МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ЛАБОРАТОРНА РОБОТА № 4

РЕАЛІЗАЦІЯ BACK-END НА ОСНОВІ PYTHON FLASK

Варіант 83

Виконав: ст. гр. ІР-24,

Дзень С. А.

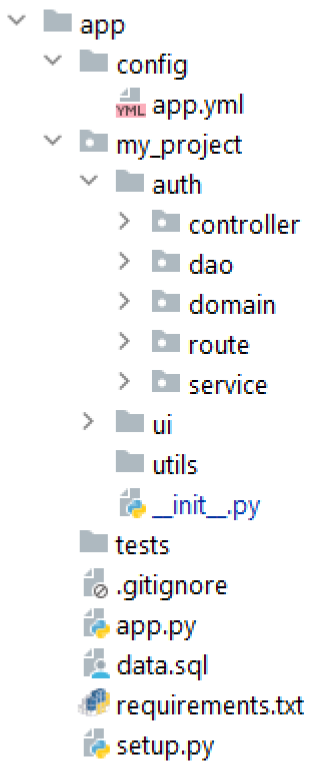
Прийняла: к. т. н., стар. вик.

Лагун І. І.

Львів-2024

Порядок виконання роботи

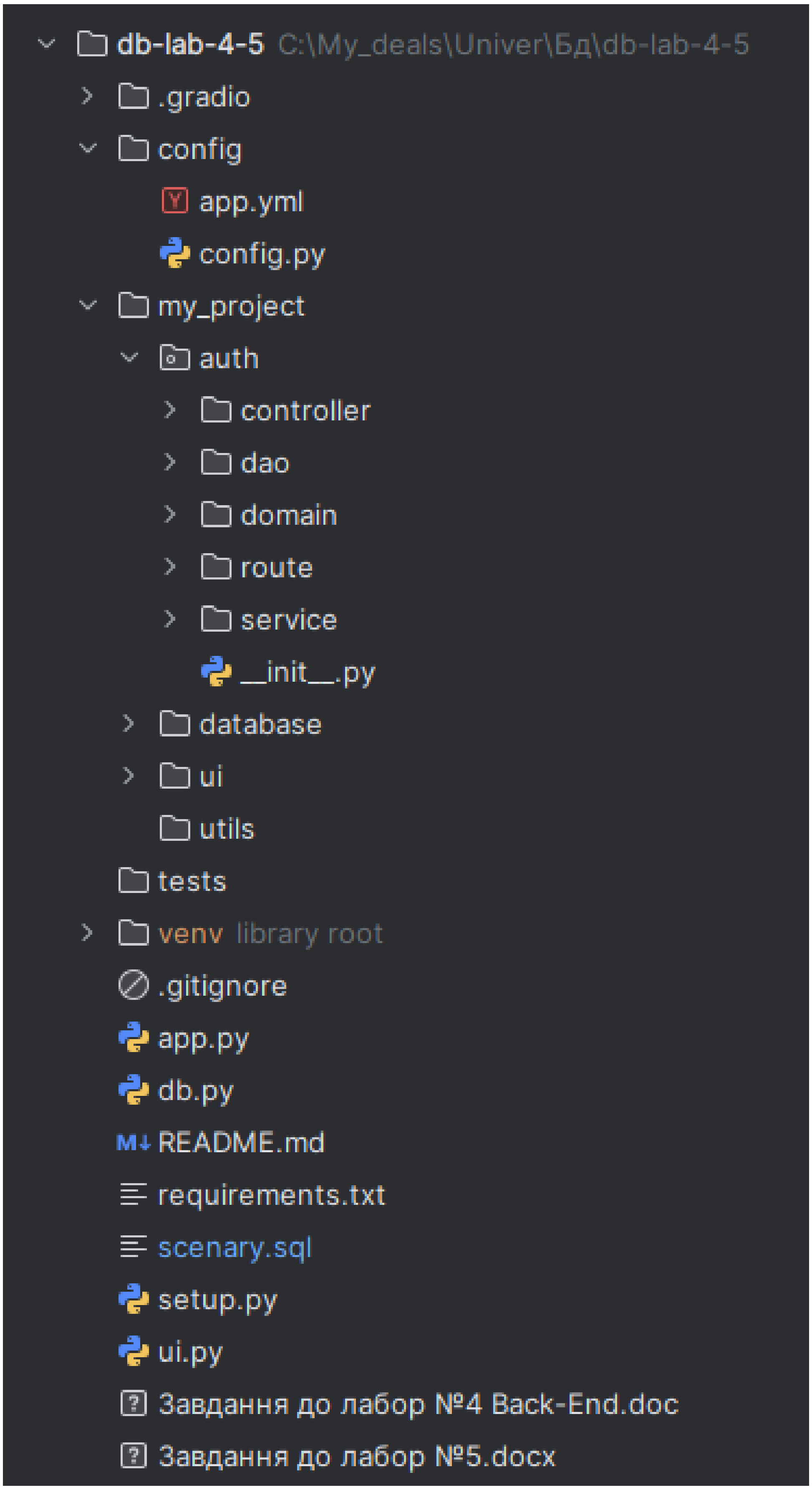
1. Для спроектованої бази даних реалізувати Back-End проект з використанням Flask+Python з підключенням до MySQL.
2. Структура проекту має мати приблизно такий вигляд:



```
app
  config
    app.yml
  my_project
    auth
      controller
      dao
      domain
      route
      service
    ui
    utils
    __init__.py
  tests
  .gitignore
  app.py
  data.sql
  requirements.txt
  setup.py
```

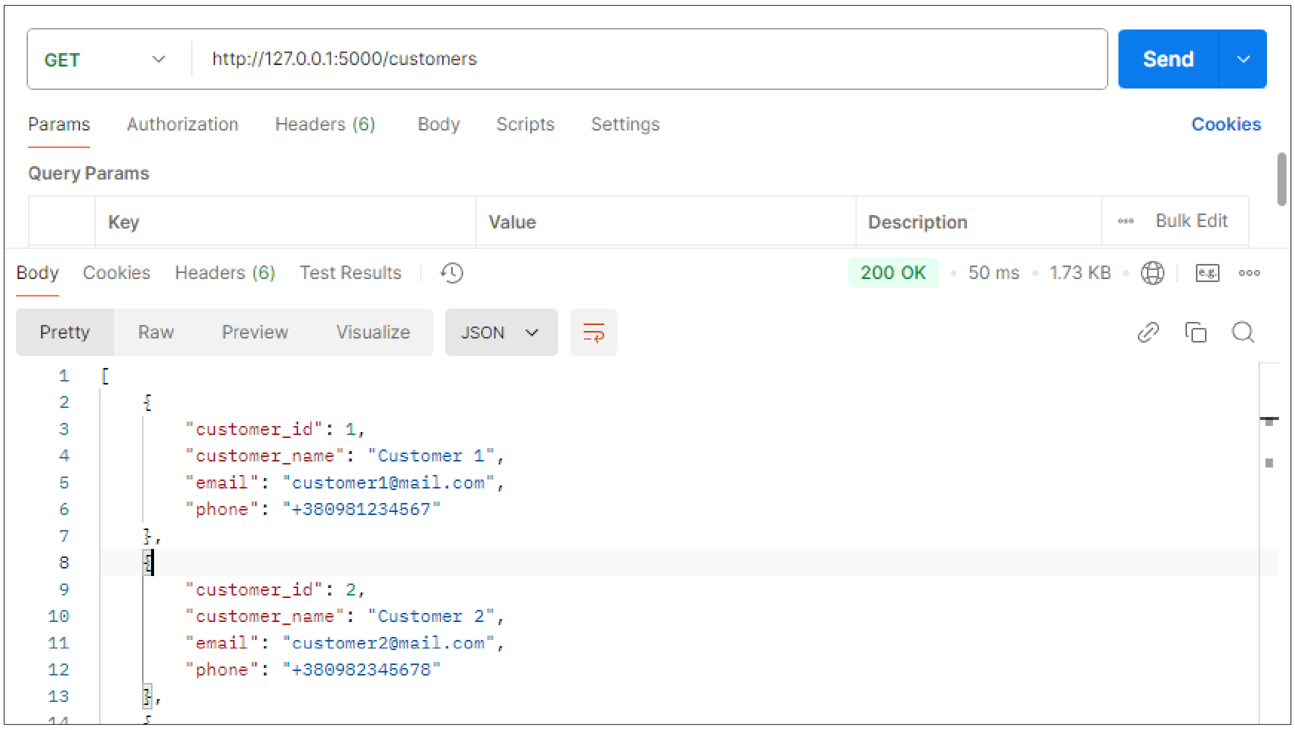
3. Web-контролери повинні опрацьовувати запити для CRUD-операцій та повертати необхідні дані у вигляді DTO-об'єктів.
4. Сервіси повинні містити основну бізнес-логіку для роботи з даними.
5. DAO повинні містити усі необхідні методи для роботи з даними в БД.
6. Клієнтську роботу з даними протестувати через Postman:
   - вивід даних з таблиць;
   - вставку даних у таблиці;
   - обновлення даних у таблицях.
   - видалення даних з таблиці;
   - вивід даних зі сторони зв'язку M:1, тобто, наприклад, для кожного міста вивести людей, які в ньому проживають;
   - вивід даних зі стикувальної таблиці зв'язку M:M, тобто вивести для кожного суб'єкта з одної таблиці усі суб'єкти другої таблиці, які приєднані до нього.
7. Реалізований проект слід залити на GitHub.

Посилання на гітхаб:
https://github.com/sergiyclas/db-lab-4-5.git



```
db-lab-4-5  C:\My_deals\Univer\Бд\db-lab-4-5
  .gradio
  config
    app.yml
    config.py
  my_project
    auth
      controller
      dao
      domain
      route
      service
      __init__.py
    database
    ui
    utils
  tests
  venv  library root
  .gitignore
  app.py
  db.py
  README.md
  requirements.txt
  scenary.sql
  setup.py
  ui.py
  Завдання до лабор №4 Back-End.doc
  Завдання до лабор №5.docx
```

Метод GET для всіх юзерів з таблиці:

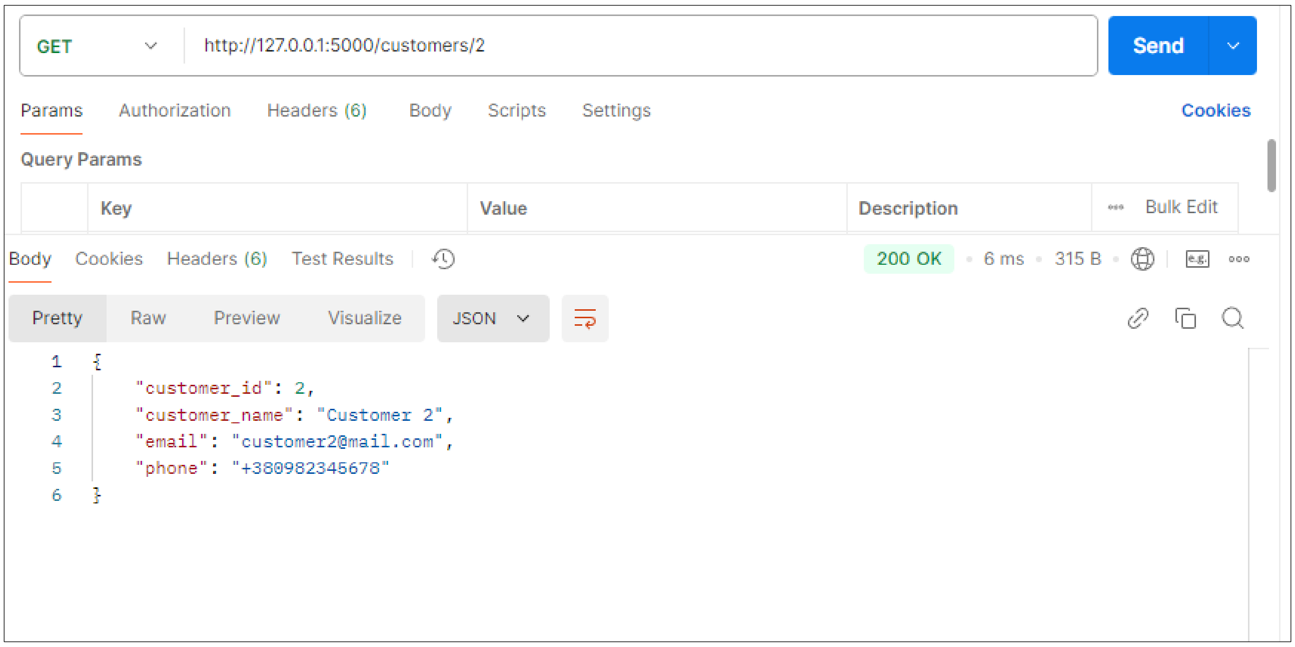


http://127.0.0.1:5000/customers

200 OK · 50 ms · 1.73 KB

```
[
    {
        "customer_id": 1,
        "customer_name": "Customer 1",
        "email": "customer1@mail.com",
        "phone": "+380981234567"
    },
    {
        "customer_id": 2,
        "customer_name": "Customer 2",
        "email": "customer2@mail.com",
        "phone": "+380982345678"
    },
```

