудуємо бізнес архітектуру відділу кадрів

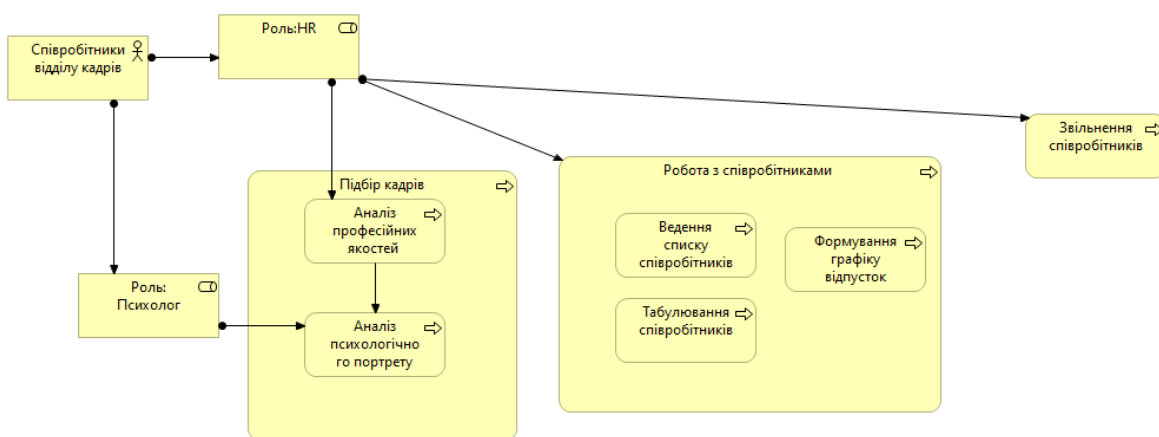


Рис. 3. Бізнес архітектура відділу кадрів

### Лабораторна робота №3

#### Моделювання системної архітектури підприємства

**Мета роботи:** навчитися будувати системну архітектуру підприємства у редакторі Archimate.

## **Завдання на лабораторну роботу**

1. Визначити основні додатки та побудувати їх карту.
2. Побудувати діаграму взаємодії прикладного програмного забезпечення підприємства.
3. Побудувати діаграму інтеграції прикладного програмного забезпечення підприємства лише для ІС, що розробляється
4. Побудувати діаграму представлення структури прикладної програми підприємства лише для ІС, що розробляється

## **Теоретичні відомості**

*Карта додатків* використовується для поділу додатків на групи, наприклад, на основі бізнес-одиниць.

*Діаграма бачення взаємодії додатків* використовується для відображення міграції потоків даних між додатками.

*Діаграма представлення структури додатку* використовується при розробці програмного додатку для розуміння структури додатку, його компонентів та пов'язаних з ними даних. Дану діаграму також можна використовувати для декомпозиції системи: представлення складових частин системи / компонентів, сервісів додатків (або інтерфейсів додатків).

*Діаграма бачення архітектури додатку* призначена для відображення як додатків, так і модулів додатків.

*Діаграма бачення технологічного забезпечення* прикладного рівня архітектури підприємства призначена для відображення способів підтримки прикладного рівня архітектури підприємства за допомогою програмного та апаратного забезпечення, зокрема, фізичними пристроями, мережевим або системним програмним забезпеченням.

## **Приклад виконання**

Необхідно розробити системну архітектуру підприємства для компанії, що займається розробкою Web сайтів.

1. Визначимо основні додатки та побудуємо їх карту (рис.1).

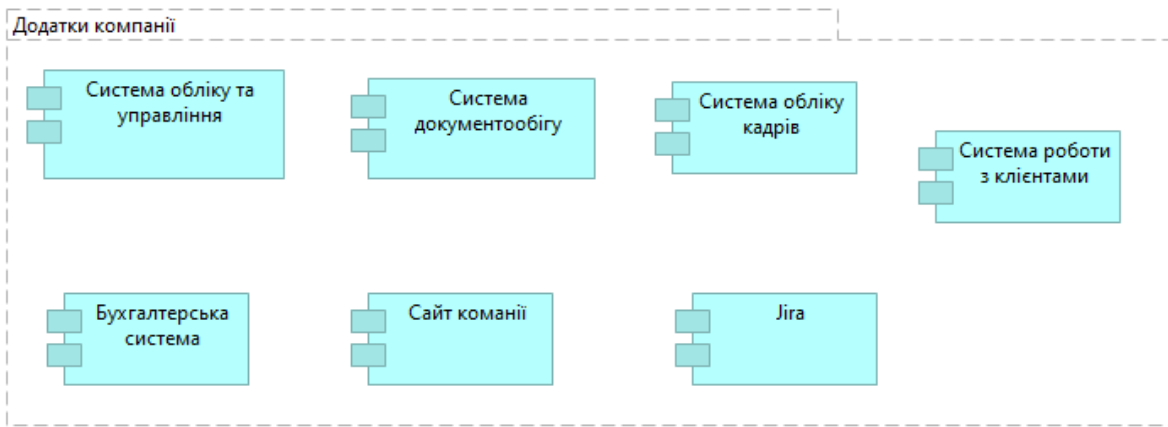


Рис. 1. Карта додатків

2. Розглянемо їх взаємодію (рис. 2).

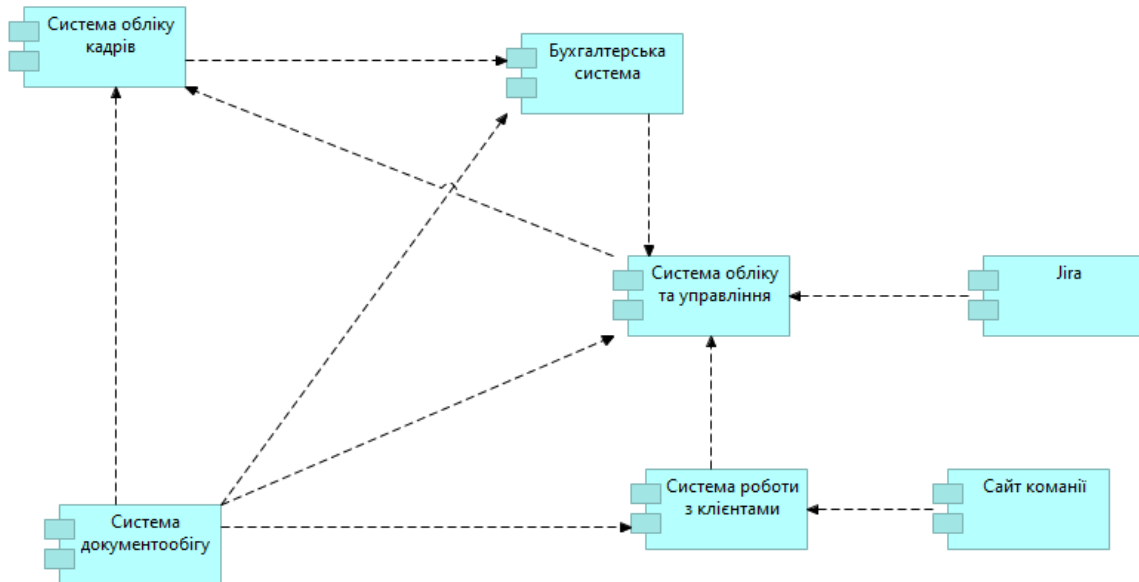


Рис. 2. Взаємодія додатків

3. Розглянемо інтеграцію ІС відділу кадрів з іншими додатками.

Система обліку кадрів взаємодії з двома підсистемами:

- система обліку кадрів реалізує сервіс «Документування діяльності співробітників», які використовується Бухгалтерською системою;
- система обліку кадрів формує документ «Штатний розклад», який передається до системи документообігу.

Відповідна діаграма зображена на рис.3.

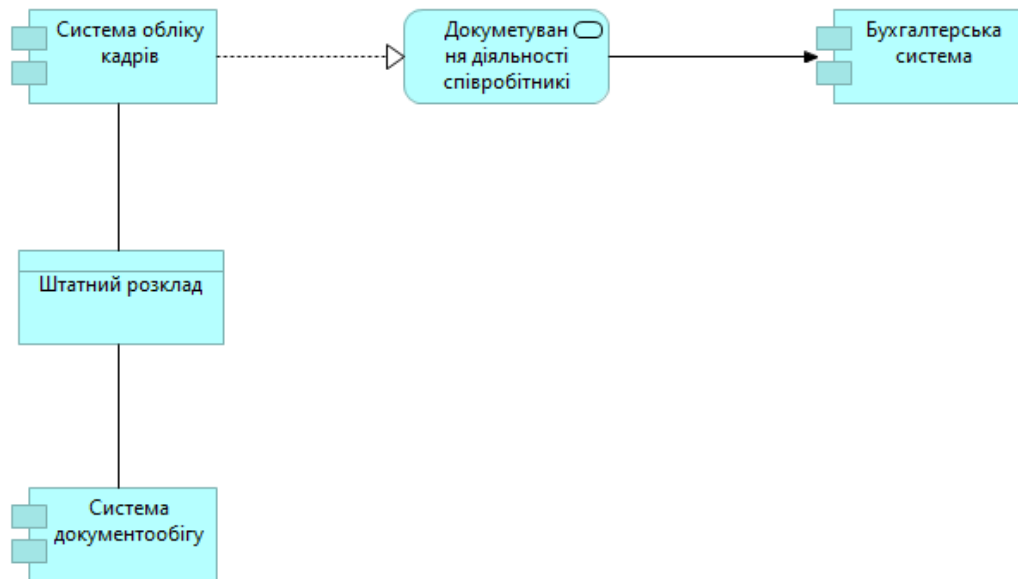


Рис. 3. Діаграма інтеграції прикладного програмного забезпечення

4. Розробимо структуру системи обліку кадрів, вона буде складатися з наступних модулів:

- модуль ведення списку співробітників;
- модуль табулювання співробітників;
- модуль обліку штатних одиниць;
- модуль формування штатного розкладу.

Відповідна діаграма зображена на рис.4.

