Лабораторная работа 3: "Внедрение Dependency Injection и SQLAlchemy в Litestar"

ФИО: Ляхова Елизавета Олеговна
Группа: РИМ-150950

Цель работы: Освоить принципы Dependency Injection и интеграцию SQLAlchemy ORM в веб-приложении на базе фреймворка Litestar написав CRUD.

Ход работы

1. Создание репозитория:

```
from sqlalchemy import select, func
from sqlalchemy.ext.asyncio import AsyncSession
from api.models.user import User, UserCreate, UserUpdate
from typing import List, Optional
import bcrypt


class UserRepository:

    async def get_by_id(self, session: AsyncSession, user_id: int) -> Optional[User]:
        result = await session.execute(select(User).where(User.id == user_id))
        return result.scalar_one_or_none()

    async def get_by_email(self, session: AsyncSession, email: str) -> Optional[User]:
        result = await session.execute(select(User).where(User.email == email))
```

```
        return result.scalar_one_or_none()

            async   def   get_by_filter(self,   session:
AsyncSession, count: int = 10, page: int = 1, **kwargs)
-> List[User]:
        query = select(User)

        # Применяем фильтры
        for key, value in kwargs.items():
                if hasattr(User, key) and value is not
None:
                 query = query.where(getattr(User, key)
== value)

        # Применяем пагинацию
        offset = (page - 1) * count
        query = query.offset(offset).limit(count)

        result = await session.execute(query)
        return result.scalars().all()

          async   def   get_total_count(self,   session:
AsyncSession, **kwargs) -> int:
        query = select(func.count(User.id))

        for key, value in kwargs.items():
                if hasattr(User, key) and value is not
None:
                 query = query.where(getattr(User, key)
== value)

        result = await session.execute(query)
        return result.scalar()
```

```python
    def _hash_password(self, password: str) -> str:
        """Хеширование пароля"""
        salt = bcrypt.gensalt()
                                            hashed      =
bcrypt.hashpw(password.encode('utf-8'), salt)
        return hashed.decode('utf-8')

      def  _verify_password(self,  plain_password:  str,
hashed_password: str) -> bool:
        """Проверка пароля"""
                                                  return
bcrypt.checkpw(plain_password.encode('utf-8'),
hashed_password.encode('utf-8'))

       async  def  create(self,  session:  AsyncSession,
user_data: UserCreate) -> User:
        # Хеширование пароля
                                      hashed_password     =
self._hash_password(user_data.password)

        user = User(
            email=user_data.email,
            username=user_data.username,
            password_hash=hashed_password
        )

        session.add(user)
        await session.commit()
        await session.refresh(user)
        return user

       async  def  update(self,  session:  AsyncSession,
user_id: int, user_data: UserUpdate) -> User:
        user = await self.get_by_id(session, user_id)
```

```
        if not user:
            raise ValueError(f"User with ID {user_id}
not found")

                                    update_data     =
user_data.model_dump(exclude_unset=True)
        for field, value in update_data.items():
            if field == 'password' and value:
                        setattr(user, 'password_hash',
self._hash_password(value))
            elif hasattr(user, field) and value is not
None:
                setattr(user, field, value)

        await session.commit()
        await session.refresh(user)
        return user

       async def delete(self, session: AsyncSession,
user_id: int) -> None:
        user = await self.get_by_id(session, user_id)
        if user:
            await session.delete(user)
            await session.commit()
```

2.Создание сервисного слоя:

```
from api.repositories.user_repository import
UserRepository
from api.models.user import User, UserCreate,
UserUpdate
from typing import List, Optional
from sqlalchemy.ext.asyncio import AsyncSession
```

```python
class UserService:

    def __init__(self, user_repository: UserRepository,
db_session: AsyncSession):
        self.user_repository = user_repository
        self.db_session = db_session

    async def get_by_id(self, user_id: int) ->
Optional[User]:
        return await
self.user_repository.get_by_id(self.db_session,
user_id)

    async def get_by_email(self, email: str) ->
Optional[User]:
        return await
self.user_repository.get_by_email(self.db_session,
email)

    async def get_by_filter(self, count: int = 10,
page: int = 1, **kwargs) -> List[User]:
        return await
self.user_repository.get_by_filter(self.db_session,
count, page, **kwargs)

    async def get_total_count(self, **kwargs) -> int:
        return await
self.user_repository.get_total_count(self.db_session,
**kwargs)

    async def create(self, user_data: UserCreate) ->
User:
```

```
        # Проверка уникальности email
        existing_user = await
self.get_by_email(user_data.email)
        if existing_user:
            raise ValueError(f"User with email
{user_data.email} already exists")

        # Проверка уникальности username
        users_with_username = await
self.get_by_filter(username=user_data.username)
        if users_with_username:
            raise ValueError(f"User with username
{user_data.username} already exists")

        return await
self.user_repository.create(self.db_session, user_data)

    async def update(self, user_id: int, user_data:
UserUpdate) -> User:
        # Проверка существования пользователя
        existing_user = await self.get_by_id(user_id)
        if not existing_user:
            raise ValueError(f"User with ID {user_id}
not found")

        # Проверка уникальности email если он
обновляется
        if user_data.email and user_data.email !=
existing_user.email:
            user_with_email = await
self.get_by_email(user_data.email)
            if user_with_email:
                raise ValueError(f"User with email
{user_data.email} already exists")
```

```
        # Проверка уникальности username если он
обновляется
        if user_data.username and user_data.username !=
existing_user.username:
            users_with_username = await
self.get_by_filter(username=user_data.username)
            if users_with_username:
                raise ValueError(f"User with username
{user_data.username} already exists")

        return await
self.user_repository.update(self.db_session, user_id,
user_data)

    async def delete(self, user_id: int) -> None:
        await
self.user_repository.delete(self.db_session, user_id)
```