Метод GET для окремих юзерів з таблиці:

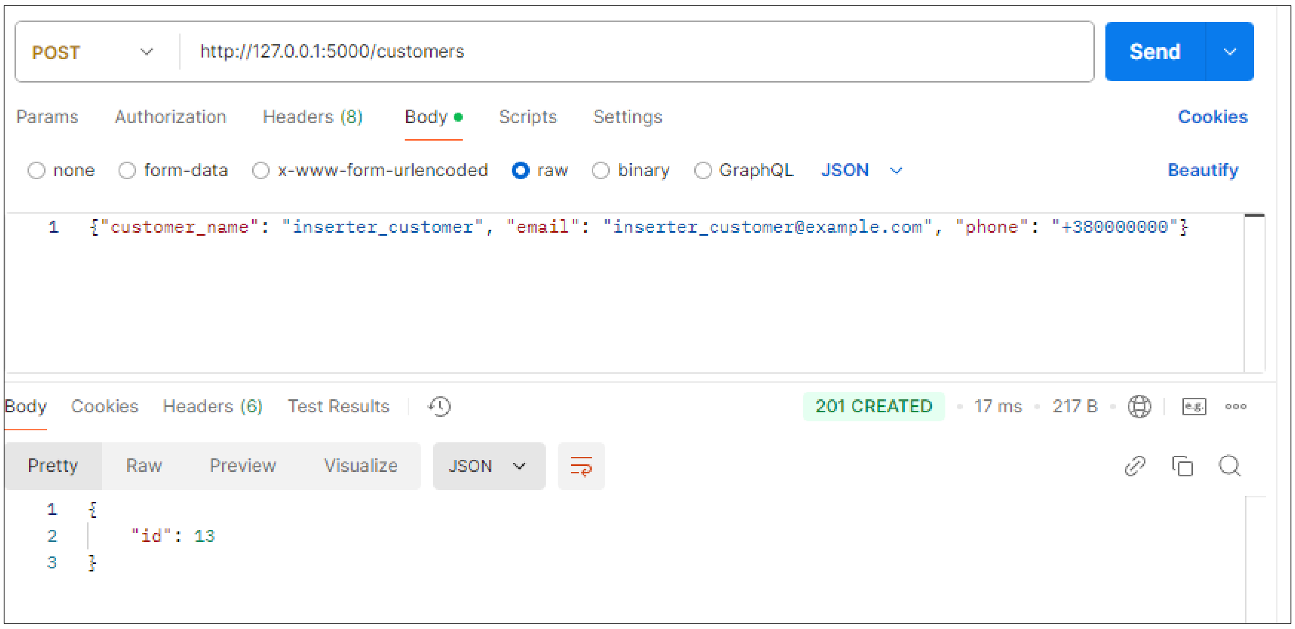


http://127.0.0.1:5000/customers/2

200 OK · 6 ms · 315 B

```
{
    "customer_id": 2,
    "customer_name": "Customer 2",
    "email": "customer2@mail.com",
    "phone": "+380982345678"
}
```

Метод POST:

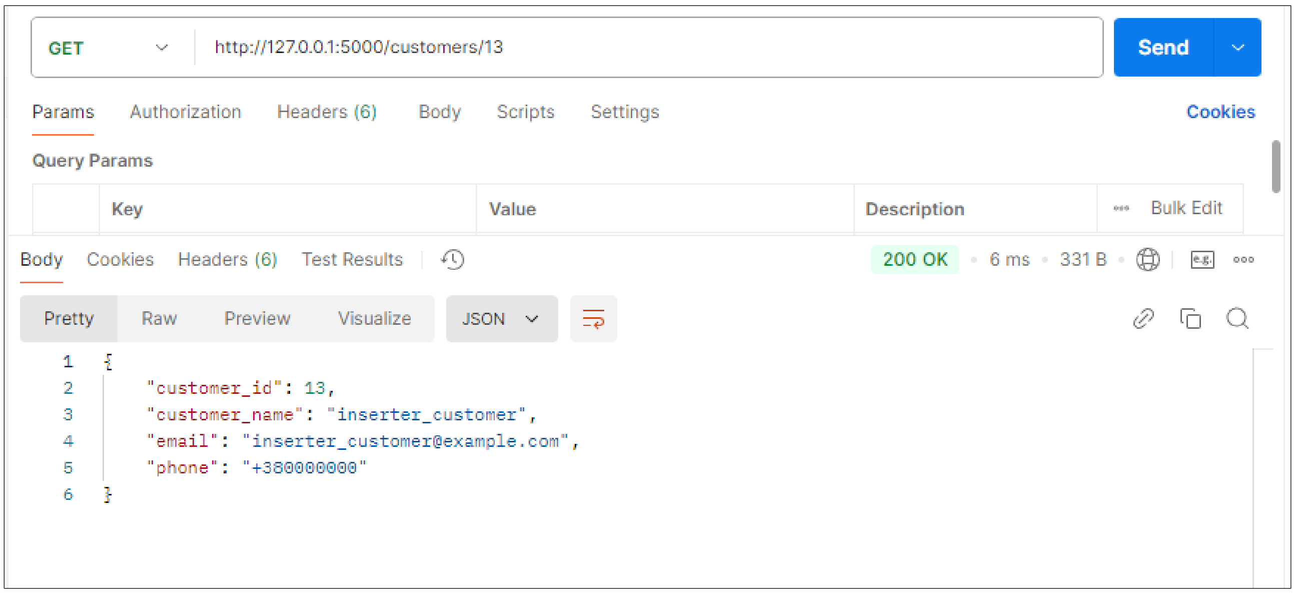


POST http://127.0.0.1:5000/customers

```
{"customer_name": "inserter_customer", "email": "inserter_customer@example.com", "phone": "+380000000"}
```

201 CREATED • 17 ms • 217 B

```
{
    "id": 13
}
```

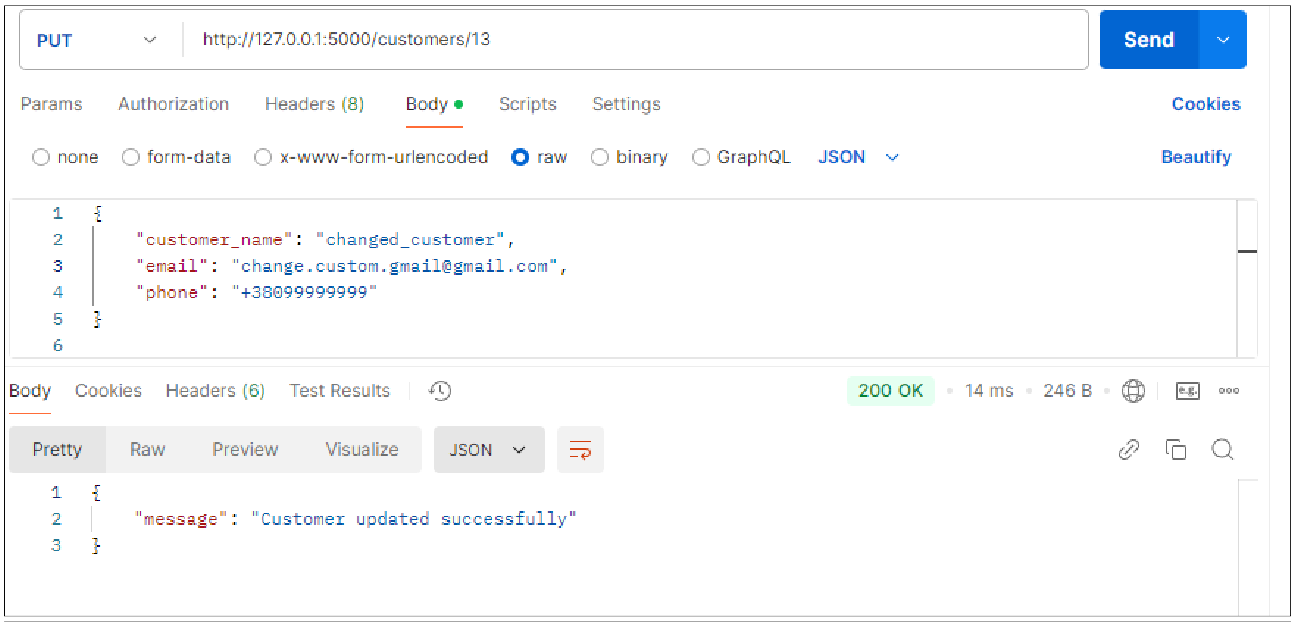


GET http://127.0.0.1:5000/customers/13

200 OK • 6 ms • 331 B

```
{
    "customer_id": 13,
    "customer_name": "inserter_customer",
    "email": "inserter_customer@example.com",
    "phone": "+380000000"
}
```

Метод PUT:



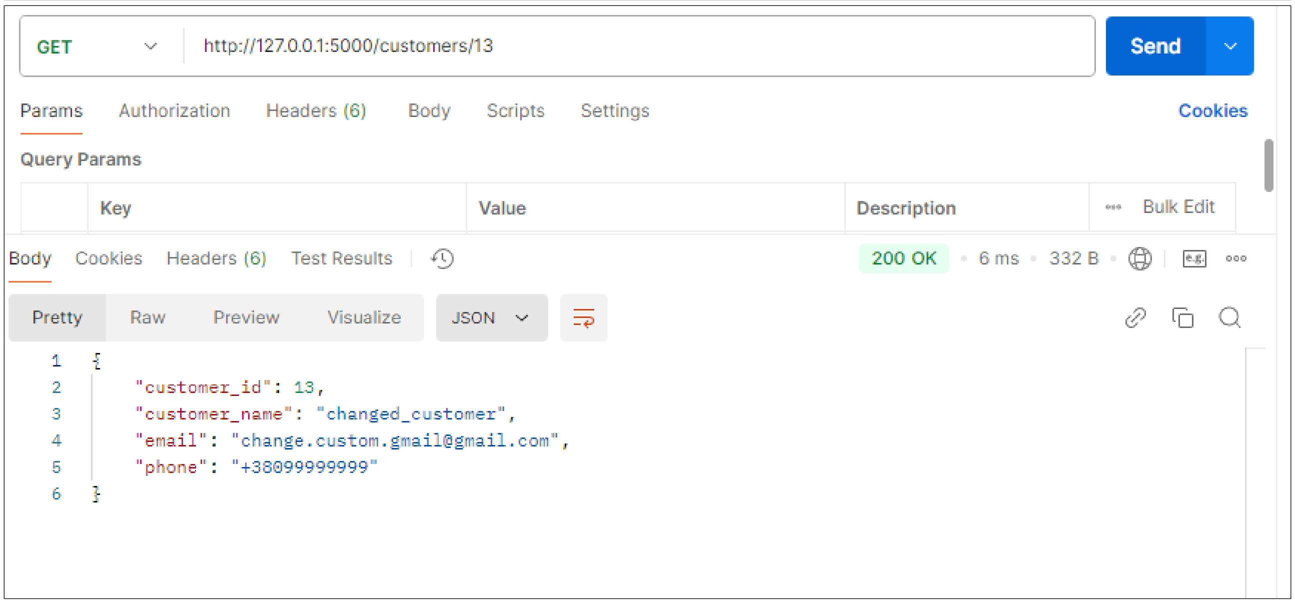
http://127.0.0.1:5000/customers/13

Body: [✓ raw] JSON

```
{
    "customer_name": "changed_customer",
    "email": "change.custom.gmail@gmail.com",
    "phone": "+38099999999"
}
```