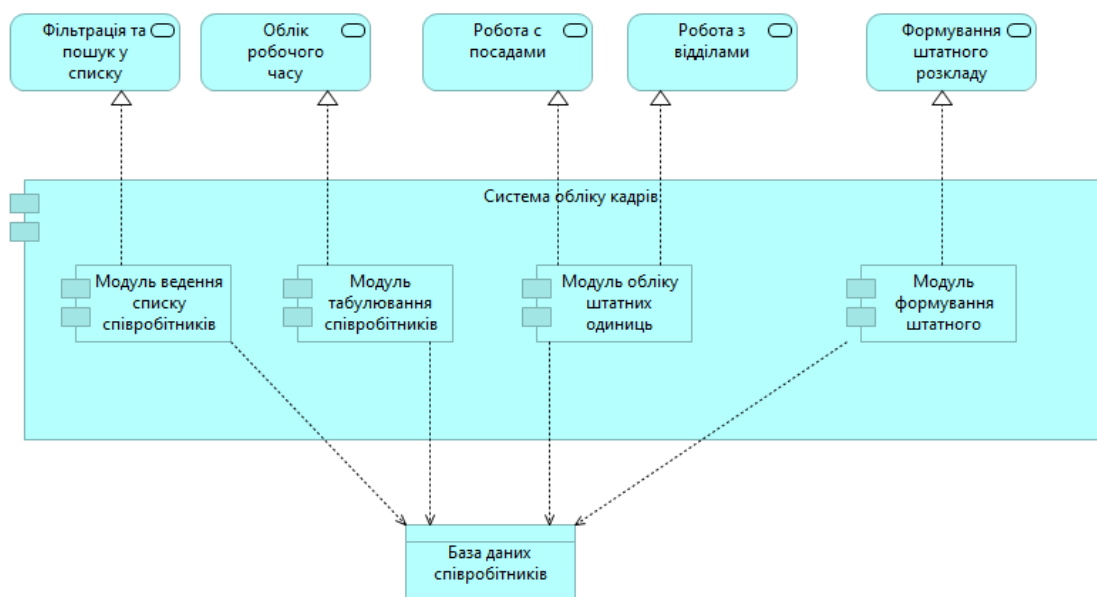


Рис. 4. Діаграма представлення структури прикладної програми підприємства



## Лабораторна робота № 4

### Моделювання технічної архітектури підприємства

**Мета роботи:** навчитися будувати технічну архітектуру підприємства у редакторі Archimate.

#### Завдання на лабораторну роботу

На основі діаграм, що побудовані у попередній лабораторній роботі побудувати :

1. Діаграма бачення інфраструктури підприємства для ІС, що розробляється;
2. Розробити діаграму бачення технологічного забезпечення прикладного рівня архітектури підприємства для ІС, що розробляється.

#### Теоретичні відомості

Програмний засіб ArchiMate 3.0 підтримує розробку таких типів діаграм технологічної моделі архітектури підприємства (рис. 6.1):

- діаграма бачення технічного обслуговування;
- діаграма бачення платформи;
- діаграма бачення технологічної карти;
- діаграма бачення інфраструктури.

#### Приклад виконання

Базуючись на діаграмі представлення структури ІС, що розроблена у лабораторній роботі №3, побудуємо діаграми бачення інфраструктури підприємства (рис. 1) та технологічного забезпечення прикладного рівня архітектури підприємства (рис. 2).

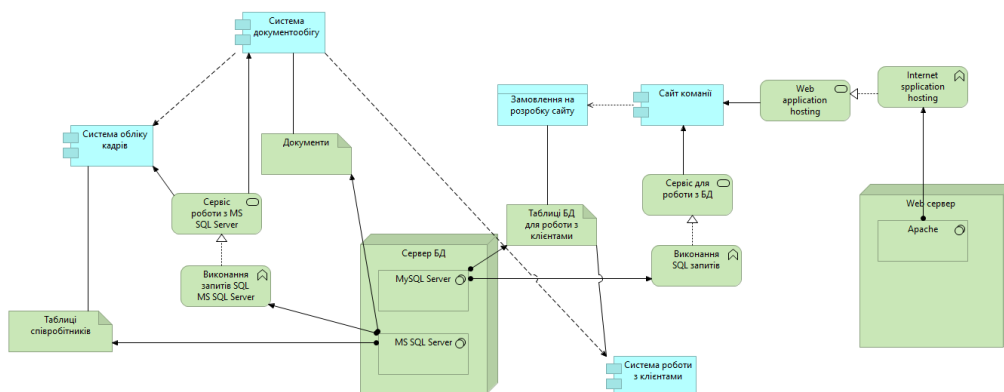


Рис. 1. Діаграма бачення інфраструктури підприємства

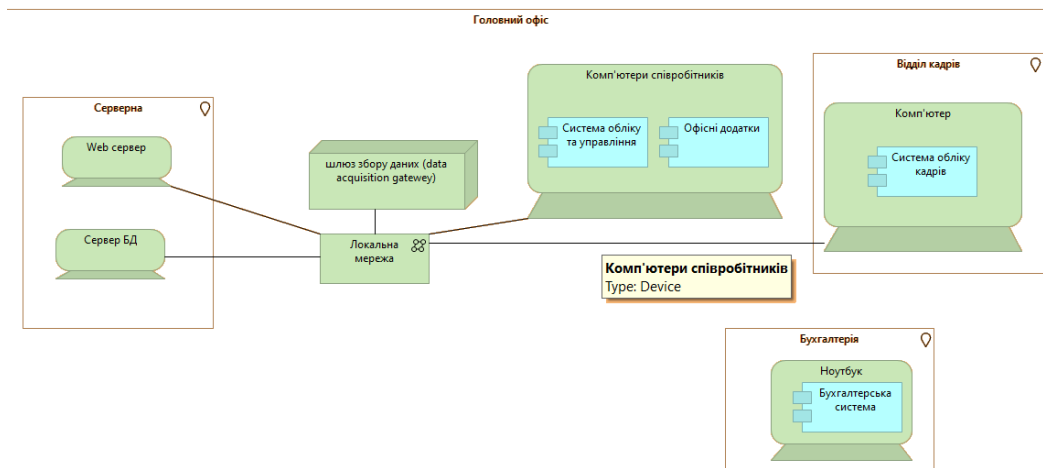


Рис. 2. Діаграма бачення технологічного забезпечення прикладного рівня архітектури підприємства

## Лабораторна робота № 5 Розробка концептуальної та логічної схем БД

**Мета роботи:** Закріпити навички розробки схем БД.

### Завдання на лабораторну роботу

На базі попередніх необхідно:

1. Виділити основні сутності та описати їх. Не менше п'яти сутностей.
2. Розробити концептуальну модель БД.
3. Розробити логічну схему БД.

### Приклад виконання

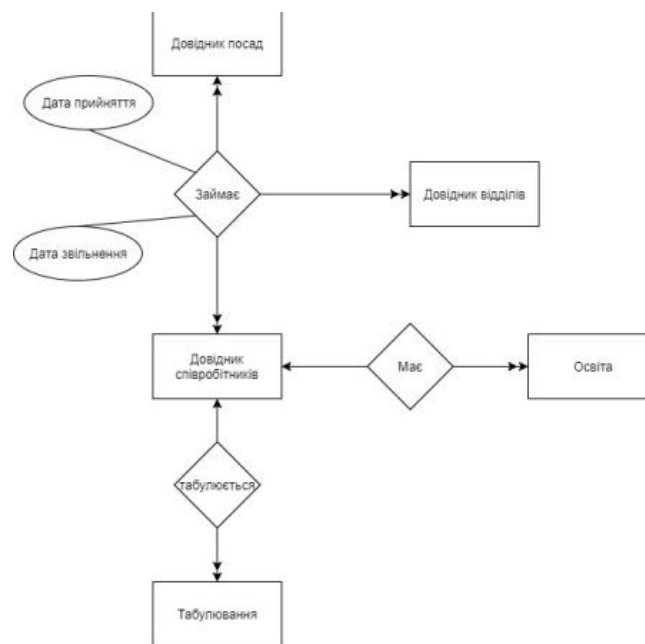
Опишемо основні сутності ІС відділу кадрів:

1. Довідник посад. Вміщує перелік посад підприємства. Основні атрибути: код посади, назва посади, оклад.
2. Довідник відділів. Вміщує інформацію про відділи, що існують на підприємстві. Основні атрибути: код відділу, назва відділу.
3. Довідник співробітників. Вміщує перелік співробітників підприємства. Основні атрибути: табельний номер, ПІБ, стать, дата народження, стаж, ідентифікаційний код, паспорт, місце проживання,

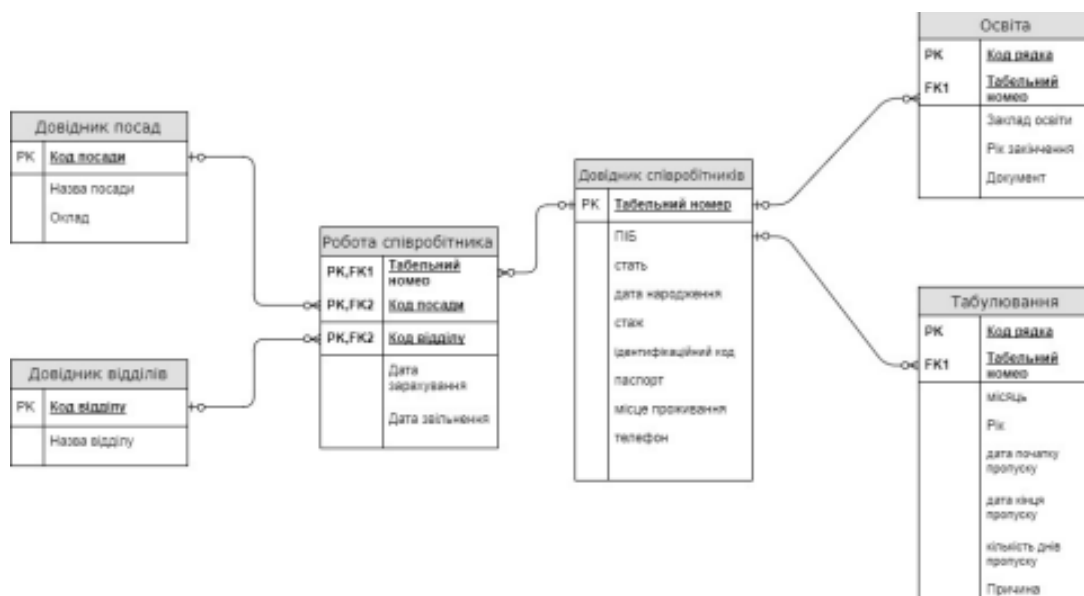
телефон.

4. Освіта. Вміщує інформацію про освіту конкретного співробітника. Має атрибути: табельний номер, що закінчив, коли закінчив, номер документу.

5. Табулювання. Вміщує інформацію про наявність співробітника на робочому місці. Основні атрибути: табельний номер, місяць, рік, дата початку пропуску, дата кінця пропуску, кількість днів пропуску, причина. Концептуальна модель буде мати наступний вигляд.



Розробимо логічну модель БД. При цьому необхідно пам'ятати, що зв'язок багато до багатьох перетворюється у окрему таблицю.



## Лабораторна робота № 6

### Розробка фізичної схеми БД у клієнт-серверній СУБД

**Мета роботи:** Закріпити навички розробки схем БД.

#### Завдання на лабораторну роботу

Використовуючи лабораторну роботу №5 розробити фізичну модель БД, включаючи таблицю опису фізичної моделі.

#### Теоретичні відомості

Приведемо короткий огляд типів даних MySQL.

Для зберігання символічної інформації використовуються символічні типи **Текстові типи** (табл. 1).

*Таблиця 1*

Тип	Опис
CHAR()	Рядок тексту сталої довжини, заданої у дужках, до 255 символів
VARCHAR()	Рядок тексту змінної довжини до заданої у дужках, до 255 символів
TEXT	Рядок тексту до 65 535 символів
BLOB	Двійковий об'єкт до 65 535 байт даних
MEDIUMTEXT	Рядок тексту до 16 777 215 символів
MEDIUMBLOB	Двійковий об'єкт до 16 Мегабайт даних
LONGTEXT	Рядок тексту до 4 294 967 295 символів
LOBLOB	Двійковий об'єкт до 4 Гігабайт даних
ENUM(x,y,...)	Список можливих значень, до 65 535 різних. Якщо значення, що вставляють у поле, не перелічене у списку, буде вставлено порожнє значення. Упорядкування — у порядку запису
SET(x,y,...)	Подібно до ENUM, але може містити до 64 значень у списку. Комірка може містити довільну підмножину множини перелічених значень

## Числові типи

Для роботи з невід'ємними цілими числами потрібно вписати службове слово UNSIGNED після назви типу (табл. 2).

Таблиця 2

Тип	Опис
TINYINT	Ціле від -128 до 127 або від 0 до 255
SMALLINT	Ціле від -32 768 до 32 767 або від 0 до 65 535
MEDIUMINT	Ціле від -8 388 608 до 8 388 607 або від 0 до 16 777 215
INT	Ціле від -2 147 483 648 до 2 147 483 647 або від 0 до 4 294 967 295
BIGINT	Ціле -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 або від 0 до 18 446 744 073 709 551 615
FLOAT(n,k)	Число з рухомою крапкою (4 байти) з додатним модулем від $1.2 \cdot 10^{-39}$ до $3.4 \cdot 10^{38}$
DOUBLE(n,k)	Число з рухомою крапкою подвійної точності (8 байт) з додатним модулем від $2.2 \cdot 10^{-308}$ до $1.8 \cdot 10^{308}$
DECIMAL(n,k), NUMERIC(n,k) або DEC(n,k)	Число з рухомою крапкою, збережене як рядок, (M + 2) байти

Максимальну кількість цифр чисел з рухомою крапкою вказують як параметр *n*. Максимальну кількість цифр після десяткової крапки вказують як параметр *k*.

Тип BOOL — синонім до TINYINT, а сталі TRUE та FALSE — до 1 та 0 відповідно. **Типи дати й часу** (табл. 3)

Таблиця 3

Тип	Опис
DATE	Дата у форматі YYYY-MM-DD (3 байти). Підтримано діапазон від '1000-01-01' до '9999-12-31'
DATETIME	Формат YYYY-MM-DD HH:MM:SS (8 байт). Підтримано діапазон від '1000-01-01 00:00:00' до '9999-12-31 23:59:59'

Продовження таблиці 3

TIMESTAMP	Кількість секунд з початку епохи Unix у форматі YYYY-MM-DD HH:MM:SS (4 байти). Підтримано діапазон від '1970-01-01 00:00:01' до '2038-01-09 03:14:07'
TIME	Час у форматі HH:MM:SS (3 байти). Підтримано діапазон від '-838:59:59' до '838:59:59'
YEAR(M)	Рік у M-цифровому форматі (M = 2, 4). Значення, дозволені в 4-цифровому форматі: від 1901 до 2155. Значення дозволені у 2-цифровому форматі: від 70 до 69, що відповідає 1970 та 2069.

Навіть якщо DATETIME та TIMESTAMP повертають однакові формати, вони працюють дуже по різному. В запиті INSERT або UPDATE формат TIMESTAMP передбачає автоматичне встановлення поточного часу й дати. Також TIMESTAMP приймає різні формати: YYYYMMDDHHMMSS, YMMDDHHMMSS, YYYYMMDD і YMMDD.

### Приклад виконання

Опис основних таблиць фізичної моделі наведено у табл. 4.

Таблиця 4

Назва поля в логічній моделі	Назва поля в фізичній моделі	Тип	Опис
Довідник посад – Dovid_posad			
код посади	Id_posad	INT	Первинний ключ
назва посади	Naz_posad	VARCHAR(25)	
оклад	Oklad	DECIMAL	
Довідник відділів – Dovid_viddil			
код відділу	Id_viddil	INT	Первинний ключ
назва відділу	Naz_viddil	VARCHAR(25)	
Довідник співробітників – Dovid_spiv			
Табельний номер	Tab_nom	INT	Первинний ключ

Продовження таблиці 4

ПІБ	PIB	VARCHAR(25)	
Стать	Stat	VARCHAR(7)	Може приймати лише два значення «жінка» або «чоловік»
Дата народження	Date_nar	DATE	Дата народження повинна бути менша за поточну дату на 16 років та більша ніж поточна дата – 80 років
Стаж	Stag	INT	Ціле позитивне число менше за 65.
Ідентифікаційний код	ID_kod	BIGINT	Значення більше за нуль складається з десяти цифр
Адреса проживання	Adress	VARCHAR(100)	
Паспорт	Passport	VARCHAR(100)	Включає серію та номер паспорта та ким та коли виданий
Телефон	Telephone	VARCHAR(20)	У форматі +38(..)...-...-..
Освіта – Osvita			
Номер рядка	Id_osvita	INT	Первинний ключ
Табельний номер	Tab_nom	INT	Вторинний ключ до таблиці Dovid_spiv
Заклад освіти	Zaklad	VARCHAR(50)	
Рік закінчення	Rik_zak	DATE	
Документ	Documents	VARCHAR(25)	
Таблювання – Tabul			
Номер рядка	Id_tab	INT	Первинний ключ

Продовження таблиці 4

Табельний номер	Tab_nom	INT	Вторинний ключ до таблиці Dovid_spiv
Місяць	Mis	INT	Може приймати значення від 1 до 12
Рік	Rik	DATE	
Дата початку пропуску	Date_nac	DATE	
Дата кінця пропуску	Date_kin	DATE	
Кількість днів пропуску	Kil_dniv	INT	Ціле позитивне число, максимальне значення 31
Причина	Pricina	VARCHAR(100)	
Робота співробітника – Rab_spiv			
Номер рядка	Id_rab	INT	Первинний ключ
Табельний номер	Tab_nom	INT	Вторинний ключ до таблиці Dovid_spiv
код посади	Id_posad	INT	Вторинний ключ до таблиці Dovid_posad
код відділу	Id_viddil	INT	Вторинний ключ до таблиці Dovid_viddil
Дата зарахування	Date_zarach	DATE	
Дата звільнення	Date_zvil	DATE	



## Лабораторна робота № 7

### Розробка БД та створення таблиць у клієнт-серверній СУБД

**Мета роботи:** навчитися розробляти БД та створювати таблиці за допомогою клієнт-серверної СУБД

#### Завдання на лабораторну роботу

Використовуючи розроблену у попередньої лабораторної роботі модель фізичну БД створити таблиці БД у клієнт-серверній СУБД, наприклад, MySQL. Таблиці зв'язати між собою зовнішніми ключами відповідно до розробленої моделі.

#### Теоретичні відомості

##### *Приклад виконання (MySQL)*

1. Спочатку встановимо XAMPP для розгортання локального веб серверу. Можна скористатися будь яким іншим аналогічним пакетом. Скачати його можна за посиланням <https://sourceforge.net/projects/xampp/>

2. При встановленні обов'язково встановити phpMyAdmin ну і звісно сам MySQL. Мінімальні налаштування виглядають наступним чином (рис. 1)

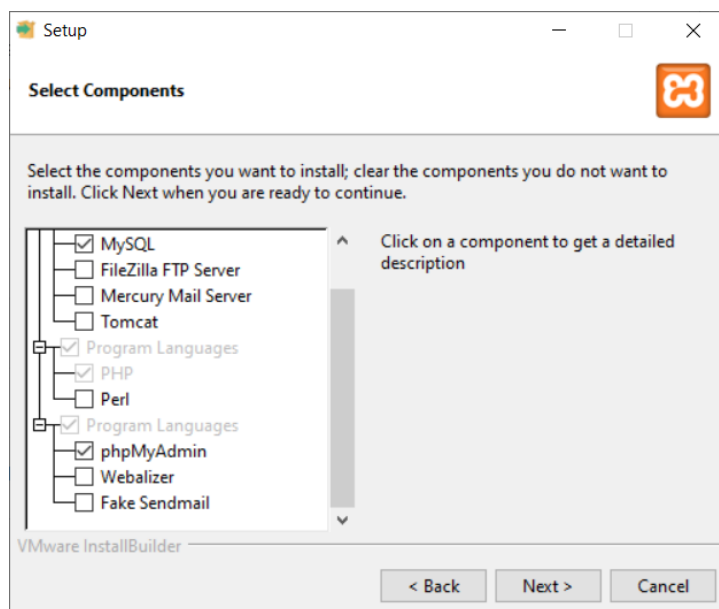


Рис. 1. Встановлення XAMPP

3. Відкриємо панель XAMP у режимі адміністратора та запустимо сервер та MySQL (рис. 2).

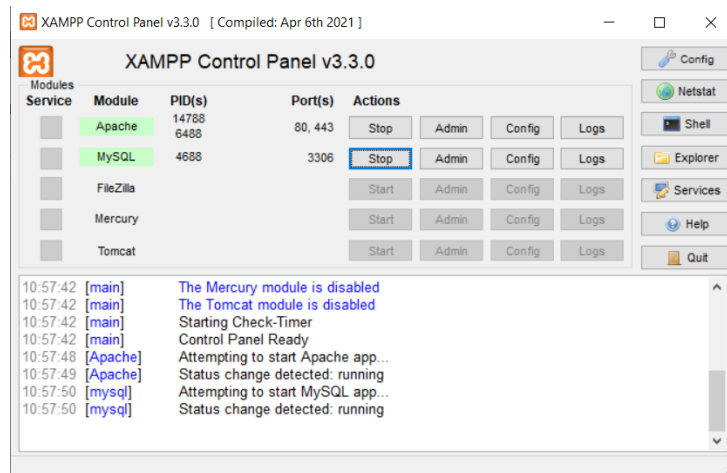


Рис. 2. Панель ХАМРР

4. Відкриємо phpMyAdmin натиснувши на кнопку Admin у панелі та створимо нову БД MySQL з ім'ям Kadri на вкладці Бази даних.

5. Перейдемо до розробки таблиць. Спочатку створимо таблиці довідники і лише після цього перейдемо до розробки дочірніх таблиць із зовнішніми ключами.

6. Для створення таблиці оберемо розроблену БД та вкажимо ім'я таблиці та кількість стовпців (рис. 3).