200 OK • 14 ms • 246 B

```
{
    "message": "Customer updated successfully"
}
```

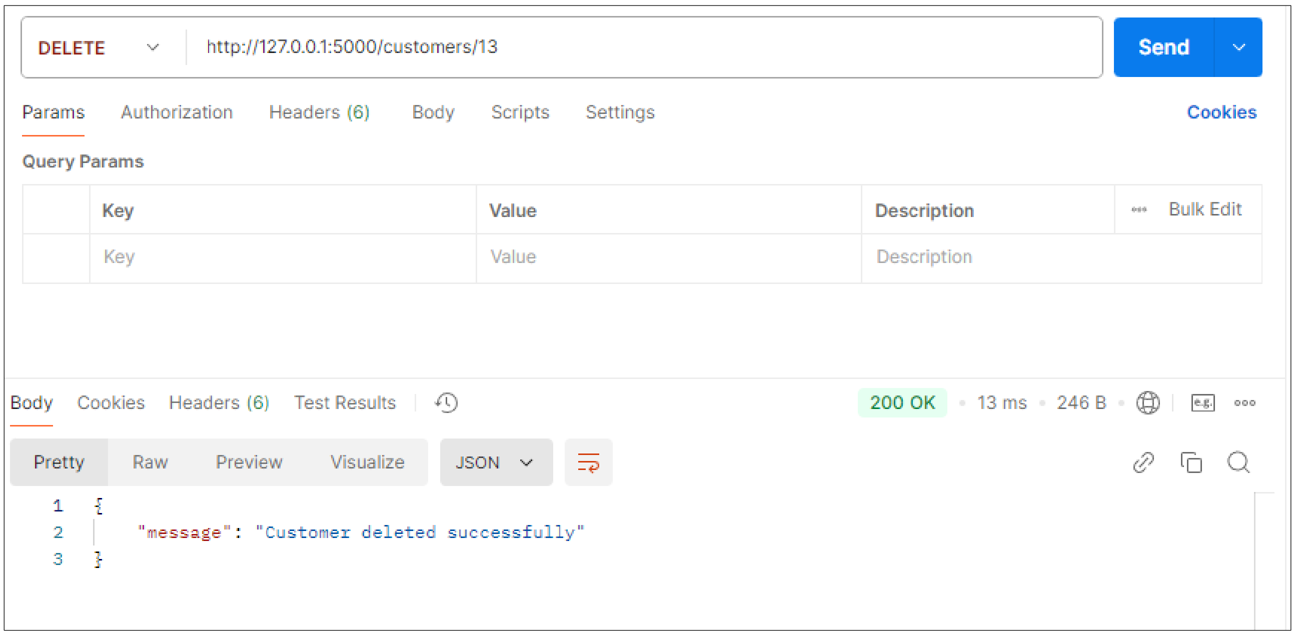


http://127.0.0.1:5000/customers/13

200 OK • 6 ms • 332 B

```
{
    "customer_id": 13,
    "customer_name": "changed_customer",
    "email": "change.custom.gmail@gmail.com",
    "phone": "+38099999999"
}
```

Метод DELETE:



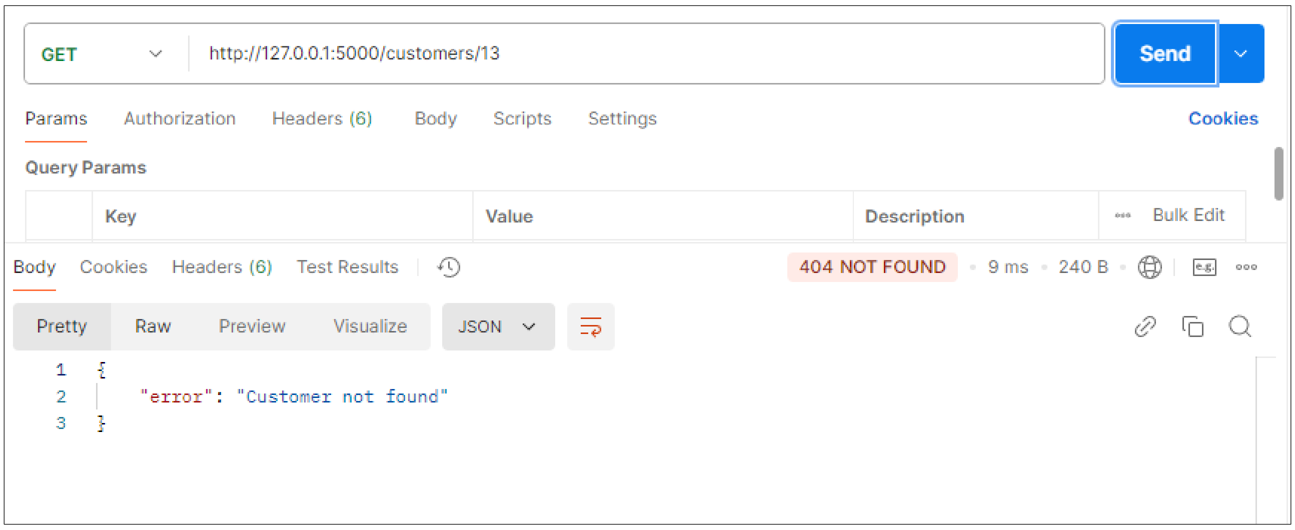
http://127.0.0.1:5000/customers/13

Query Params

Key	Value	Description
Key	Value	Description

200 OK • 13 ms • 246 B

```
{
    "message": "Customer deleted successfully"
}
```



http://127.0.0.1:5000/customers/13

Query Params

Key	Value	Description

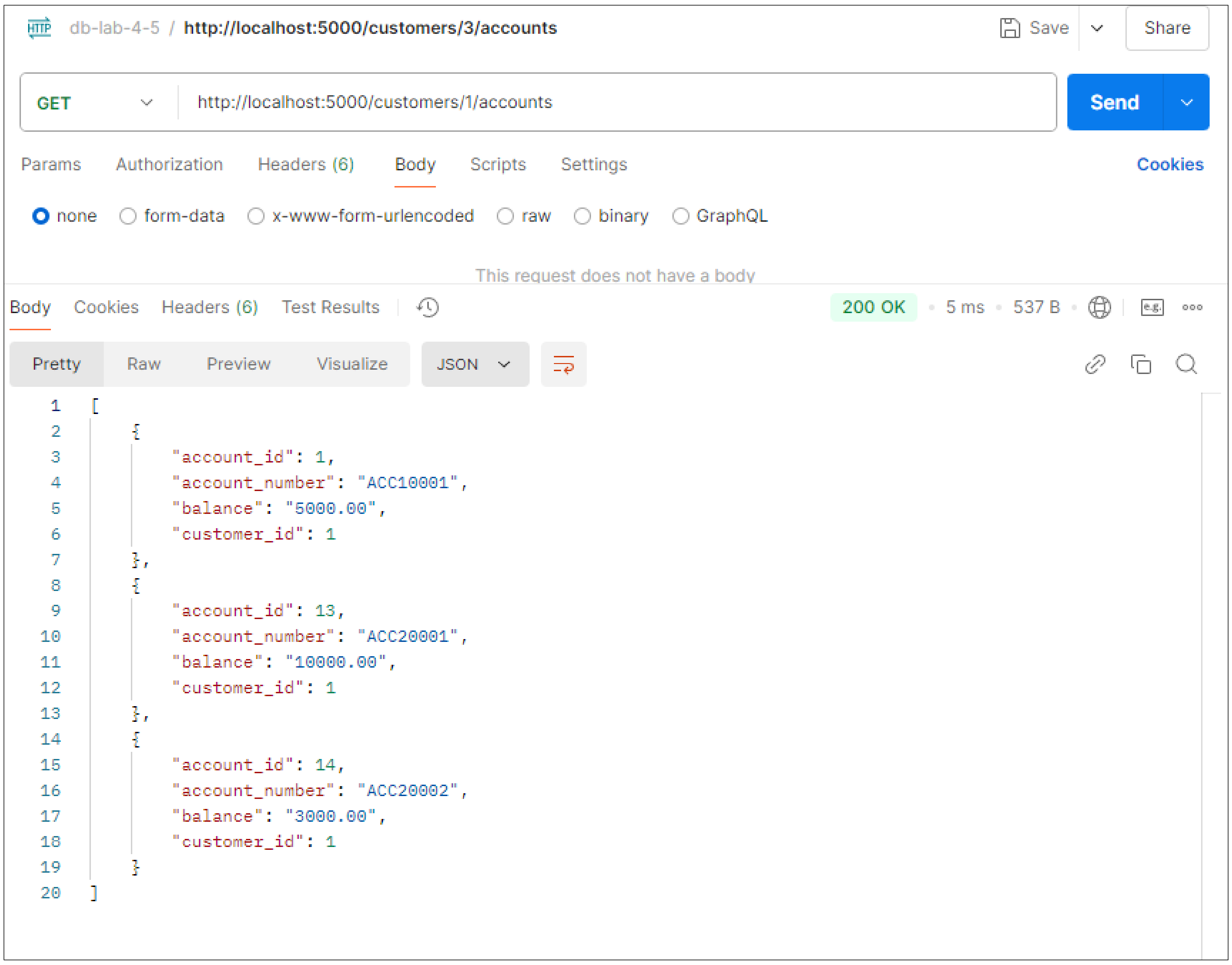
404 NOT FOUND • 9 ms • 240 B

```
{
    "error": "Customer not found"
}
```

Зв'язок 1:M :

Табличка accounts яка з’єднується з табличкою customers:

- Виводяться усі наявні акаунти вказаного користувача



GET http://localhost:5000/customers/1/accounts

none [✓] form-data x-www-form-urlencoded raw binary GraphQL

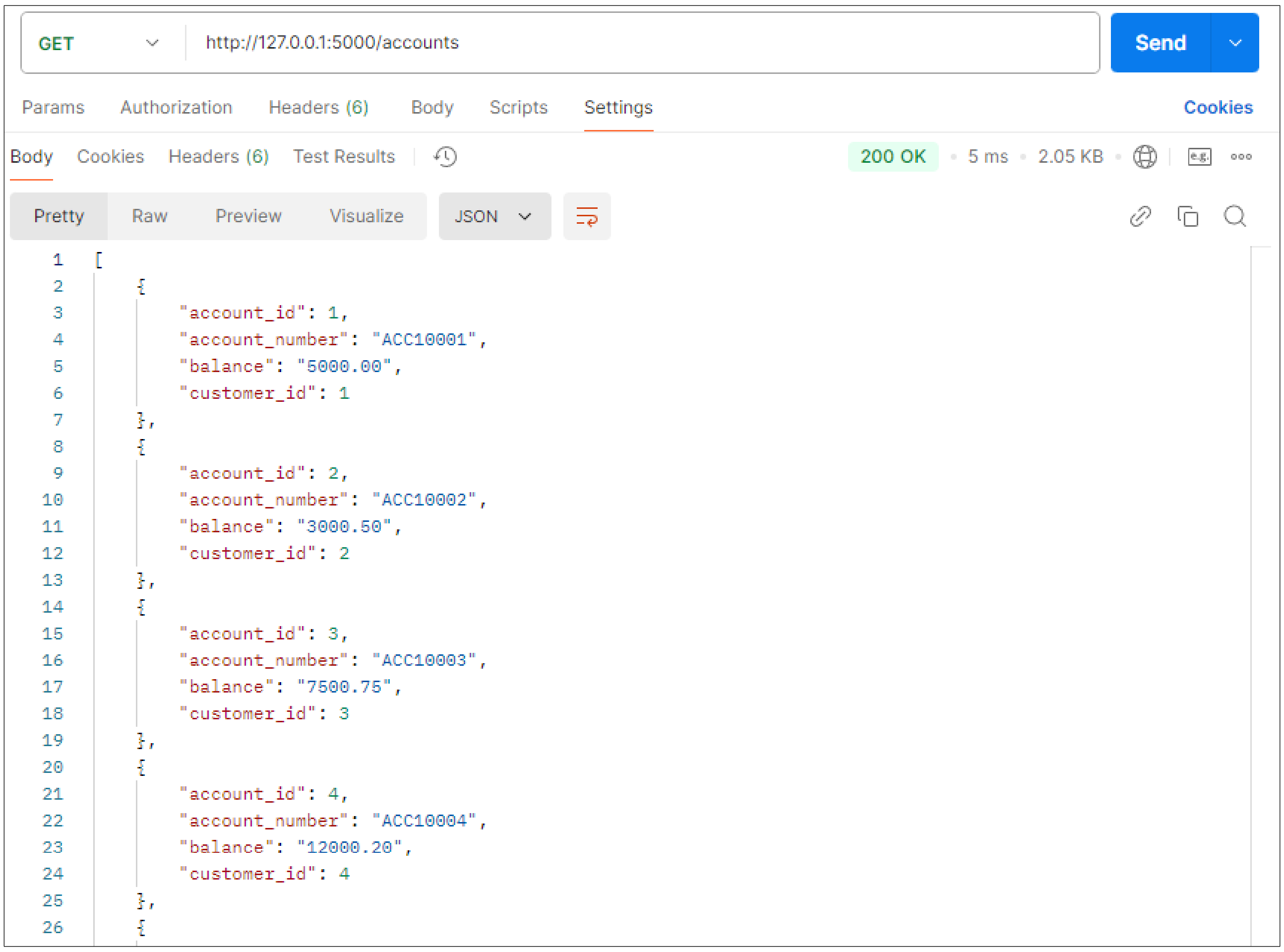
This request does not have a body

200 OK • 5 ms • 537 B

```
[
    {
        "account_id": 1,
        "account_number": "ACC10001",
        "balance": "5000.00",
        "customer_id": 1
    },
    {
        "account_id": 13,
        "account_number": "ACC20001",
        "balance": "10000.00",
        "customer_id": 1
    },
    {
        "account_id": 14,
        "account_number": "ACC20002",
        "balance": "3000.00",
        "customer_id": 1
    }
]
```