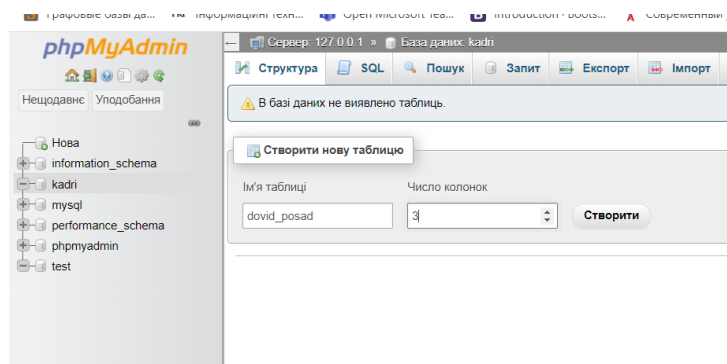


Рис. 3. Створення таблиці

7. Додаємо до таблиці стовпці (рис. 4) та натискаємо кнопку Зберегти.

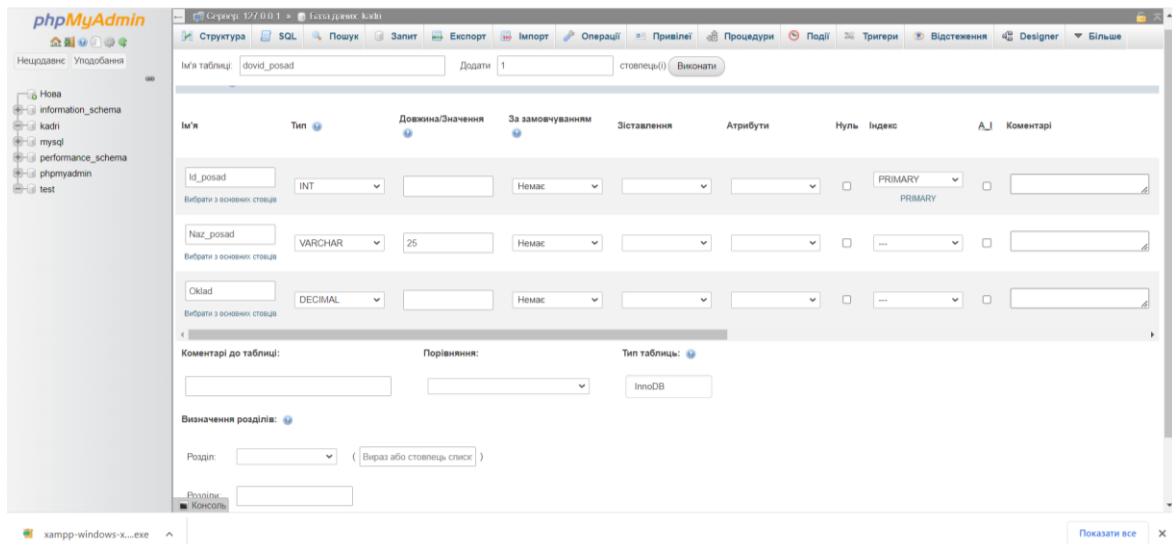


Рис. 4. Додавання стовпців

8. Аналогічно створюємо інші таблиці без зовнішніх ключів.

9. Створимо таблицю з зовнішнім ключем, спочатку аналогічно попереднім прикладам створюємо таблицю, наприклад Довідник освіти співробітника, потім переходимо на вкладку Вид відносин та встановлюємо обмеження зовнішнього ключа (рис. 5) та натискаємо кнопку Зберегти.

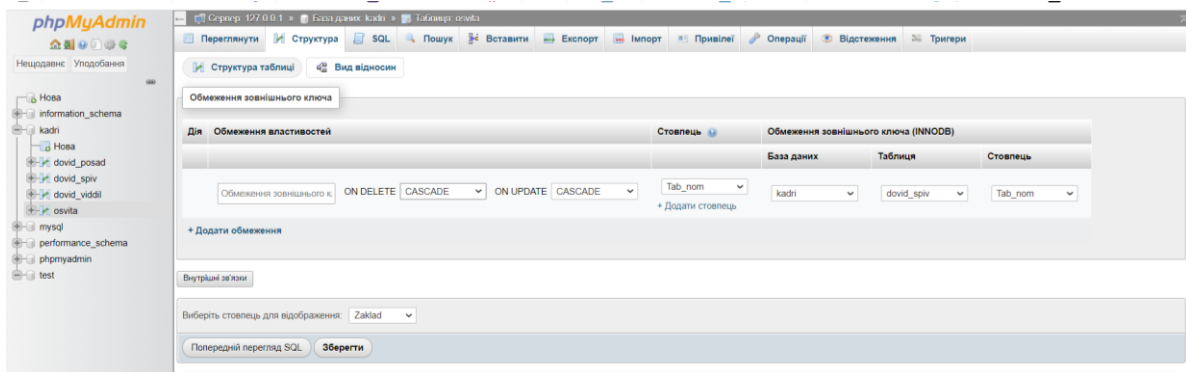
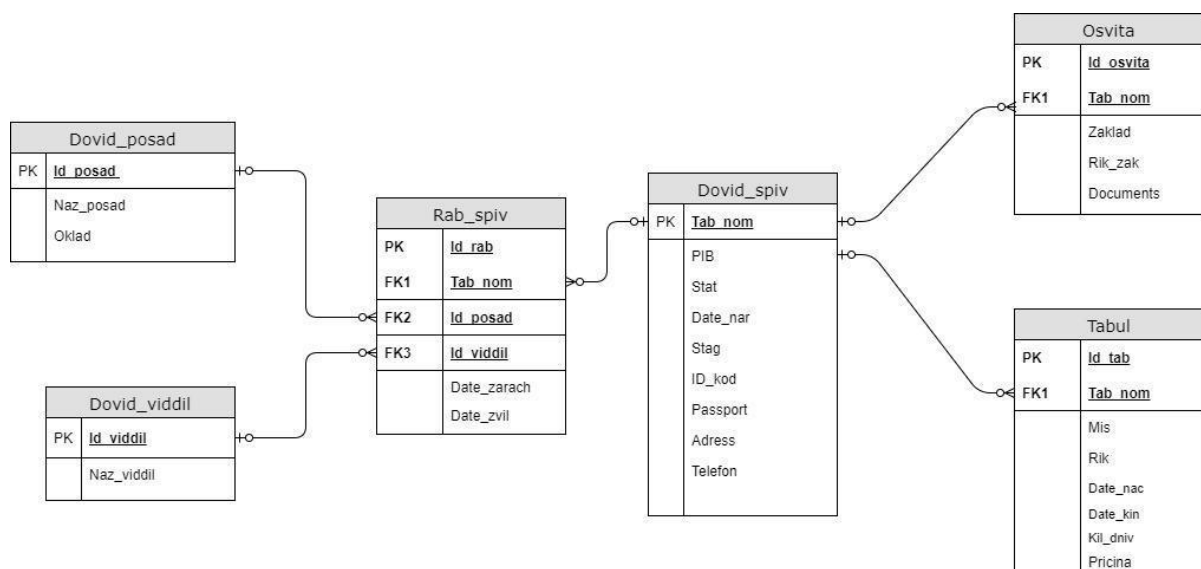


Рис. 5. Додавання зовнішніх ключів

10. Аналогічно створимо усі інші таблиці.



## Лабораторна робота № 8

### Створення та тестування серверного додатку

**Мета роботи:** Навчитися створювати застосунки на node.js для роботи з таблицями БД.

#### Завдання на лабораторну роботу

Використовуючи розроблені у попередньої лабораторної роботі БД MySQL, розробити частину серверного застосунку для роботи з декількома (як мінімум з 3) таблицями БД, а саме для виконання операцій CRUD (створення, читання, зміни та видалення даних). Протестувати роботу застосунку з використанням Postman або інших засобів тестування серверних БД.

#### Теоретичні відомості

##### *Express (<https://expressjs.com/>)*

*Express* — це мінімальний і гнучкий фреймворк веб-додатків для Node.js, який забезпечує надійний набір функцій для веб- і мобільних додатків.

Завдяки безлічі методів по роботі з HTTP та наявності проміжного програмного забезпечення фреймворк надає можливість створення надійного API швидко й легко.

*Express.js* – це одна з найпоширеніших JavaScript-платформ для бекенд-розробки. Фактично він є open source фреймворком Node.js, і використовується для створення сайтів та веб-застосунків, але найпоширеніша область його застосування - створення Restful API. Express надає безліч готових функцій та механізмів, які значно спрощують процес розробки та роблять її у рази швидше.

### ***Postman (<https://www.postman.com/>)***

Postman - клієнт HTTP для тестування API. Клієнти HTTP перевіряють надсилання запитів від клієнта на сервер та отримують відповідь від сервера.

API (Application Programming Interface) - це інтерфейс обміну даними з сервера між двома програмами або компонентами програмного забезпечення. Postman допомагає розробити дизайн API та створити mock сервери (імітатори додатків). Наприклад, за допомогою Postman можна перевірити, як API реєструє нового користувача програми, як він додає та видаляє дані про нього на сервері.

Використовуючи Postman, можна:

- складати та надсилати HTTP - запити в API;
- створити колекції (набір послідовних запитів) та запитів для скорочення часу тестування;
- змінити параметри запитів (наприклад, ключі авторизації та URL);
- змінити середовище запитів (наприклад, на тестовому стенді, локально або на сервері);
- додати контрольні точки при виклику API (фіксація передачі даних);
- проводити автоматизоване тестування API шляхом збору запитів за допомогою колекційного бігуна.

Для роботи з серверами програма використовує протокол HTTP. Тестер надсилає тестові запити від клієнта на сервер і отримує відповідь, якщо в API є помилка.

Postman доступний у формі [програми](#) Windows, Linux та MacOS, а також у веб - інтерфейсі (вам потрібно встановити програму [агента](#) Desktop Postman Desktop).

## Приклад виконання

1. Ініціалізуємо новий проєкт за допомогою команди `npm init -y`. Після ініціалізації у корні з'являється новий файл `package.json` у якому наведено опис проєкту.
2. Встановимо фреймворк Express для роботи з серверними застосунками (<https://expressjs.com/> `npm install express`)
3. Опишемо у файлі `package.json` у вкладці `script` механізм запуску проєкта через `node.js` (рис. 1).



```
{
  "name": "lab8_2023",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

Рис. 1.

Файл `package.json` з налаштуваннями запуску проєкту

4. Створимо файл `index.js` та напишемо в ньому команду для виведення у консоль, далі запусимо проєкт за допомогою команди `npm start`, та побачимо результат виведення у консоль.

5. Вкажемо у файлі `package.json`, те що для розробки застосунку будемо використовувати `require` для з'єднання модулів для цього додамо поле `type: "type": "commonjs"`,

А у файлі `index.js` імпортуємо `express`:

```
// Підключаємо express
const express = require('express');
```

6. Створимо новий сервер, який буде прослуховувати 5000 порт:

```

JS index.js > ...
1
2  const express = require('express');
3  const bodyParser = require('body-parser');
4  // Встановлюємо порт
5  const PORT = 5000;
6  const app = express();
7  // створюємо екземпляр застосунку
8  //вхід
9  //порт
10 //callback функція, яка відпрацює лише у випадку успішного запуску серверу
11 app.listen(PORT, () => console.log("SERVER START!!!"))

```

7. При запуску серверу у браузері за допомогою адресного рядка <http://localhost:5000/> можна побачити помилку обробки GET запиту. Напишемо перший endpoint для обробки GET запиту за адресою ‘/’

```

JS index.js > ...
1
2  const express = require('express');
3  const bodyParser = require('body-parser');
4  // Встановлюємо порт
5  const PORT = 5000;
6  const app = express();
7  // Обробка get запиту - endpoint
8  // Вхід
9  // Адреса за якою він буде працювати
10 // Функція, що буде виконуватися по запиту на цей endpoint
11 // Параметри функції
12 // req - запит
13 // res - відповідь
14 app.get('/', (req, res) => {
15     // вказуємо 200 - успішний статус
16     // json - тіло відповіді - повідомлення
17     res.status(200).json("Сервер працює123");
18 })
19 // створюємо екземпляр застосунку
20 //вхід
21 //порт
22 //callback функція, яка відпрацює лише у випадку успішного запуску серверу
23 app.listen(PORT, () => console.log("SERVER START!!!"))

```

8. Після зупинки та нового запуску побачимо у браузері повідомлення Сервер працює.

9. Встановимо залежність nodemon для того, щоб оновлювати сервер без перезапуску *npm i --D nodemon*

10. Внесемо зміни до файлу package.json додавши новий скрипт *"dev": "nodemon index.js"*

Запуск скрипта буде відбуватися за допомогою команди *npm run dev*

11. Встановимо додатковий пакет Body Parser для аналізу тіла вхідних запитів. *npm install body-parser --save*

Та додамо цю бібліотеку у файл index.js

```

// Підключаємо bodyParser
const bodyParser = require('body-parser');

```

12. Далі необхідно встановити MySQL *npm install mysql --save*

13. Змінемо index.js додавши до нього endpoint для парсингу різних типів запитів

```

JS index.js > ...
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  // Встановлюємо порт
4  const PORT = 5000;
5  const app = express();
6  // парсити запити типу content-type - application/x-www-form-urlencoded
7  app.use(bodyParser.urlencoded({ extended: true }));
8  // парсити запити типу content-type - application/json
9  app.use(bodyParser.json());
10 // Шаблони сторінок у форматі hbs
11 app.set("view engine", "hbs");
12 // Обробка get запиту - endpoint
13 // Вхід
14 // Адреса за якою він буде працювати
15 // Функція, що буде виконуватися по запиту на цій endpoint
16 // Параметри функції
17 // req - запит
18 // res - відповідь
19 app.get('/', (req, res) => {
20   // Вказуємо 200 - успішний статус
21   // json - тіло відповіді - повідомлення
22   res.status(200).json("Сервер працює123");
23 });
24 // створюємо екземпляр застосунку
25 //вхід
26 //порт
27 //callback функція, яка відпрацює лише у випадку успішного запуску серверу
28 app.listen(PORT, () => console.log("SERVER START!!!"));
29 // Require employee routes
30 const departmentRoutes = require('./router/department.routes')
31 app.use('/api/department', departmentRoutes);

```

14. Створимо папку config а вній файл для конфігурації з'єднання з БД config\config.bd.js. Припишемо код з'єднання з БД, яку створили у минулій лабораторній

```

config > JS config.bd.js > ...
1  // import mysql from 'mysql'
2  const mysql = require('mysql');
3  // Створюємо нове з'єднання на локальному хості
4  // з вказаними параметрами
5  var connection = mysql.createConnection({
6    host: 'localhost',
7    user: 'root',
8    password: '',
9    database: 'kadri'
10 });
11 // З'єднуємося з БД
12 // Якщо вдало - виводимо, що з'єднання відбулося
13 // Якщо ні виводимо помилку
14 connection.connect(function (err) {
15   if (!err) {
16     console.log("Database is connected");
17   } else {
18     console.log("Error while connecting with database");
19   }
20 });
21
22 // Експорт з'єднання
23 module.exports = connection;
24

```

15. Підключимо даний файл до основного файлу index. Js

*var connection = require ('./../config/config.bd');*

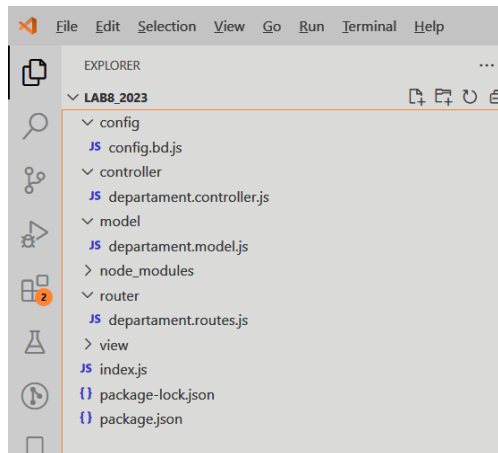
16. Запустимо XAMPP та MySQL а потім сервер Node.js якщо з'єднання з БД вдалося то у терміналі відобразиться відповідне повідомлення.



17. Для подальшого створення додатку будемо використовувати багатопарову архітектуру, а саме:

- шар model – для взаємодії з БД
- шар controller для взаємодії з моделлю
- шар router для маршрутизації запитів відповідно до REST

Створимо окремі папки для котролерів, моделі та роумінгу. Проєкт буде мати наступну структуру:



18. Спочатку розробимо модель. Сама вона буде з'єднуватися з БД та отримувати з неї дані та відправляти їх до БД. Створимо новий файл моделі для взаємодії з таблицею відділів у БД model\departament.model.js. Оскільки вся взаємодія з БД буде відбуватися у шарі з моделлю перенесемо підключення до БД до файлу моделі з файлу index.js.

19. Підключимо БД до файлу model\departament.model.js., опишемо функцію створення окремого об'єкту відділу та його методи для всіх операцій з БД вставлення, пошуку за id, виведення всіх записів з таблиці, зміни певного запису та видалення певного запису.

```
model > JS departament.model.js > ...
1 // Підключення до БД
2 var connection = require('../config/config.bd');
3 // Функція для створення об'єкту Department
4 var Department = function (departament) {
5     this.id_viddil = departament.Id_viddil;
6     this.naz_viddil = departament.Naz_viddil;
7 }
8 // Створення нового запису у БД
9 // newDep - об'єкт департамент зі значеннями які створюються
10 // result - результат створення
11 Department.create = function (newDep, result) {
12     connection.query("INSERT INTO dovid_viddil set ?", newDep, function (err, res) {
13         if (err) {
14             console.log("error: ", err);
15             result(err, null);
16         }
17         else {
18             console.log(res.insertId);
19             result(null, res.insertId);
20         }
21     });
22 };
```

```

23 // Пошук у таблиці за id
24 //id - значення id відділу
25 // result - результат запиту з пошуку
26 Departament.findById = function (id, result) {
27     connection.query("Select * from dovid_viddil where id_viddil = ? ", id,
28         function (err, res) {
29             if (err) {
30                 console.log("error: ", err);
31                 result(err, null);
32             }
33             else {
34                 result(null, res);
35             }
36         });
37 };
38 // Виведення усіх департаментів, що є у таблиці
39 // result - результат запиту
40 Departament.findAll = function (result) {
41     connection.query("Select * from dovid_viddil",
42         function (err, res) {
43             if (err) {
44                 console.log("error: ", err);
45                 result(null, err);
46             }
47             else {
48                 console.log('Departament : ', res);
49                 result(null, res);
50             }
51         });
52 };
53 // Зміна запису з певним id у БД
54 //id - значення id відділу
55 // dep - значення що змінюється включає назву відділу
56 // result - результат запиту
57 Departament.update = function (id, dep, result) {
58     connection.query("UPDATE dovid_viddil SET Naz_viddil=? WHERE id_viddil = ?",
59         [dep.naz_viddil, id],
60         function (err, res) {
61             if (err) {
62                 console.log("error: ", err);
63                 result(null, err);
64             } else {
65                 result(null, res);
66             }
67         });
68 };
69 // Видалення запису з певним id у БД
70 //id - значення id відділу
71 // result - результат запиту
72 Departament.delete = function (id, result) {
73     connection.query("DELETE FROM dovid_viddil WHERE id_viddil = ?", [id],
74         function (err, res) {
75             if (err) {
76                 console.log("error: ", err);
77                 result(null, err);
78             }
79             else {
80                 result(null, res);
81             }
82         });
83 };
84 // Вказуємо, що експортуємо з модуля Departament
85 module.exports = Departament;

```

20. Далі необхідно розробити controller для цього створимо файл controller\departament.controller.js та імпортуємо у нього модель:

```

// import Departament from "../model/departament.model";
const Departament = require ('../model/departament.model');

```

Далі необхідно написати методи (ENDPOINT), які будуть викликати відповідні методи моделі та перевіряти на відсутність помилок. Назви методів будуть відповідні до методів які описані у моделі.

На вході у кожного методу буде запит, записаний у змінну req.

На виході результат отриманий від моделі, інформація о помилках якщо вони є, і error:false якщо помилок нема у форматі JSON.

```

controller > JS department.controller.js > ...
1 // import Department from "../model/department.model";
2 const Department = require("../model/department.model");
3 // Виведення всієї інформації з таблиці
4 exports.findAll = function (req, res) {
5     Department.findAll(function (err, department) {
6         console.log('controller')
7         if (err)
8             res.send(err);
9         res.send(department);
10    });
11 };
12 // Створення нового запису
13 exports.create = function (req, res) {
14     const new_department = new Department(req.body);
15     //handles null error
16     if (req.body.constructor === Object && Object.keys(req.body).length === 0) {
17         res.status(400).send({ error: true, message: 'Please provide all required field' });
18     } else {
19         Department.create(new_department, function (err, department) {
20             if (err)
21                 res.send(err);
22             res.json({ error: false, message: "department added successfully!", data: department });
23         });
24     }
25 };
26 // Пошук за id
27 exports.findById = function (req, res) {
28     Department.findById(req.params.id, function (err, department) {
29         if (err)
30             res.send(err);
31         res.json(department);
32     });
33 };
34 // редагування інформації
35 exports.update = function (req, res) {
36     if (req.body.constructor === Object && Object.keys(req.body).length === 0) {
37         res.status(400).send({ error: true, message: 'Please provide all required field' });
38     } else {
39         Department.update(req.params.id, new Department(req.body), function (err, department) {
40             if (err)
41                 res.send(err);
42             res.json({ error: false, message: 'department successfully updated' });
43         });
44     }
45 };
46 // видалення інформації
47 exports.delete = function (req, res) {
48     Department.delete(req.params.id, function (err, department) {
49         console.log("HI" + req.params.id);
50         if (err)
51             res.send(err);
52         res.json({ error: false, message: 'department successfully deleted' });
53     });
54 };

```

21. Тепер необхідно написати маршрутизацію для виклику методів для цього у відповідній папці створимо файл `router\department.routes.js` у якому підключимо фреймворк `express.js` та розроблений контролер та припишемо маршрути.