Зв'язок M:M між табличками accounts та transactions:

Табличка accounts:

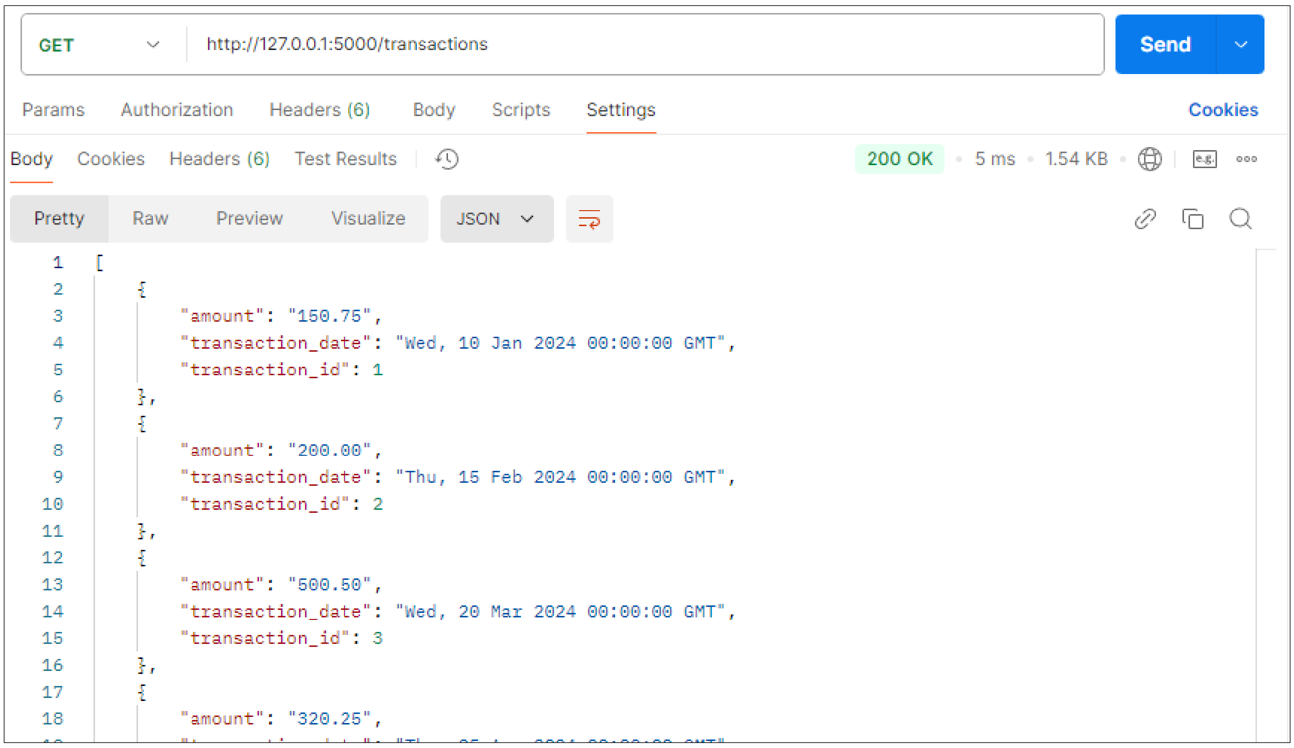


http://127.0.0.1:5000/accounts

200 OK • 5 ms • 2.05 KB

```
[
    {
        "account_id": 1,
        "account_number": "ACC10001",
        "balance": "5000.00",
        "customer_id": 1
    },
    {
        "account_id": 2,
        "account_number": "ACC10002",
        "balance": "3000.50",
        "customer_id": 2
    },
    {
        "account_id": 3,
        "account_number": "ACC10003",
        "balance": "7500.75",
        "customer_id": 3
    },
    {
        "account_id": 4,
        "account_number": "ACC10004",
        "balance": "12000.20",
        "customer_id": 4
    },
    {
```

Табличка transactions:

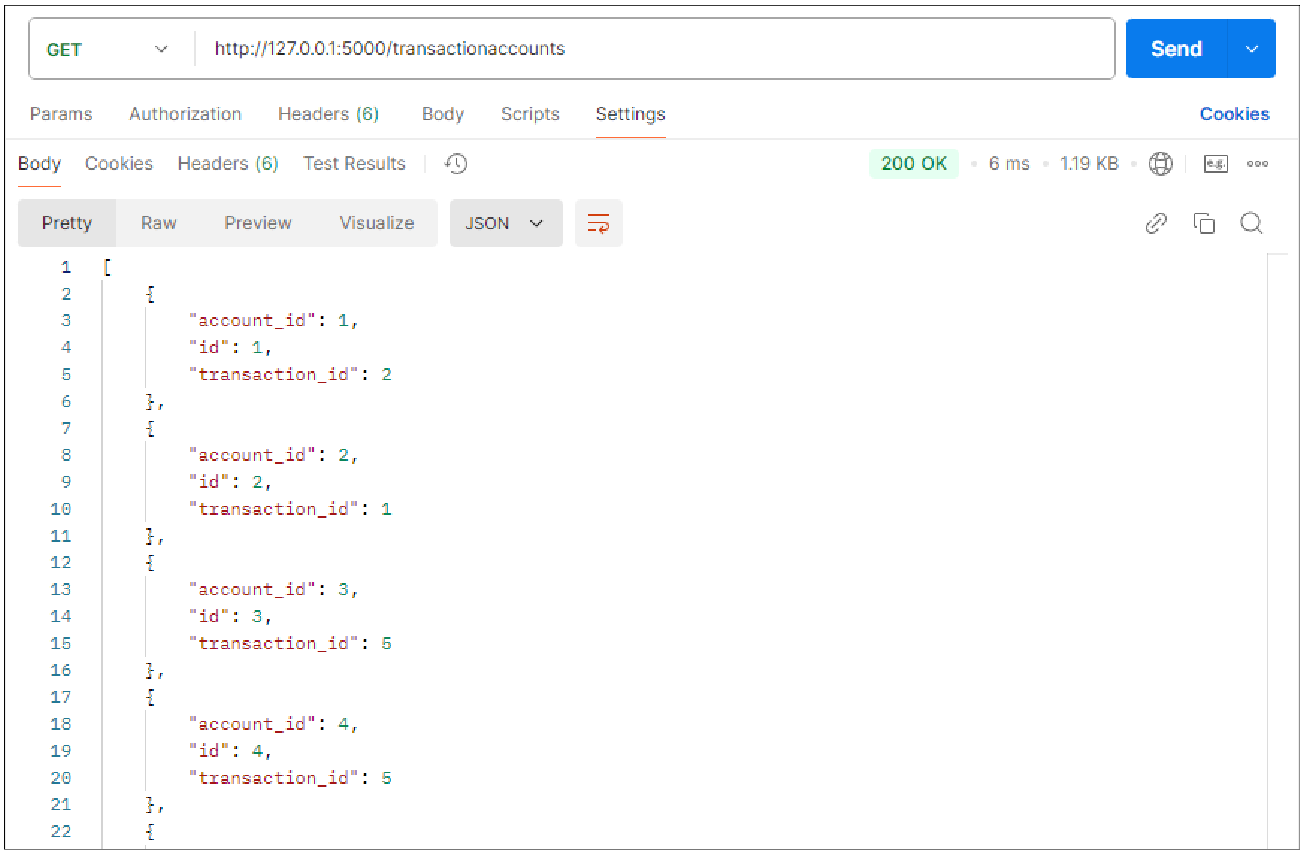


http://127.0.0.1:5000/transactions

200 OK • 5 ms • 1.54 KB

```
[
    {
        "amount": "150.75",
        "transaction_date": "Wed, 10 Jan 2024 00:00:00 GMT",
        "transaction_id": 1
    },
    {
        "amount": "200.00",
        "transaction_date": "Thu, 15 Feb 2024 00:00:00 GMT",
        "transaction_id": 2
    },
    {
        "amount": "500.50",
        "transaction_date": "Wed, 20 Mar 2024 00:00:00 GMT",
        "transaction_id": 3
    },
    {
        "amount": "320.25",
```

Стикувальна таблиця transactionaccounts:

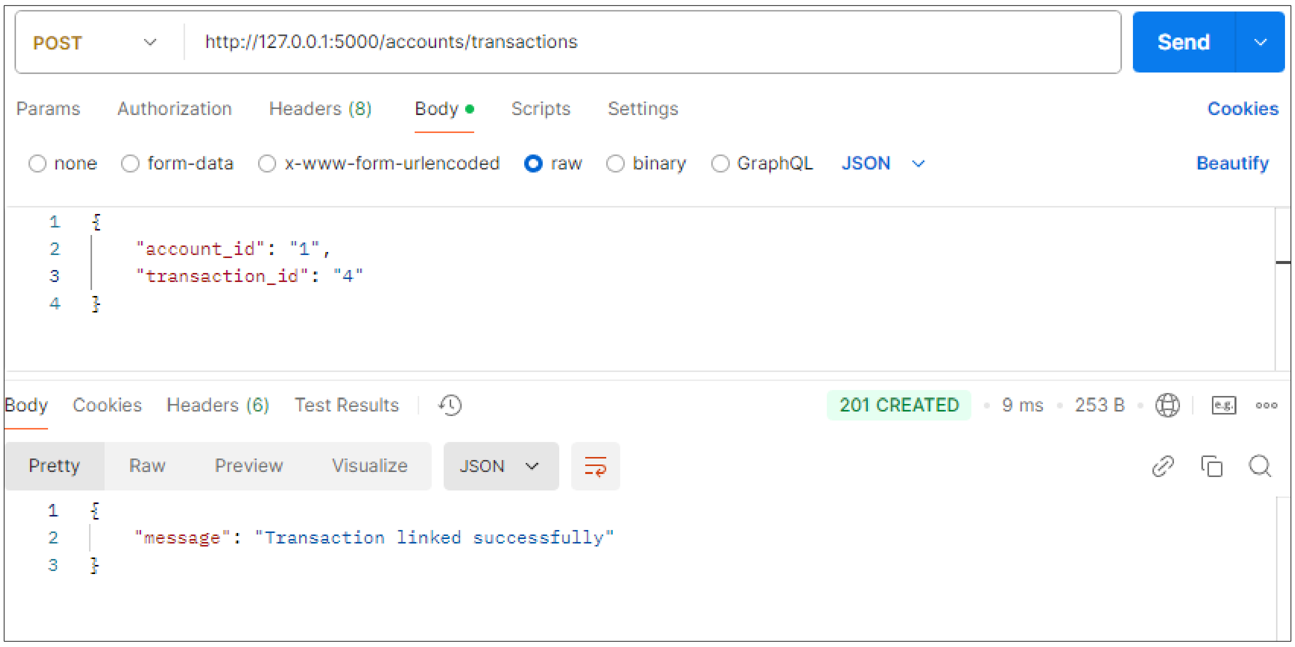


GET http://127.0.0.1:5000/transactionaccounts

200 OK • 6 ms • 1.19 KB

```
[
    {
        "account_id": 1,
        "id": 1,
        "transaction_id": 2
    },
    {
        "account_id": 2,
        "id": 2,
        "transaction_id": 1
    },
    {
        "account_id": 3,
        "id": 3,
        "transaction_id": 5
    },
    {
        "account_id": 4,
        "id": 4,
        "transaction_id": 5
    },
    {
```

створення зв’язку:



POST http://127.0.0.1:5000/accounts/transactions

Body: [✓ raw] JSON

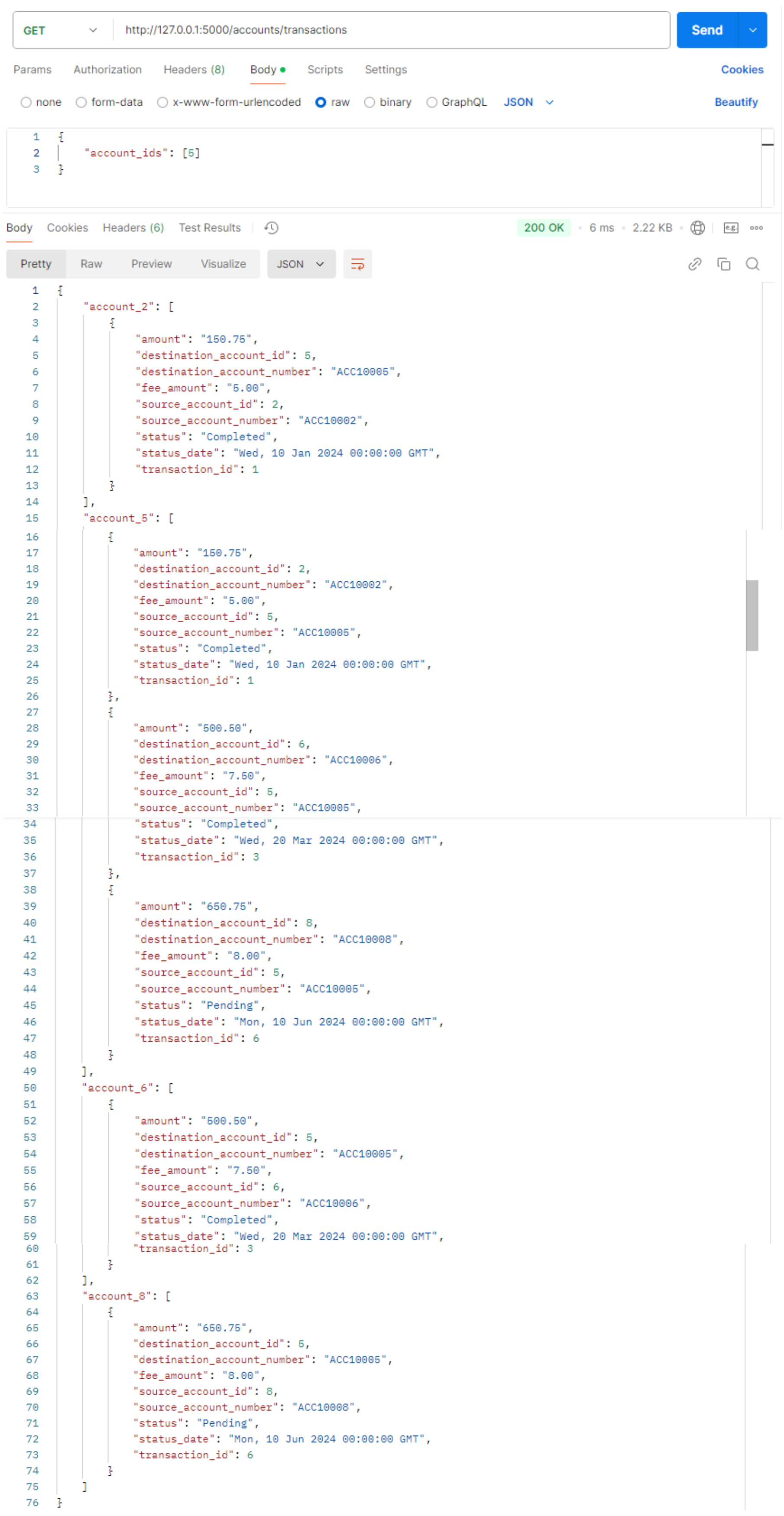
```
{
    "account_id": "1",
    "transaction_id": "4"
}
```

201 CREATED • 9 ms • 253 B

```
{
    "message": "Transaction linked successfully"
}
```

Сам зв’язок:

- виводяться усі транзакції пов’язані з аккаунтом 5, де він як отримувач так і відправник



GET http://127.0.0.1:5000/accounts/transactions

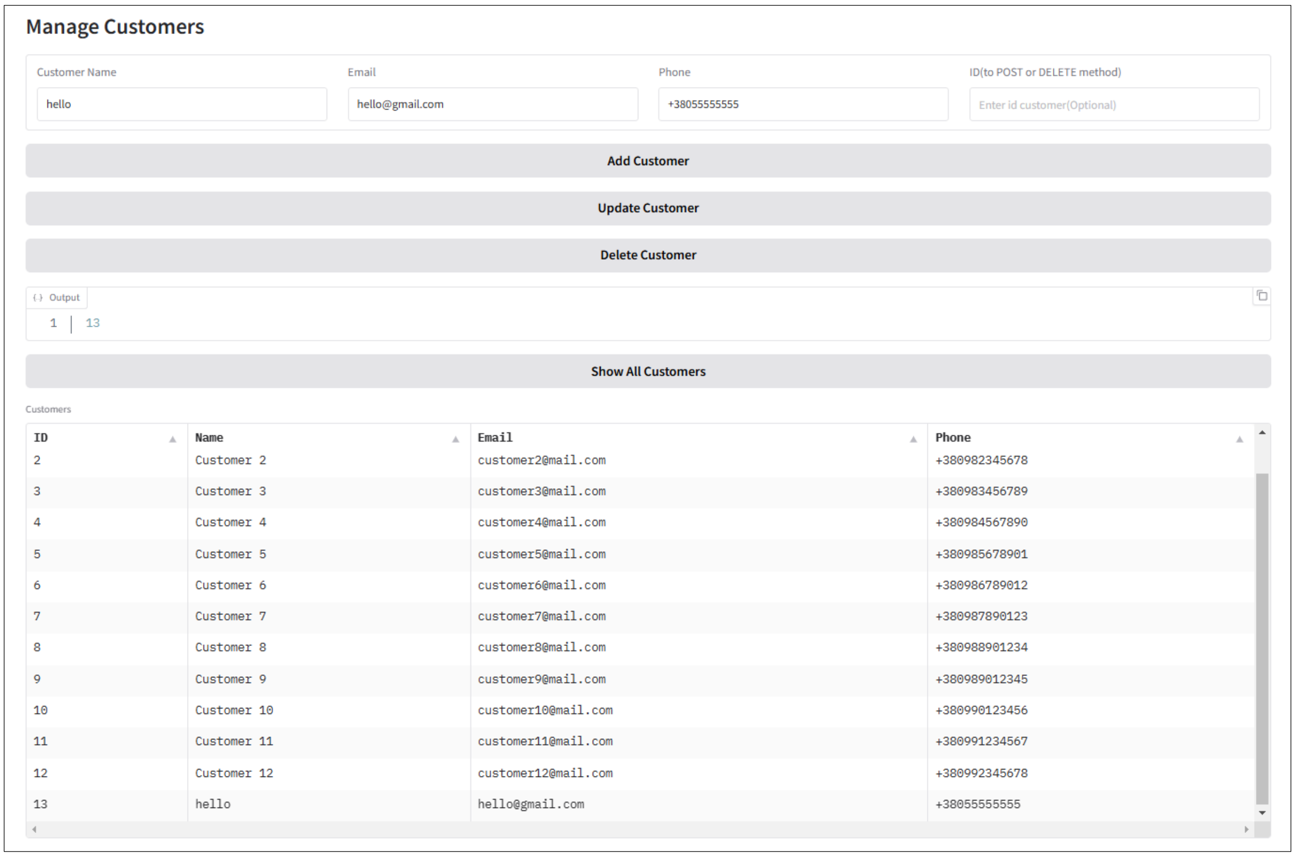
```
{
    "account_ids": [5]
}
```

```
{
    "account_2": [
        {
            "amount": "150.75",
            "destination_account_id": 5,
            "destination_account_number": "ACC10005",
            "fee_amount": "5.00",
            "source_account_id": 2,
            "source_account_number": "ACC10002",
            "status": "Completed",
            "status_date": "Wed, 10 Jan 2024 00:00:00 GMT",
            "transaction_id": 1
        }
    ],
    "account_5": [
        {
            "amount": "150.75",
            "destination_account_id": 2,
            "destination_account_number": "ACC10002",
            "fee_amount": "5.00",
            "source_account_id": 5,
            "source_account_number": "ACC10005",
            "status": "Completed",
            "status_date": "Wed, 10 Jan 2024 00:00:00 GMT",
            "transaction_id": 1
        },
        {
            "amount": "500.50",
            "destination_account_id": 6,
            "destination_account_number": "ACC10006",
            "fee_amount": "7.50",
            "source_account_id": 5,
            "source_account_number": "ACC10005",
            "status": "Completed",
            "status_date": "Wed, 20 Mar 2024 00:00:00 GMT",
            "transaction_id": 3
        },
        {
            "amount": "650.75",
            "destination_account_id": 8,
            "destination_account_number": "ACC10008",
            "fee_amount": "8.00",
            "source_account_id": 5,
            "source_account_number": "ACC10005",
            "status": "Pending",
            "status_date": "Mon, 10 Jun 2024 00:00:00 GMT",
            "transaction_id": 6
        }
    ],
    "account_6": [
        {
            "amount": "500.50",
            "destination_account_id": 5,
            "destination_account_number": "ACC10005",
            "fee_amount": "7.50",
            "source_account_id": 6,
            "source_account_number": "ACC10006",
            "status": "Completed",
            "status_date": "Wed, 20 Mar 2024 00:00:00 GMT",
            "transaction_id": 3
        }
    ],
    "account_8": [
        {
            "amount": "650.75",
            "destination_account_id": 5,
            "destination_account_number": "ACC10005",
            "fee_amount": "8.00",
            "source_account_id": 8,
            "source_account_number": "ACC10008",
            "status": "Pending",
            "status_date": "Mon, 10 Jun 2024 00:00:00 GMT",
            "transaction_id": 6
        }
    ]
}
```

Результати на UI

Методи POST і GET:

- Вставка в таблицю customers і перевірка одразу результату



Customer Name	Email	Phone	ID(to POST or DELETE method)
hello	hello@gmail.com	+38055555555	Enter id customer(Optional)