```

router > JS department.routes.js > ...
1 // import express from "express";
2 // import exports from "../controller/department.controller";
3 const express = require('express')
4 // Створюємо новий маршрутизатор
5 const router = express.Router()
6 const departmentController = require('../controller/department.controller');
7 // Перегляд всіх відділів
8 router.get('/', departmentController.findAll);
9 // Створення нового відділу
10 router.post('/', departmentController.create);
11 // Пошук відділу за id
12 router.get('/:id', departmentController.findById);
13 // Редагування відділу id
14 router.put('/:id', departmentController.update);
15 // Видалення відділу за id
16 router.delete('/:id', departmentController.delete);
17 // Експортуємо за замовченням router
18 module.exports = router

```

22. В основному файлі `index.js` підключаємо лише файл маршрутизації. Таким чином основний файл застосунку буде зв'язаний лише з маршрутизатором.

```
JS index.js > ...
1 // Підключаємо express
2 const express = require('express');
3 // Підключаємо bodyParser
4 const bodyParser = require('body-parser');
5 // Встановлюємо порт
6 const PORT = 5000;
7 const app = express();
8 // парсити запити типу content-type - application/x-www-form-urlencoded
9 app.use(bodyParser.urlencoded({ extended: true }));
10 // парсити запити типу content-type - application/json
11 app.use(bodyParser.json());
12 // Обробка get запиту - endpoint
13 // Вхід
14 // Адреса за якою він буде працювати
15 // Функція, що буде виконуватися по запиту на цей endpoint
16 // Параметри функції
17 // req - запит
18 // res - відповідь
19 app.get('/', (req, res) => {
20   // Вказуємо 200 - успішний статус
21   // json - тіло відповіді - повідомлення
22   res.status(200).json("Сервер працює");
23 })
24 // Require department routes
25 const departmentRoutes = require('./router/department.routes')
26 app.use('/api/department', departmentRoutes);
27 // створюємо екземпляр застосунку
28 //вхід
29 //порт
30 //callback функція, яка відпрацює лише у випадку успішного запуску серверу
31 app.listen(PORT, () => console.log("SERVER START!!!"))
32
```

23. Нарешті потрібно перевірити як працює розроблений застосунок для цього застосуємо Для тестування запитів можна використовувати Postman. Для тестування у веб браузері необхідно завантажити Postman agent з сайту <https://www.postman.com> запустити його та обрати у меню, що з'явиться при кліку правої клавішею миші пункт Open Postman у браузері відкриється робоче вікно Postman Enterprise.

24. Запустимо XAMPP Control Panel на якій оберемо сервер Apache та MySQL

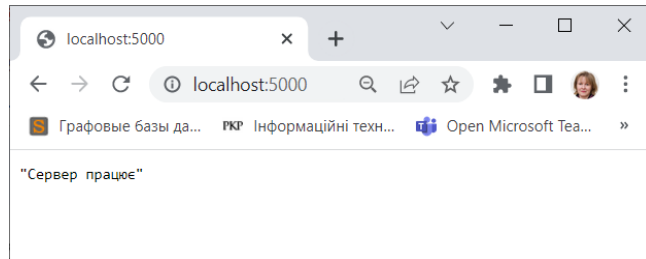
25. Запустимо сервер node.js за допомогою команди `npm run dev`

26. Якщо все пройшло успішно у терміналі побачимо наступне

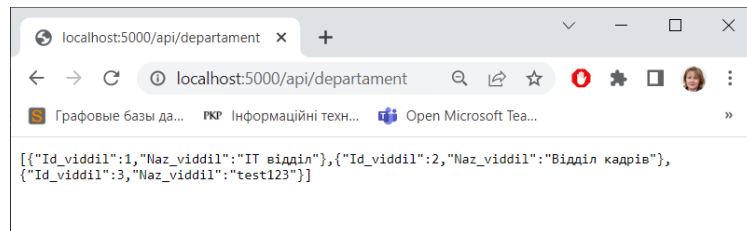
```
D:\KNU\Архітектура ІС\primeri\lab8_2023>npm run dev
> lab8_2023@1.0.0 dev
> nodemon index.js

[nodemon] 2.0.21
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
SERVER START!!!
Database is connected
█
```

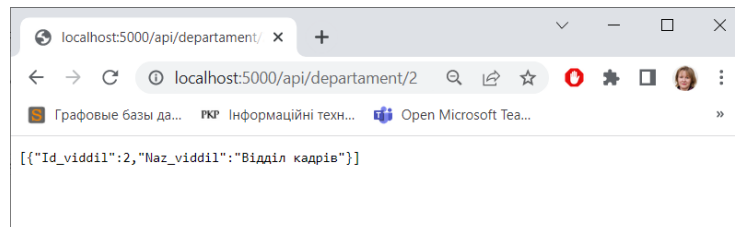
27. Відкриємо веб сторінку за адресою <http://localhost:5000/> та побачимо наступне:



28. Перевіримо роботу GET запиту який виводить усі дані з таблиці з відділами для цього перейдемо за відповідним маршрутом

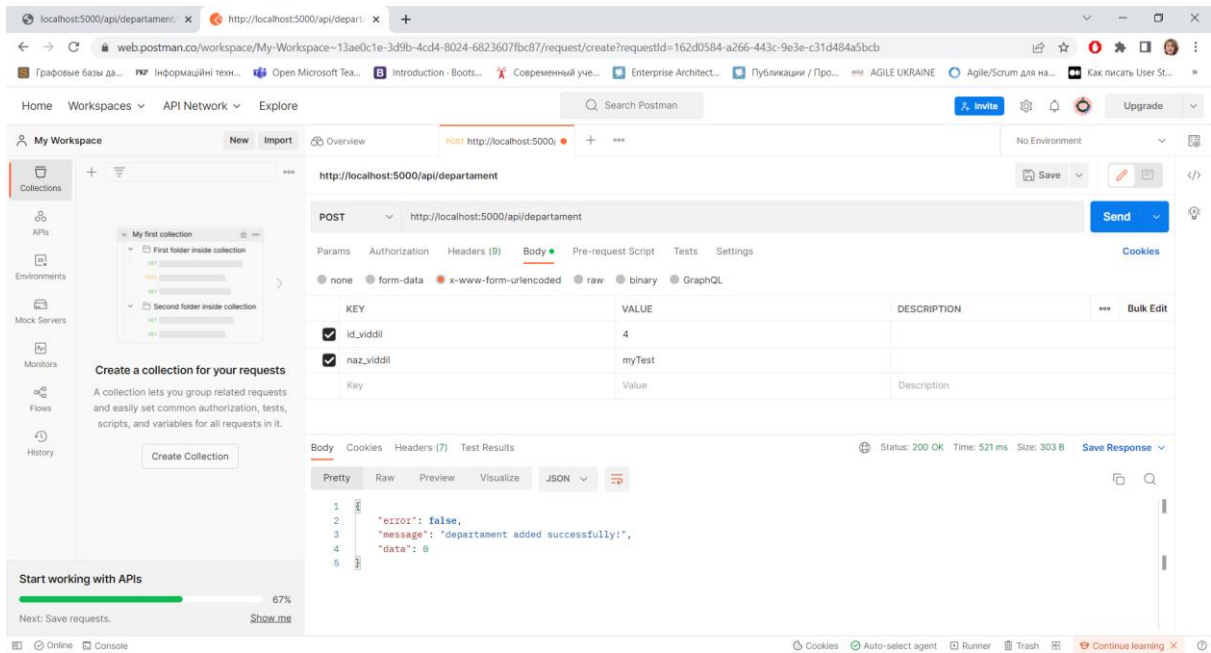


29. Перевіримо GET запит для пошуку за id=2

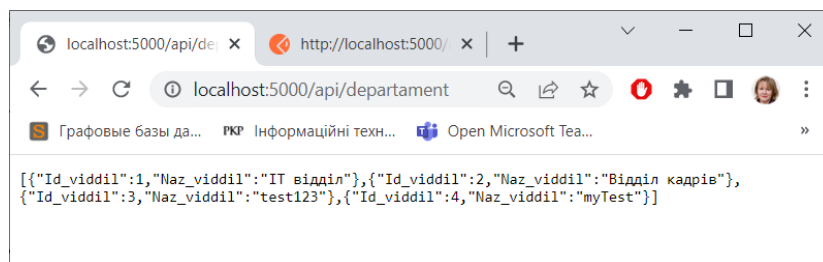


30. Для перевірки інших запитів будемо використовувати Postman. Для цього натиснемо на кнопку Send Request та перейдемо до вікна роботи з запитами. У поле з URL будемо вводити відповідну адресу у Body буде вводити параметри. Для виконання запитів натиснемо кнопку Send. Розглянемо тестування кожного із запитів

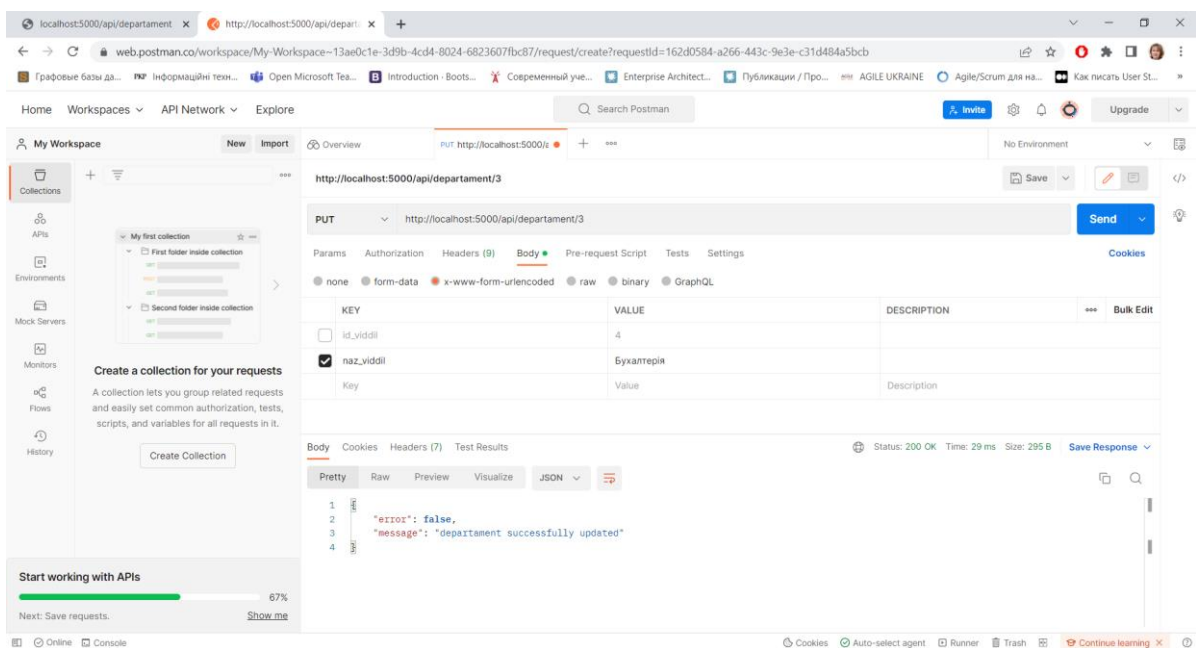
31. Запит додавання запису



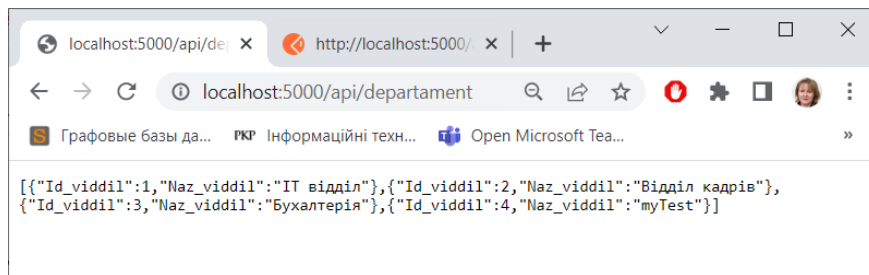
## Результат



## 32. Запит редагування запису за id=3

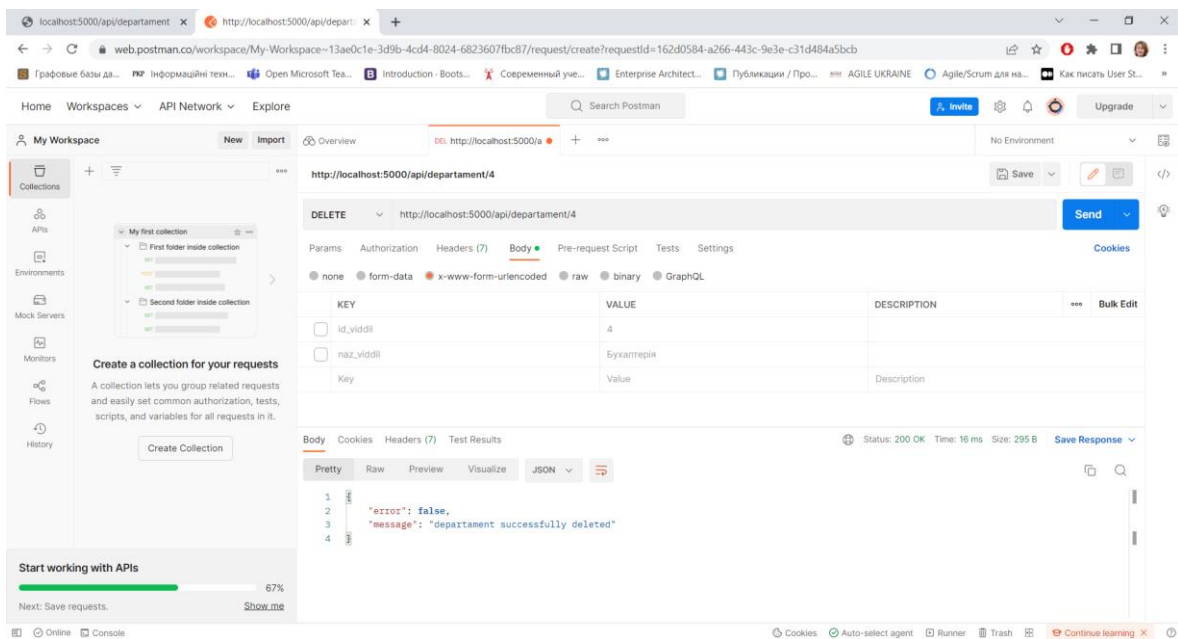


## Результат



```
[{"Id_viddil":1,"Naz_viddil":"IT відділ"}, {"Id_viddil":2,"Naz_viddil":"Відділ кадрів"}, {"Id_viddil":3,"Naz_viddil":"Бухгалтерія"}, {"Id_viddil":4,"Naz_viddil":"myTest"}]
```

## 33. Запит видалення запису з id=4

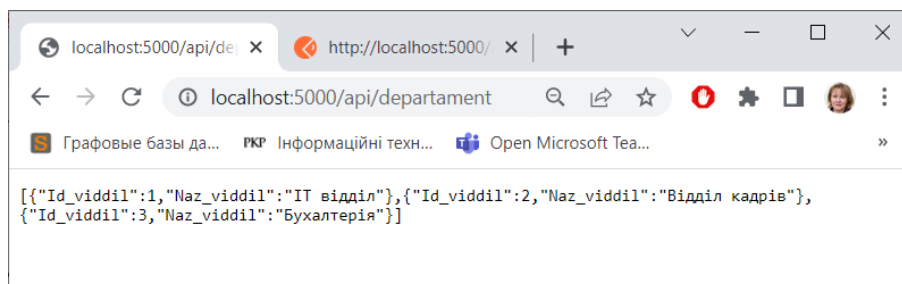


DELETE http://localhost:5000/api/departament/4

KEY	VALUE	DESCRIPTION
id_viddil	4	
naz_viddil	Бухгалтерія	
Key	Value	Description

```
1 {  
2   "error": false,  
3   "message": "department successfully deleted"  
4 }
```

## Результат



```
[{"Id_viddil":1,"Naz_viddil":"IT відділ"}, {"Id_viddil":2,"Naz_viddil":"Відділ кадрів"}, {"Id_viddil":3,"Naz_viddil":"Бухгалтерія"}]
```

## Лабораторна робота № 9

### Створення адмін панелі для роботи з БД

**Мета роботи:** Навчитися створювати застосунки з інтерфейсом на node.js для роботи з таблицями БД.

#### Завдання на лабораторну роботу

Використовуючи розроблену у попередніх лабораторних роботах БД MySQL та застосунок для роботи з нею, розробити інтерфейсну частину серверного застосунку для роботи з декількома таблицями БД (**не менше 3**), а саме для виконання операцій CRUD (створення, читання, зміни та видалення даних).

#### Теоретичні відомості

*Ejs* (<https://ejs.co/>)

*EJS* — це проста мова шаблонів, яка дозволяє створювати розмітку HTML за допомогою простого JavaScript.

*Особливості:*

- Швидка компіляція та рендеринг
- Прості шаблонні теги: `<% %>`
- Спеціальні роздільники (наприклад, використовуйте `[? ?]` замість `<% %>`)
- Можна використовувати під шаблони
- Поставляється з CLI
- Підтримка як сервера, так і браузера
- Статичне кешування проміжного JavaScript
- Статичне кешування шаблонів
- Відповідає системі Express view

#### Приклад виконання

1. Спочатку проінсталуємо `jes` у застосунок, що був розроблений у попередній лабораторній роботі за допомогою команди `npm install ejs`

2. Опишемо у файлі `package.json` у вкладці `dependencies` нову залежність:

```
"ejs": "^1.0.0"
```



3. Створимо директорію для збереження візуальної частини додатку з іменем views всередині якої створимо файл department.ejs та опишемо у ньому шаблон для виведення переліку відділів, який будемо отримувати у Get запиті. Виводити будемо у таблиці. Для реалізації адаптивності та оформлення будемо використовувати Bootstrap

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="../public/style/main.css" rel="stylesheet" type="text/css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">
  <title>Відділи</title>
</head>
<body class="body">
  <h1 class="text-center">Список відділів</h1>
  <div class="table-responsive">
    <table class="table table-primary table-hover table-sm">
      <thead>
        <tr>
          <th>Код відділу</th>
          <th>Назва</th>
        </tr>
      </thead>
      <tbody>
        <% for (var i = 0; i < Department.length; i++) { %>
          <tr>
            <td>
              <%= Department[i].Id_viddil %>
            </td>
            <td>
              <%= Department[i].Naz_viddil %>
            </td>
            <td style="background-color: #white;">
              <a class="btn btn-danger" href="/api/department/delete/<%= Department[i].Id_viddil%>">Видалити</a>
            </td>
          </tr>
          <tr>
            <td style="background-color: #white;">
              <a class="btn btn-info" href="/api/department/<%= Department[i].Id_viddil%>">Редагувати</a>
            </td>
          </tr>
        <% } %>
      </tbody>
    </table>
  </div>
</body>
```

Додаємо свої стилі до оформлення сторінки для цього необхідно, щоб сервер Node.js їх побачив. Тому у файл index.js необхідно додати наступний рядок

```
///Щоб на сервері бачити усі рішення
app.use(express.static("."));
```

4. Тапер створимо папку public у якій створимо файл main.css у якому пропишемо додаткові стилі для оформлення таблиці та сторінки.

5. З'єднання Views та розробленої раніше функціональності буде відбуватися у контролері. Змінимо його:

```

// !Додаємо для виведення
const express = require('express')
var app = express();
var path = require('path');
app.engine('ejs', require('ejs').__express);
const Department = require('./model/department.model');
// Виведення всій інформації з таблиці
exports.findAll = function (req, res) {
  Department.findAll(function (err, department) {
    console.log('controller')
    if (err)
      res.send(err);
    // !З'єднуємо з файлом виведення
    res.render('department.ejs', { Department: department });
    // res.send(department);
  });
};

```

6. Тепер при виконанні get запиту за адресою <http://localhost:5000/api/department> буде виводитися таблиця з даними.