Output

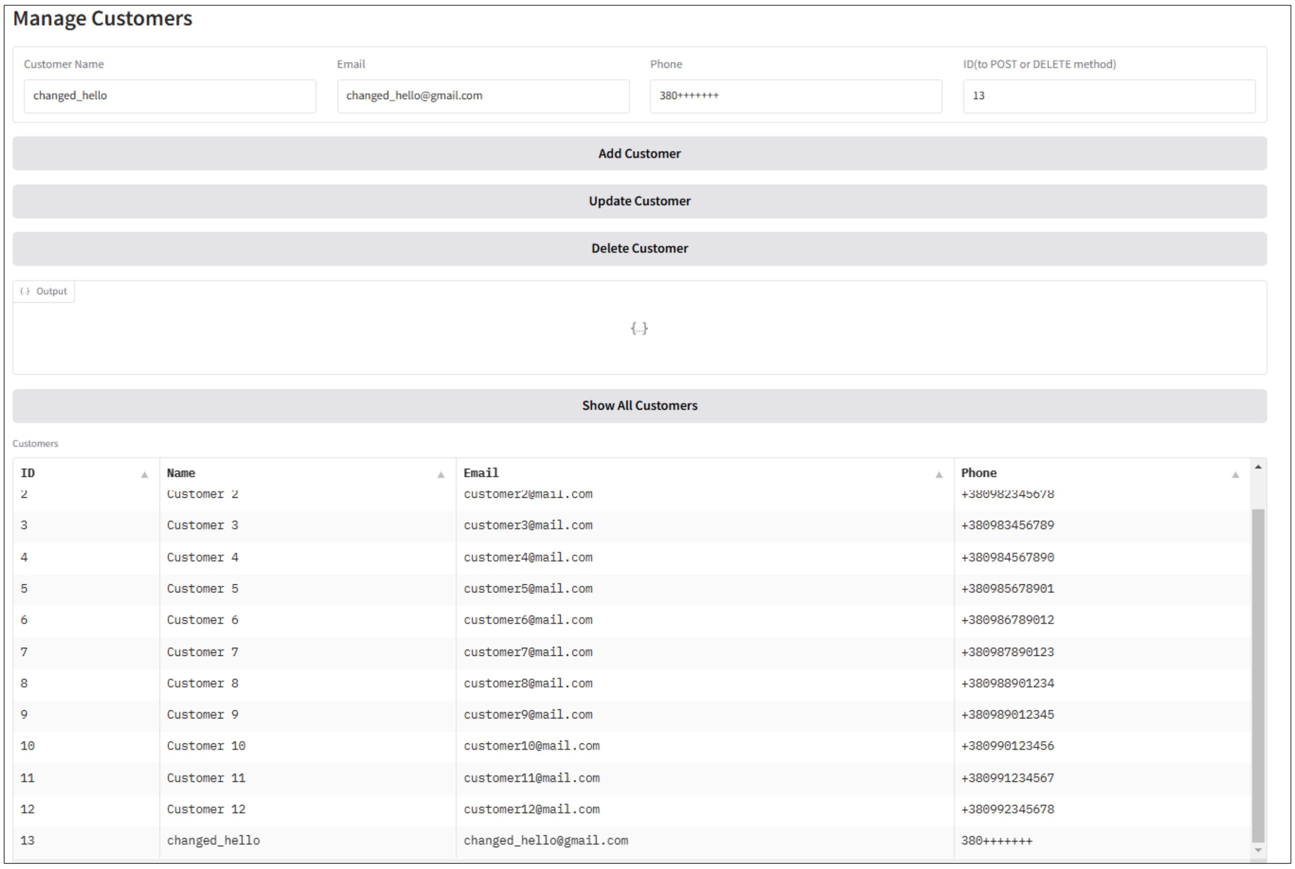
```
13
```

Customers

ID	Name	Email	Phone
2	Customer 2	customer2@mail.com	+380982345678
3	Customer 3	customer3@mail.com	+380983456789
4	Customer 4	customer4@mail.com	+380984567890
5	Customer 5	customer5@mail.com	+380985678901
6	Customer 6	customer6@mail.com	+380986789012
7	Customer 7	customer7@mail.com	+380987890123
8	Customer 8	customer8@mail.com	+380988901234
9	Customer 9	customer9@mail.com	+380989012345
10	Customer 10	customer10@mail.com	+380990123456
11	Customer 11	customer11@mail.com	+380991234567
12	Customer 12	customer12@mail.com	+380992345678
13	hello	hello@gmail.com	+38055555555

Методи PUT і GET:

- Оновлення значення в таблиці customers і перевірка одразу результату



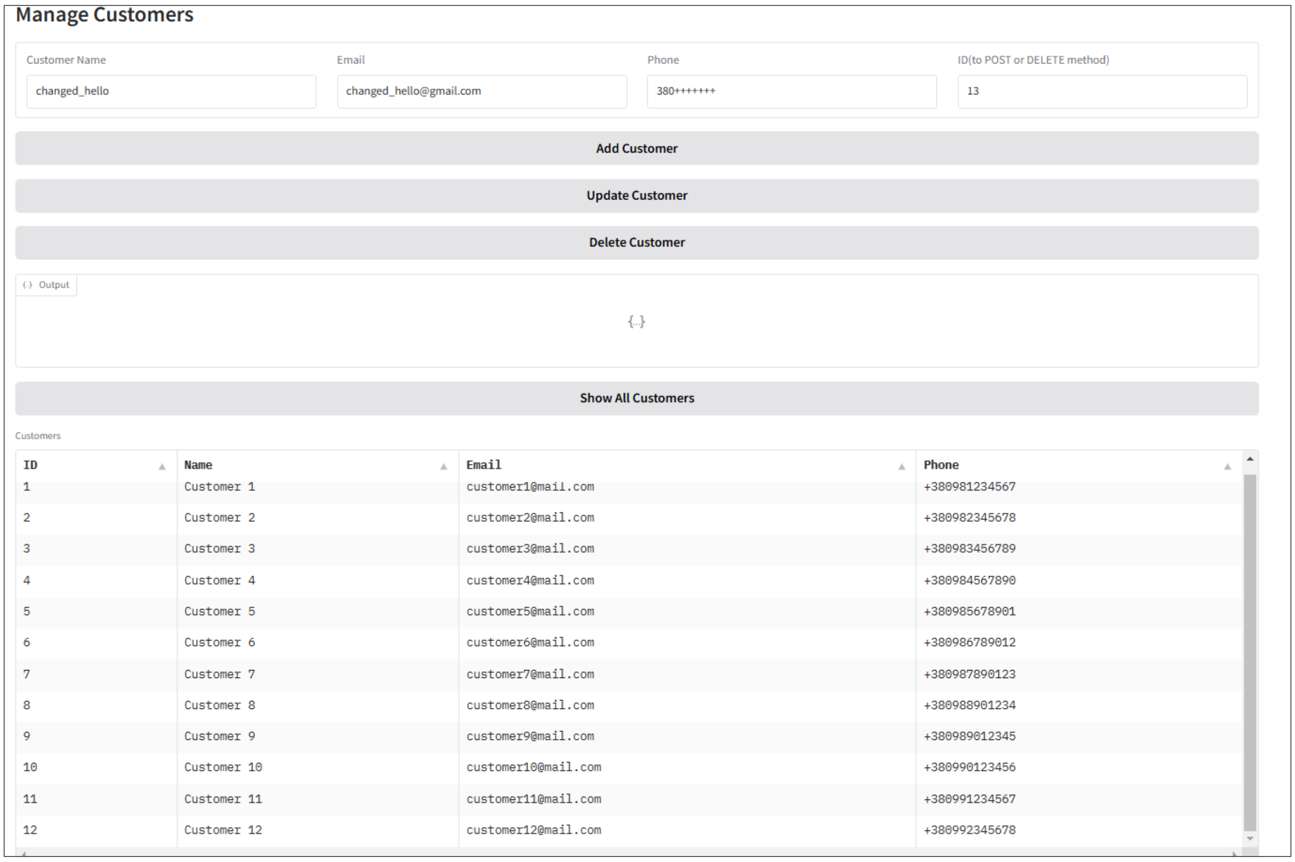
Manage Customers

Customer Name	Email	Phone	ID(to POST or DELETE method)
changed_hello	changed_hello@gmail.com	380+++++++	13

ID	Name	Email	Phone
2	Customer 2	customer2@mail.com	+380982345678
3	Customer 3	customer3@mail.com	+380983456789
4	Customer 4	customer4@mail.com	+380984567890
5	Customer 5	customer5@mail.com	+380985678901
6	Customer 6	customer6@mail.com	+380986789012
7	Customer 7	customer7@mail.com	+380987890123
8	Customer 8	customer8@mail.com	+380988901234
9	Customer 9	customer9@mail.com	+380989012345
10	Customer 10	customer10@mail.com	+380990123456
11	Customer 11	customer11@mail.com	+380991234567
12	Customer 12	customer12@mail.com	+380992345678
13	changed_hello	changed_hello@gmail.com	380+++++++

Методи DELETE і GET:

- Видалення значення в таблиці customers і перевірка одразу результату

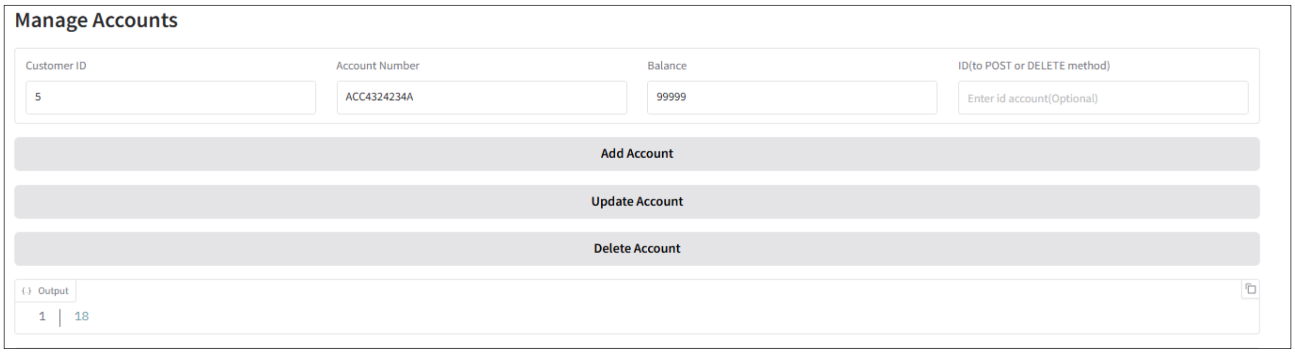


Manage Customers

Customer Name	Email	Phone	ID(to POST or DELETE method)
changed_hello	changed_hello@gmail.com	380+++++++	13

ID	Name	Email	Phone
1	Customer 1	customer1@mail.com	+380981234567
2	Customer 2	customer2@mail.com	+380982345678
3	Customer 3	customer3@mail.com	+380983456789
4	Customer 4	customer4@mail.com	+380984567890
5	Customer 5	customer5@mail.com	+380985678901
6	Customer 6	customer6@mail.com	+380986789012
7	Customer 7	customer7@mail.com	+380987890123
8	Customer 8	customer8@mail.com	+380988901234
9	Customer 9	customer9@mail.com	+380989012345
10	Customer 10	customer10@mail.com	+380990123456
11	Customer 11	customer11@mail.com	+380991234567
12	Customer 12	customer12@mail.com	+380992345678

Усі аналогічно реалізовані методи для таблиці accounts:



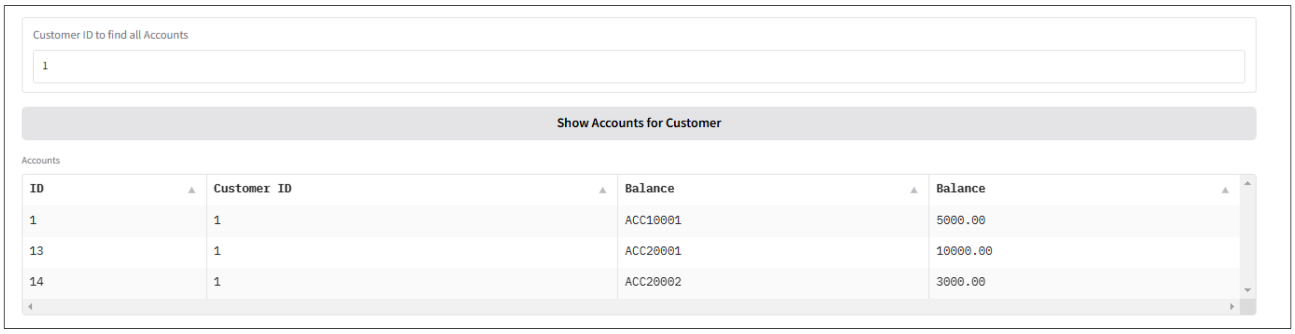
Manage Accounts

Customer ID	Account Number	Balance	ID(to POST or DELETE method)
5	ACC4324234A	99999	Enter id account(Optional)

```
18
```

Зв’язок 1:M:

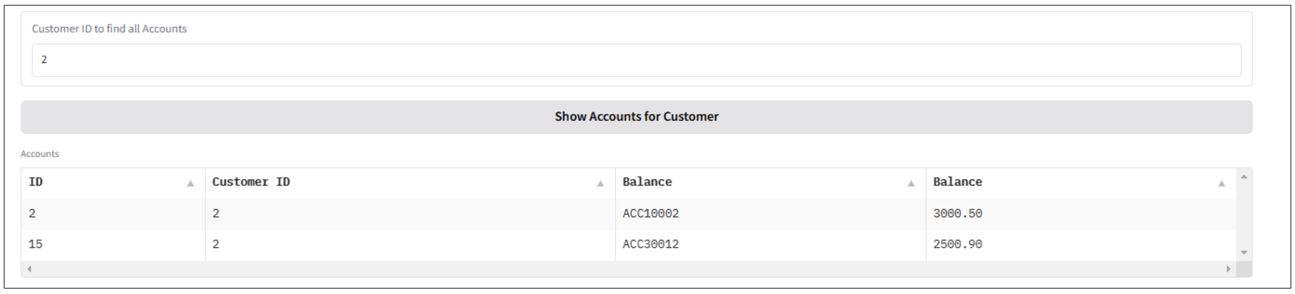
- показує усі аккаунти вибраного користувача:



Customer ID to find all Accounts: 1

Accounts

ID	Customer ID	Balance	Balance
1	1	ACC10001	5000.00
13	1	ACC20001	10000.00
14	1	ACC20002	3000.00

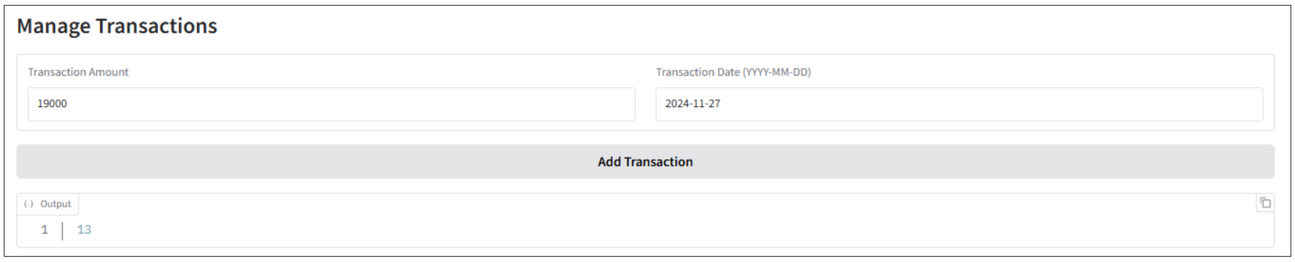


Customer ID to find all Accounts: 2

Accounts

ID	Customer ID	Balance	Balance
2	2	ACC10002	3000.50
15	2	ACC30012	2500.90

Метод POST для таблиці transactions:

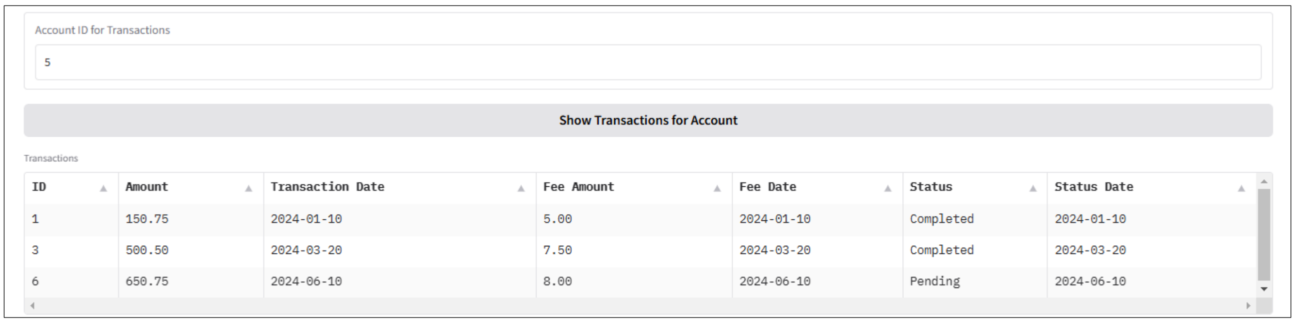


Manage Transactions

Transaction Amount	Transaction Date (YYYY-MM-DD)
19000	2024-11-27

```
13
```

Зв’язок 1:М:
- показує усі транзакції вибраного аккаунту:



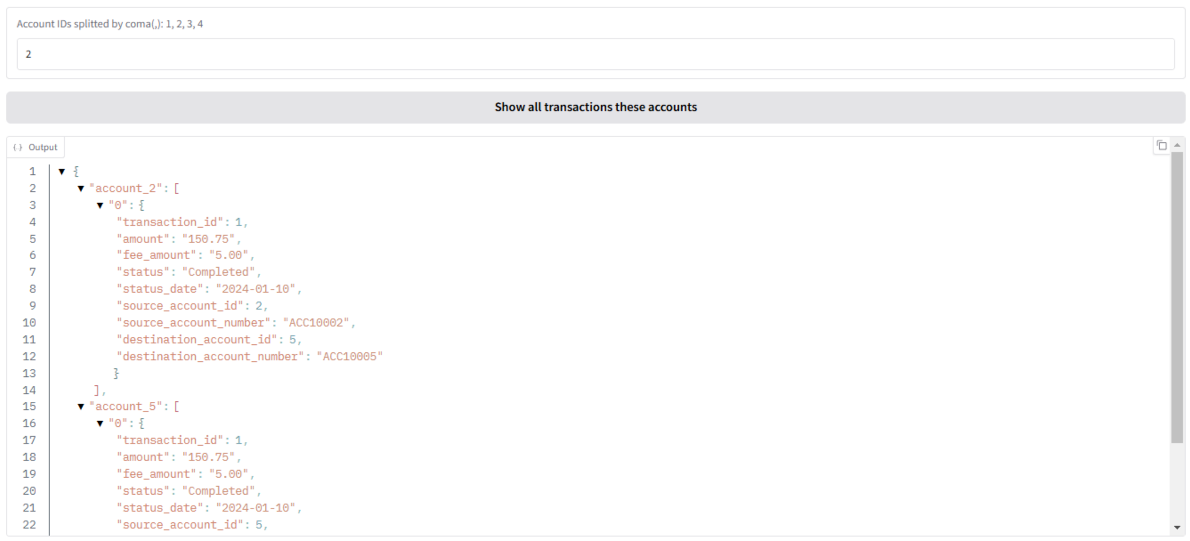
Account ID for Transactions: 5

Transactions

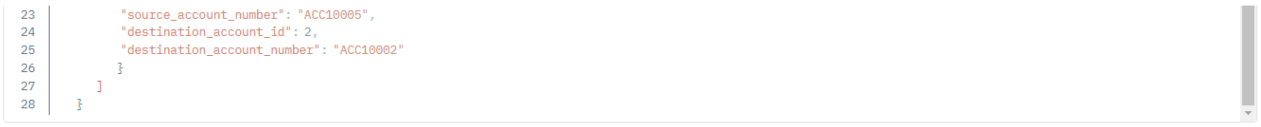
ID	Amount	Transaction Date	Fee Amount	Fee Date	Status	Status Date
1	150.75	2024-01-10	5.00	2024-01-10	Completed	2024-01-10
3	500.50	2024-03-20	7.50	2024-03-20	Completed	2024-03-20
6	650.75	2024-06-10	8.00	2024-06-10	Pending	2024-06-10

Зв’язок М:М:

- показує усі транзакції де був задіяний вказаний аккаунт, і там де він був отримувачем і там де був відправником


Account IDs splitted by coma(,): 1, 2, 3, 4
2
Show all transactions these accounts
Output

```
{
  "account_2": [
    "0": {
      "transaction_id": 1,
      "amount": "150.75",
      "fee_amount": "5.00",
      "status": "Completed",
      "status_date": "2024-01-10",
      "source_account_id": 2,
      "source_account_number": "ACC10002",
      "destination_account_id": 5,
      "destination_account_number": "ACC10005"
    }
  ],
  "account_5": [
    "0": {
      "transaction_id": 1,
      "amount": "150.75",
      "fee_amount": "5.00",
      "status": "Completed",
      "status_date": "2024-01-10",
      "source_account_id": 5,
```



```
      "source_account_number": "ACC10005",
      "destination_account_id": 2,
      "destination_account_number": "ACC10002"
    }
  ]
}
```

app.py

```
from flask import Flask
# from my_project.auth.route.user_route import init_routes
from my_project.auth.route.customer_route import init_customer_routes
from my_project.auth.route.account_route import init_account_routes
from my_project.auth.route.transaction_route import init_transaction_routes
from my_project.auth.route.transactionAccount_route import
init_transaction_account_routes

from config import config
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

# app.config.from_pyfile('config/app.yml')
db_config = config.load_db_config()

init_customer_routes(app)
init_account_routes(app)
init_transaction_routes(app)
init_transaction_account_routes(app)

if __name__ == "__main__":
    app.run(debug=True)
```

customer.dao

```
from my_project.auth.domain.domains import (
    Customer
)
from my_project.auth.dao.Base_dao import BaseDAO



class CustomerDAO(BaseDAO):
    def get_all_customers(self):
        self.cursor.execute("SELECT * FROM customers")
        result = self.cursor.fetchall()
        return [Customer(**row) for row in result]

    def get_customer_by_id(self, customer_id):
        self.cursor.execute("SELECT * FROM customers WHERE customer_id = %s",
(customer_id,))
        row = self.cursor.fetchone()
        if row:
            return Customer(**row)
        return None

    def create_customer(self, customer_name, email, phone):
        self.cursor.execute(
            "INSERT INTO customers (customer_name, email, phone) VALUES (%s,
%s, %s)",
            (customer_name, email, phone)
        )
        self.connection.commit()
        return self.cursor.lastrowid

    def update_customer(self, customer_id, customer_name, email, phone):
        query = "UPDATE customers SET customer_name = %s, email = %s, phone =
%s WHERE customer_id = %s"
        self.cursor.execute(query, (customer_name, email, phone, customer_id))
        self.connection.commit()

    def delete_customer(self, customer_id):
        self.cursor.execute("DELETE FROM customers WHERE customer_id = %s",
(customer_id,))
        self.connection.commit()
```

domains.py

```
from datetime import datetime
from typing import Optional

# Клас для таблиці customers
class Customer:
    def __init__(self, customer_id: int, customer_name: str, email: Optional[str], 
phone: Optional[str]):
        self.customer_id = customer_id
        self.customer_name = customer_name
        self.email = email
        self.phone = phone

    def to_dict(self):
        return {
            "customer_id": self.customer_id,
            "customer_name": self.customer_name,
            "email": self.email,
            "phone": self.phone
        }

# Клас для таблиці accounts
class Account:
    def __init__(self, account_id: int, customer_id: int, account_number: str, balance: 
float):
        self.account_id = account_id
        self.customer_id = customer_id
        self.account_number = account_number
        self.balance = balance

    def to_dict(self):
        return {
            "account_id": self.account_id,
            "customer_id": self.customer_id,
            "account_number": self.account_number,
            "balance": self.balance
        }

# Клас для таблиці transactions
class Transaction:
    def __init__(self, transaction_id, amount, transaction_date):
        self.transaction_id = transaction_id
        self.amount = amount
        self.transaction_date = transaction_date

    def to_dict(self):
        return {
            "transaction_id": self.transaction_id,
            "amount": self.amount,
            "transaction_date": self.transaction_date,
        }

# Клас для таблиці transactions
class Transaction_All_Info:
    def __init__(self, transaction_id, amount, transaction_date, fee_amount=None,
fee_date=None, status=None, status_date=None):
        self.transaction_id = transaction_id
        self.amount = amount
        self.transaction_date = transaction_date
        self.fee_amount = fee_amount
        self.fee_date = fee_date
        self.status = status
        self.status_date = status_date

    def to_dict(self):
        return {
            "transaction_id": self.transaction_id,
            "amount": self.amount,
            "transaction_date": self.transaction_date,
            "fee_amount": self.fee_amount,
            "fee_date": self.fee_date,
            "status": self.status,
            "status_date": self.status_date
        }

# Клас для таблиці TransactionsAccounts
class TransactionsAccounts:
    def __init__(self, id: int, account_id: int, transaction_id: int):
        self.id = id
        self.account_id = account_id
        self.transaction_id = transaction_id
    def to_dict(self):
        return {
            "id": self.id,
            "account_id": self.account_id,
            "transaction_id": self.transaction_id
        }

# Клас для таблиці payment_templates
class PaymentTemplate:
    def __init__(self, template_id: int, account_id: int, template_name: str,
template_details: Optional[str]):
        self.template_id = template_id
        self.account_id = account_id
        self.template_name = template_name
        self.template_details = template_details

    def to_dict(self):
        return {
            "template_id": self.template_id,
            "account_id": self.account_id,
            "template_name": self.template_name,
            "template_details": self.template_details
        }

# Клас для таблиці cards
class Card:
    def __init__(self, card_id: int, account_id: int, card_number: str, card_type:
Optional[str], expiry_date: Optional[datetime]):
        self.card_id = card_id
        self.account_id = account_id
        self.card_number = card_number
        self.card_type = card_type
        self.expiry_date = expiry_date

    def to_dict(self):
        return {
            "card_id": self.card_id,
            "account_id": self.account_id,
            "card_number": self.card_number,
            "card_type": self.card_type,
            "expiry_date": self.expiry_date.strftime('%Y-%m-%d') if self.expiry_date else None
        }

# Клас для таблиці needs
class Need:
    def __init__(self, need_id: int, transaction_id: int, service_name: str, payment_date: Optional[datetime], description: Optional[str], category: Optional[str], priority: Optional[int]):
        self.need_id = need_id
        self.transaction_id = transaction_id
        self.service_name = service_name
        self.payment_date = payment_date
        self.description = description
        self.category = category
        self.priority = priority

    def to_dict(self):
        return {
            "need_id": self.need_id,
            "transaction_id": self.transaction_id,
            "service_name": self.service_name,
            "payment_date": self.payment_date.strftime('%Y-%m-%d') if self.payment_date else None,
            "description": self.description,
            "category": self.category,
            "priority": self.priority
        }

# Клас для таблиці authorizations
class Authorization:
    def __init__(self, auth_id: int, account_id: int, login_time: Optional[datetime], logout_time: Optional[datetime], password: str):
        self.auth_id = auth_id
        self.account_id = account_id
        self.login_time = login_time
        self.logout_time = logout_time
        self.password = password

    def to_dict(self):
        return {
            "auth_id": self.auth_id,
            "account_id": self.account_id,
            "login_time": self.login_time.strftime('%Y-%m-%d %H:%M:%S') if
self.login_time else None,
            "logout_time": self.logout_time.strftime('%Y-%m-%d %H:%M:%S') if
self.logout_time else None,
            "password": self.password
        }

# Клас для таблиці fees
class Fee:
    def __init__(self, fee_id: int, transaction_id: int, fee_amount: float, fee_date:
Optional[datetime]):
        self.fee_id = fee_id
        self.transaction_id = transaction_id
        self.fee_amount = fee_amount
        self.fee_date = fee_date
    def to_dict(self):
        return {
            "fee_id": self.fee_id,
            "transaction_id": self.transaction_id,
            "fee_amount": self.fee_amount,
            "fee_date": self.fee_date.strftime('%Y-%m-%d') if self.fee_date else None
        }
# Клас для таблиці status_transactions
class StatusTransaction:
    def __init__(self, status_id: int, transaction_id: int, status: str, status_date:
Optional[datetime]):
        self.status_id = status_id
        self.transaction_id = transaction_id
        self.status = status
        self.status_date = status_date
    def to_dict(self):
        return {
            "status_id": self.status_id,
            "transaction_id": self.transaction_id,
            "status": self.status,
            "status_date": self.status_date.strftime('%Y-%m-%d') if self.status_date else
None
        }
```