**Список відділів**

Код відділу	Назва		
1	IT відділ	Видалити	Редагувати
2	Відділ кадрів	Видалити	Редагувати
3	Бухгалтерія	Видалити	Редагувати
4	Відділ закупівель	Видалити	Редагувати
5	Відділ по роботі з клієнтами	Видалити	Редагувати
6	Склад	Видалити	Редагувати

7. Додаємо інтерфейс для додавання нових записів у файл department.ejs

```

<h2 class="text-center">
  Додати відділ
</h2>
<form action="department" method="POST" class="text-center">
  <div class="mb-3">
    <label class="form-label">
      ID відділу
      <input type="number" placeholder="id" class="form-control" id="Id_viddil" name="Id_viddil" min="1">
    </label class="form-label">
    <label>
      Назва відділу
      <input type="text" class="form-control" placeholder="Назва" id="Naz_viddil" name="Naz_viddil">
    </label>
  </div>
  <button type="submit" class="btn btn-primary">Додати</button>
</form>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-Mrcw6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxXWM"
  crossorigin="anonymous"></script>
</body>
</html>

```

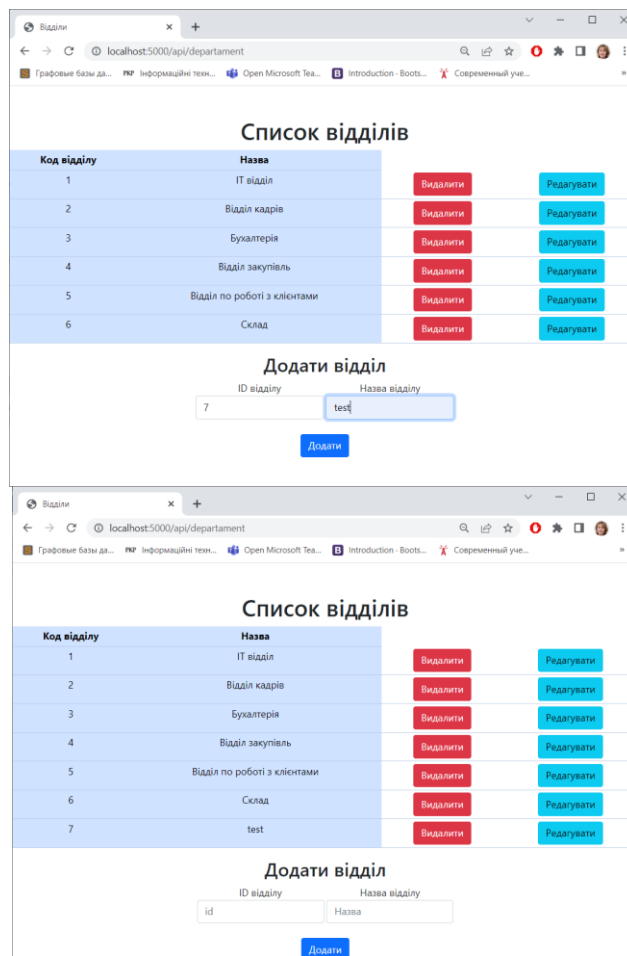
8. Для того, щоб після додавання записи одразу відображалися у таблиці змінимо метод exports.create у файлі з контролером додавши редирект на сторінку з департаментом

```

// Створення нового запису
exports.create = function (req, res) {
  const new_department = new Department(req.body);
  //handles null error
  if (req.body.constructor === Object && Object.keys(req.body).length === 0) {
    res.status(400).send({ error: true, message: 'Please provide all required field' });
  } else {
    Department.create(new_department, function (err, department) {
      if (err)
        res.send(err);
      // Переходимо на сторінку з таблицею відділів
      res.redirect('/api/department')
    });
  }
};
};

```

9. Протестуємо роботу застосунку додавши новий запис про відділ



10. Тепер додамо до адмін панелі можливість видалення турів для цього необхідно у файлі з роутингом вид запити з DELETE на GET

```

// Видалення відділу за id
// !router.delete('/:id', departmentController.delete);
router.get('/delete/:id', departmentController.delete);

```

11. Та прописати редирект у файлі з контролером

```

// видалення інформації
exports.delete = function (req, res) {
  Department.delete(req.params.id, function (err, department) {
    console.log("HI" + req.params.id);
    if (err)
      res.send(err);
    res.redirect('/api/department')
    // res.json({ error: false, message: 'department successfully deleted' });
  });
};

```

12. Найскладнішою є операція зі зміни значення. Для цієї операції створимо окремий Views `department_edit.ejs`

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity=
8 <title>Редагування</title>
9 </head>
10 <body>
11   <h1 class="text-center">Редагувати відділ</h1>
12   <form action="" method="POST" class="text-center">
13     <div class="mb-3">
14       <label class="form-label">
15         ID відділу
16         <input type="number" placeholder="id" class="form-control" id="Id_viddil" name="Id_viddil" min="1"
17         value=<%= Department[0].Id_viddil %>
18       >
19     </label class="form-label">
20     <label>
21       Назва відділу
22       <input type="text" class="form-control" placeholder="Назва" id="Naz_viddil" name="Naz_viddil"
23       style="width:300px" value="<%= Department[0].Naz_viddil %>"
24     >
25   </label>
26 </div>
27 <button type="submit" class="btn btn-primary" formaction="/api/department/put/<%= Department[0].Id_viddil%>">
28   Редагувати</button>
29 </form>
30 <a class="btn btn-info" href="/api/department">Повернутися</a>
31 </body>
32 </html>

```

13. Дана форма буде викликатися при пошуку запису змінюємо відповідний метод у файлі контролера, крім того після зміни знову необхідно перейти на екран з таблицею, тому змінюємо метод `update`

```

35 // Пошук за id
36 exports.findById = function (req, res) {
37   Department.findById(req.params.id, function (err, department) {
38     if (err)
39       res.send(err);
40     // Перехід на сторінку редагування
41     res.render('department_edit.ejs', { Department: department });
42     // res.json(department);
43   });
44 };
45 // редагування інформації
46 exports.update = function (req, res) {
47   if (req.body.constructor === Object && Object.keys(req.body).length === 0) {
48     res.status(400).send({ error: true, message: 'Please provide all required field' });
49   } else {
50     Department.update(req.params.id, new Department(req.body), function (err, department) {
51       if (err)
52         res.send(err);
53       // Повернення на сторінку з таблицею відділів
54       res.redirect('/api/department')
55       // res.json({ error: false, message: 'department successfully updated' });
56     });
57   }
58 };

```

14. Тепер потрібно змінити у файлі роутингу вид запиту з PUT на POST

```
// Редагування відділу id
// !router.put('/:id', departmentController.update);
router.post('/put/:id', departmentController.update)
```

15. Проведемо тестування редагування запису

The image consists of three browser screenshots illustrating the editing process:

- Top Left:** A screenshot of the 'Список відділів' (Department List) page. The table shows 7 departments. The 7th department, 'test', is highlighted. Below the table is a 'Додати відділ' (Add Department) form with fields for 'ID відділу' and 'Назва відділу'. A blue arrow points from the 'test' row to the 'Додати відділ' form.
- Top Right:** A screenshot of the 'Редагувати відділ' (Edit Department) page. The 'ID відділу' field contains '7' and the 'Назва відділу' field contains 'test new'. A blue arrow points from the 'test new' text to the 'Додати відділ' form in the previous screenshot.
- Bottom:** A screenshot of the 'Список відділів' (Department List) page after the update. The 7th department is now 'test new'. A blue arrow points from the 'test new' text in this screenshot to the 'test new' text in the 'Редагувати відділ' page.

16. Аналогічним чином розробимо View для інших таблиць.  
17. Створимо головну сторінку для роботи з таблицями index.ejs

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link href="./public/style/main.css" rel="stylesheet" type="text/css">
8   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="
9   <title>ІС відділу кадрів</title>
10 </head>
11 <body>
12   <header>
13
14   <nav class="main-menu">
15     <a class="btn btn-info" href="/api/position">Посади</a>
16     <a class="btn btn-info" href="/api/departament">Відділи</a>
17   </nav>
18 </header>
19 </body>
20 </html>

```

18. Додамо переходи на цю сторінку на сторінках відділів та посад

```

<a class="btn btn-outline-secondary" href="/" style="margin-left: 50px;">На
головну</a>

```

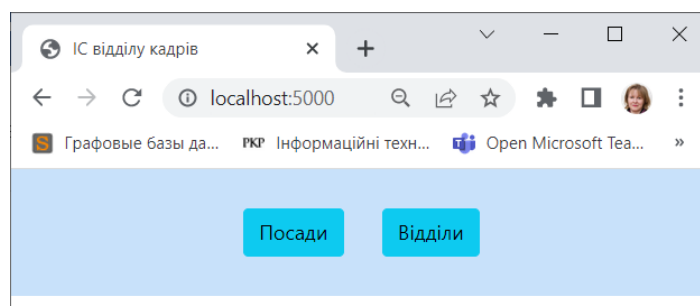
19. Підключимо сторінку у головному файлі index.js

```

app.get('/', (req, res) => {
  // Вказуємо 200 - успішний статус
  // json - тіло відповіді - повідомлення
  // res.status(200).json("Сервер працює");
  //! Запуск головної сторінки
  res.render('index.ejs');
})

```

20. Вигляд головної сторінки застосунку наведено нижче



## Список літератури

1. The Open Group Architecture Framework (TOGAF). URL: <https://www.opengroup.org/togaf>
2. Software Engineering Institute (SEI) - Architecture Practices: URL: <https://www.sei.cmu.edu/architecture/>
3. Microsoft Architecture Guide. URL: <https://docs.microsoft.com/en-us/azure/architecture/guide/>
4. "Software Architecture in Practice" by Len Bass, Paul Clements, Rick Kazman. URL: <https://www.amazon.com/Software-Architecture-Practice-3rd-Engineering/dp/0321815734>
5. "Patterns of Enterprise Application Architecture" by Martin Fowler. URL: <https://www.amazon.com/Patterns-Enterprise-Application-Architecture-Martin/dp/0321127420>
6. "Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. URL: <https://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612>
7. "Clean Architecture: A Craftsman's Guide to Software Structure and Design" by Robert C. Martin. URL: <https://www.amazon.com/Clean-Architecture-Craftsmans-Software-Structure/dp/0134494164>
8. "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions" by Gregor Hohpe, Bobby Woolf. URL: <https://www.amazon.com/Enterprise-Integration-Patterns-Designing-Deploying/dp/0321200683>

### Курси та навчальні матеріали онлайн:

1. Coursera: "Software Architecture & Design" course by University of Alberta.
2. edX: "Software Architecture for the Internet of Things" course by EIT Digital.
3. Udemy: "Software Architecture: Complete Introduction For Beginners" by Academind.

Навчально-методичне видання

## Архітектура інформаційних систем

Методичні вказівки  
до виконання лабораторних робіт № 1-9  
для підготовки здобувачів другого магістерського рівня вищої освіти  
спеціальностей 121 «Інженерія програмного забезпечення»,  
122 «Комп'ютерні науки» та  
126 «Інформаційні системи і технології»

Укладачі: Т.А Гончаренко,  
С.В. Білощицька

Комп'ютерне верстання

Підписано до друку 22.02.2024 Формат 60 × 84 1/16  
Ум. друк. арк. 1,16. Обл.-вид. арк. 1,25.  
Електронний документ. Вид № 59/III-17.

Видавець і виготовлювач  
Київський національний університет будівництва і архітектури

Повітрофлотський проспект, 31, Київ, Україна, 03037  
Свідоцтво про внесення до Державного реєстру суб'єктів  
видавничої справи ДК № 808 від 13.02.2002 р.