customer_route.py

```
from flask import request, jsonify
from my_project.auth.service.user_service import CustomerService

customer_service = CustomerService()

def init_customer_routes(app):
    # Маршрути для клієнтів
    @app.route("/customers", methods=["GET"])
    def get_customers():
        customers = customer_service.get_all_customers()
        return jsonify(customers)

    @app.route("/customers/<int:customer_id>", methods=["GET"])
    def get_customer(customer_id):
        customer = customer_service.get_customer_by_id(customer_id)
        if customer:
            return jsonify(customer.to_dict())
        return jsonify({"error": "Customer not found"}), 404

    @app.route("/customers", methods=["POST"])
    def create_customer():
        data = request.get_json()
        customer_id = customer_service.create_customer(data['customer_name'],
data['email'], data['phone'])
        return jsonify({"id": customer_id}), 201

    @app.route("/customers/<int:customer_id>", methods=["PUT"])
    def update_customer(customer_id):
        data = request.get_json()
        customer_service.update_customer(customer_id, data['customer_name'],
data['email'], data['phone'])
        return jsonify({"message": "Customer updated successfully"})

    @app.route("/customers/<int:customer_id>", methods=["DELETE"])
    def delete_customer(customer_id):
        customer_service.delete_customer(customer_id)
        return jsonify({"message": "Customer deleted successfully"})
```

user_service

```
# from my_project.auth.dao.user_dao import CustomerDAO, AccountDAO,
TransactionDAO, TransactionAccountDAO, PaymentTemplateDAO
from flask import jsonify

from my_project.auth.dao.account_dao import AccountDAO
from my_project.auth.dao.customer_dao import CustomerDAO
from my_project.auth.dao.transaction_dao import TransactionDAO
from my_project.auth.dao.transactionAccount_dao import TransactionAccountDAO
from my_project.auth.dao.paymentTemplate_dao import PaymentTemplateDAO

class CustomerService:
    def __init__(self):
        self.customer_dao = CustomerDAO()

    def get_all_customers(self):
        customers = self.customer_dao.get_all_customers()
        return [customer.to_dict() for customer in customers]

    def get_customer_by_id(self, customer_id):
        return self.customer_dao.get_customer_by_id(customer_id)

    def create_customer(self, customer_name, email, phone):
        return self.customer_dao.create_customer(customer_name, email, phone)

    def update_customer(self, customer_id, customer_name, email, phone):
        self.customer_dao.update_customer(customer_id, customer_name, email,
phone)

    def delete_customer(self, customer_id):
        self.customer_dao.delete_customer(customer_id)


class AccountService:
    def __init__(self):
        self.account_dao = AccountDAO()

    def get_all_accounts(self):
        return self.account_dao.get_all_accounts()
    def get_account_by_id(self, account_id):
        return self.account_dao.get_account_by_id(account_id)

    def get_accounts_by_customer_id(self, customer_id):
        accounts = self.account_dao.get_accounts_by_customer_id(customer_id)
        if not isinstance(accounts, list):
            return [accounts]
        accounts = [account.to_dict() for account in accounts]
        return accounts

    def create_account(self, customer_id, account_number, balance):
        return self.account_dao.create_account(customer_id, account_number, balance)

    def update_account(self, account_id, customer_id, account_number, balance):
        self.account_dao.update_account(account_id, customer_id, account_number,
balance)

    def delete_account(self, account_id):
        self.account_dao.delete_account(account_id)

class TransactionService:
    def __init__(self):
        self.transaction_dao = TransactionDAO()

    def get_all_transactions(self):
        return self.transaction_dao.get_all_transactions()

    def get_transaction_by_id(self, transaction_id):
        return self.transaction_dao.get_transaction_by_id(transaction_id)

    def create_transaction(self, amount, transaction_date):
        return self.transaction_dao.create_transaction(amount, transaction_date)

    def update_transaction(self, transaction_id, amount, transaction_date):
        self.transaction_dao.update_transaction(transaction_id, amount,
transaction_date)

    def delete_transaction(self, transaction_id):
        self.transaction_dao.delete_transaction(transaction_id)
```

```
class TransactionAccountService:
    def __init__(self):
        self.transactions_accounts_dao = TransactionAccountDAO()

    def get_all_transactions_accounts(self):
        return self.transactions_accounts_dao.get_all_transactions_accounts()

    def get_transactions_account_by_id(self, account_id):
        transactions =
self.transactions_accounts_dao.get_transactions_account_by_id(account_id)
        if not isinstance(transactions, list):
            transactions = [transactions]
        transactions = [transaction.to_dict() for transaction in transactions]
        return transactions

    def create_transactions_account(self, account_id, transaction_id):
        return self.transactions_accounts_dao.create_transactions_account(account_id,
transaction_id)

    def delete_transactions_account(self, id):
        self.transactions_accounts_dao.delete_transactions_account(id)

    def get_transactions_account_by_ids(self, account_ids):
        return
self.transactions_accounts_dao.get_transactions_account_by_ids(account_ids)

class PaymentTemplateService:
    def __init__(self):
        self.payment_template_dao = PaymentTemplateDAO()

    def get_all_payment_templates(self):
        return self.payment_template_dao.get_all_payment_templates()

    def get_payment_template_by_id(self, template_id):
        return self.payment_template_dao.get_payment_template_by_id(template_id)

    def create_payment_template(self, account_id, template_name, template_details):
        return self.payment_template_dao.create_payment_template(account_id,
template_name, template_details)
    def delete_payment_template(self, template_id):
        self.payment_template_dao.delete_payment_template(template_id)
```

ui.py

```
import gradio as gr
from my_project.auth.service.user_service import CustomerService,
TransactionService, TransactionAccountService, \
    AccountService

customer_service = CustomerService()
account_service = AccountService()
transaction_service = TransactionService()
transaction_account_service = TransactionAccountService()

with gr.Blocks() as demo:
    gr.Markdown("# Manage Customers")
    with gr.Row():
        customer_name = gr.Textbox(label="Customer Name", placeholder="Enter
customer name")
        customer_email = gr.Textbox(label="Email", placeholder="Enter email")
        customer_phone = gr.Textbox(label="Phone", placeholder="Enter phone
number")
        customer_id = gr.Textbox(label='ID(to POST or DELETE method)',
placeholder="Enter id customer(Optional)")
    add_customer_btn = gr.Button("Add Customer")
    change_customer_btn = gr.Button("Update Customer")
    del_customer_btn = gr.Button('Delete Customer')
    customer_output = gr.JSON(label="Output")
    add_customer_btn.click(customer_service.create_customer,
inputs=[customer_name, customer_email, customer_phone],
                          outputs=customer_output)
    change_customer_btn.click(customer_service.update_customer,
                             inputs=[customer_id, customer_name, customer_email,
customer_phone],
                             outputs=customer_output)
    del_customer_btn.click(customer_service.delete_customer,
                          inputs=[customer_id],
                          outputs=customer_output)

    show_customers_btn = gr.Button("Show All Customers")
    customers_output = gr.DataFrame(headers=["ID", "Name", "Email", "Phone"],
label="Customers", interactive=False)
```

```
    show_customers_btn.click(
        lambda: [customer.values() for customer in
customer_service.get_all_customers()],
        outputs=customers_output
    )

    gr.Markdown("# Manage Accounts")
    with gr.Row():
        customer_id_for_account = gr.Textbox(label="Customer ID",
placeholder='Enter Customer id to connect')
        account_number = gr.Textbox(label="Account Number", placeholder="Enter
account number")
        account_balance = gr.Textbox(label="Balance", placeholder='Enter balance
sum')
        account_id = gr.Textbox(label="ID(to POST or DELETE method)",
placeholder='Enter id account(Optional)')

    add_account_btn = gr.Button("Add Account")
    update_account_btn = gr.Button("Update Account")
    delete_account_btn = gr.Button("Delete Account")
    account_output = gr.JSON(label="Output")
    add_account_btn.click(account_service.create_account,
                          inputs=[customer_id_for_account, account_number,
account_balance],
                          outputs=account_output)

    update_account_btn.click(account_service.update_account,
                             inputs=[account_id, customer_id_for_account, account_number,
account_balance],
                             outputs=account_output)

    delete_account_btn.click(account_service.delete_account,
                             inputs=[account_id],
                             outputs=account_output)

    with gr.Row():
        accounts_customer_id_input = gr.Textbox(label="Customer ID to find all
Accounts",
                                                placeholder='Enter customer id')
    show_accounts_btn = gr.Button("Show Accounts for Customer")
    accounts_output = gr.DataFrame(headers=["ID", "Customer ID", "Balance",
"Balance"], label="Accounts",
                                   interactive=False)

    show_accounts_btn.click(
        lambda accounts_customer_id_input: [account.values() for account in

account_service.get_accounts_by_customer_id(accounts_customer_id_input)],
        inputs=[accounts_customer_id_input],
        outputs=accounts_output
    )

    gr.Markdown("# Manage Transactions")
    with gr.Row():
        transaction_amount = gr.Textbox(label="Transaction Amount",
placeholder='Enter Amount of transaction')
        transaction_date = gr.Textbox(label="Transaction Date (YYYY-MM-DD)",
placeholder="Enter date")
    add_transaction_btn = gr.Button("Add Transaction")
    transaction_output = gr.JSON(label="Output")
    add_transaction_btn.click(transaction_service.create_transaction,
inputs=[transaction_amount, transaction_date],
                              outputs=transaction_output)

    with gr.Row():
        transactions_account_id_input = gr.Number(label="Account ID for
Transactions", precision=0)
    show_transactions_btn = gr.Button("Show Transactions for Account")
    transactions_output = gr.DataFrame(
        headers=["ID", "Amount", "Transaction Date", "Fee Amount", "Fee Date",
"Status", "Status Date"],
        label="Transactions", interactive=False)
    show_transactions_btn.click(
        lambda transactions_account_id_input: [transaction.values() for transaction in

transaction_account_service.get_transactions_account_by_id(
                                                transactions_account_id_input)],
        inputs=[transactions_account_id_input],
        outputs=transactions_output
    )
    with gr.Row():
        accounts_id_input = gr.Textbox(label="Account IDs splitted by coma(,): 1, 2, 3,
4")
    show_account_transactions_btn = gr.Button("Show all transactions these
accounts")
    account_transactions_output = gr.JSON(label="Output")
    show_account_transactions_btn.click(
        lambda accounts_id_input:
transaction_account_service.get_transactions_account_by_ids(str(accounts_id_input)
.split(',')),
        inputs=[accounts_id_input],
        outputs=account_transactions_output
    )

demo.launch(share=True)
```

Висновок

Було реалізовано бекенд на Flask+Python з підключенням до MySQL. Реалізовані методи GET, POST, PUT, DELETE. Також усі типи зв'язків: 1:1, 1:M, M:M. Для відображення результату біло написано візуальний інтерфейс за допомогою бібліотеки пайтон - gradio