3.Создание контроллера:

```
from litestar import Controller, get, post, put, delete
from litestar.params import Parameter
from litestar.di import Provide
from litestar.exceptions import NotFoundException,
ValidationException
from litestar.status_codes import HTTP_201_CREATED
from typing import List, Dict, Any
from api.services.user_service import UserService
from api.models.user import UserResponse, UserCreate,
UserUpdate


class UserController(Controller):
    path = "/users"
```

```python
    @get("/{user_id:int}")
    async def get_user_by_id(
        self,
        user_service: UserService,
        user_id: int = Parameter(gt=0),
    ) -> UserResponse:
        """Получить пользователя по ID"""
        user = await user_service.get_by_id(user_id)
        if not user:
            raise NotFoundException(detail=f"User with
ID {user_id} not found")
        return UserResponse.model_validate(user)

    @get()
    async def get_all_users(
        self,
        user_service: UserService,
        count: int = Parameter(gt=0, le=100,
default=10),
        page: int = Parameter(gt=0, default=1)
    ) -> Dict[str, Any]:
        """Получить всех пользователей с пагинацией"""
        users = await
user_service.get_by_filter(count=count, page=page)
        total_count = await
user_service.get_total_count()

        return {
            "users": [UserResponse.model_validate(user)
for user in users],
            "total_count": total_count,
            "page": page,
            "count": count,
```

```
            "total_pages": (total_count + count - 1) //
count
        }

    @post(status_code=HTTP_201_CREATED)
    async def create_user(
        self,
        user_service: UserService,
        data: UserCreate,  # Изменено с user_data на
data
    ) -> UserResponse:
        """Создать нового пользователя"""
        try:
            user = await user_service.create(data)  #
Изменено здесь
            return UserResponse.model_validate(user)
        except ValueError as e:
            raise ValidationException(detail=str(e))

    @delete("/{user_id:int}")
    async def delete_user(
        self,
        user_service: UserService,
        user_id: int,
    ) -> None:
        """Удалить пользователя по ID"""
        try:
            await user_service.delete(user_id)
        except ValueError as e:
            raise NotFoundException(detail=str(e))

    @put("/{user_id:int}")
    async def update_user(
        self,
```

```
        user_service: UserService,
        user_id: int,
        data: UserUpdate,  # Изменено с user_data на
data
    ) -> UserResponse:
        """Обновить пользователя"""
        try:
            user = await user_service.update(user_id,
data)  # Изменено здесь
            return UserResponse.model_validate(user)
        except ValueError as e:
            raise NotFoundException(detail=str(e))
```

4. Создание главного приложения:

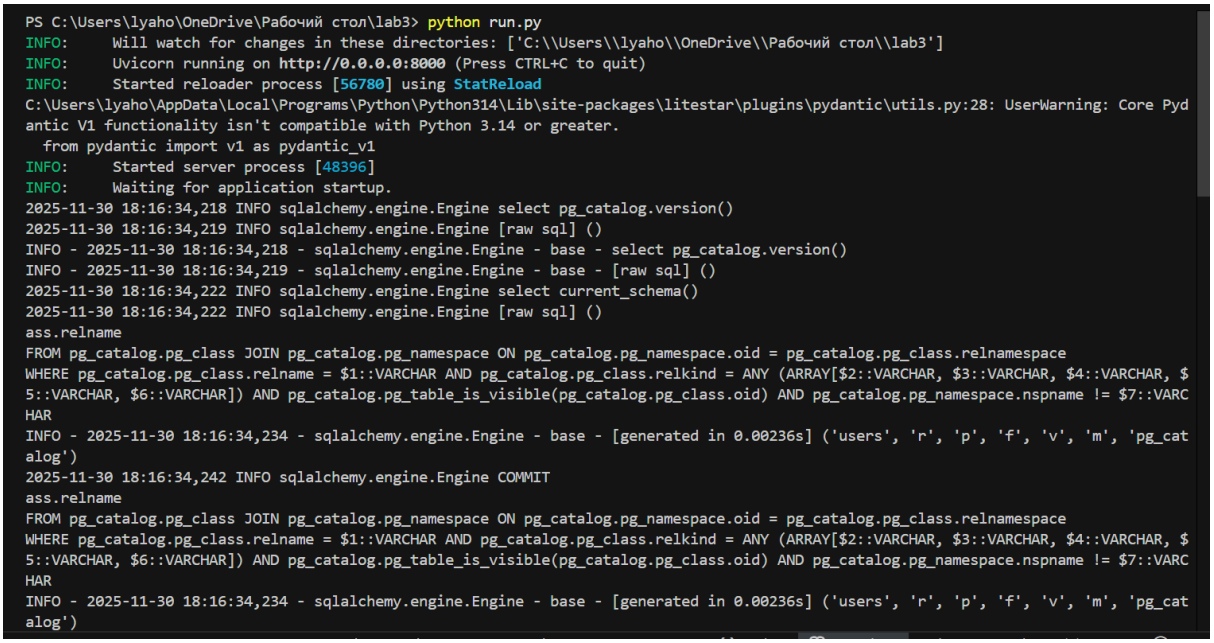
```
import os
from litestar import Litestar
from litestar.di import Provide
from sqlalchemy.ext.asyncio import create_async_engine,
AsyncSession, async_sessionmaker
from typing import AsyncGenerator
from api.controllers.user_controller import
UserController
from api.repositories.user_repository import
UserRepository
from api.services.user_service import UserService
from api.models.user import Base

# Настройка базы данных PostgreSQL
DATABASE_URL = os.getenv(
    "DATABASE_URL",
```

```
"postgresql+asyncpg://litestar_user:litestar_password@l
ocalhost:5432/litestar_lab3"
)

# Создание движка SQLAlchemy
engine = create_async_engine(
    DATABASE_URL,
    echo=True,
    pool_pre_ping=True,
    pool_recycle=300,
)

# Фабрика сессий
async_session_factory = async_sessionmaker(
    engine,
    class_=AsyncSession,
    expire_on_commit=False,
    autoflush=False
)

async def provide_db_session() ->
AsyncGenerator[AsyncSession, None]:
    """Провайдер сессии базы данных"""
    async with async_session_factory() as session:
        try:
            yield session
            await session.commit()
        except Exception:
            await session.rollback()
            raise
        finally:
            await session.close()
```

```python
async def provide_user_repository(db_session:
AsyncSession) -> UserRepository:
    """Провайдер репозитория пользователей"""
    return UserRepository()

async def provide_user_service(
    user_repository: UserRepository,
    db_session: AsyncSession
) -> UserService:
    """Провайдер сервиса пользователей"""
    return UserService(user_repository, db_session)

async def on_startup():
    """Создание таблиц при запуске приложения"""
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)
    print("✅ Database tables created successfully")

async def on_shutdown():
    """Закрытие соединений при завершении приложения"""
    await engine.dispose()
    print("✅ Database connections closed")

app = Litestar(
    route_handlers=[UserController],
    dependencies={
        "db_session": Provide(provide_db_session),
        "user_repository":
Provide(provide_user_repository),
        "user_service": Provide(provide_user_service),
    },
    on_startup=[on_startup],
    on_shutdown=[on_shutdown]
)
```

```python
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(
        "app.main:app",   # Изменено для поддержки
reload
        host="0.0.0.0",
        port=8000,
        reload=True
    )
```

Скриншоты, что приложение успешно запустилось:



```
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::VARCHAR, $4::VARCHAR, $
5::VARCHAR, $6::VARCHAR]) AND pg_catalog.pg_table_is_visible(pg_catalog.pg_class.oid) AND pg_catalog.pg_namespace.nspname != $7::VARC
HAR
INFO - 2025-11-30 18:16:34,234 - sqlalchemy.engine.Engine - base - [generated in 0.00236s] ('users', 'r', 'p', 'f', 'v', 'm', 'pg_cat
alog')
ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::VARCHAR, $4::VARCHAR, $
5::VARCHAR, $6::VARCHAR]) AND pg_catalog.pg_table_is_visible(pg_catalog.pg_class.oid) AND pg_catalog.pg_namespace.nspname != $7::VARC
HAR
ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::VARCHAR, $4::VARCHAR, $
5::VARCHAR, $6::VARCHAR]) AND pg_catalog.pg_table_is_visible(pg_catalog.pg_class.oid) Aass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
ass.relname
ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::VARCHAR, $4::VARCHAR, $
5::VARCHAR, $6::VARCHAR]) AND pg_catalog.pg_table_is_visible(pg_catalog.pg_class.oid) Aass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::namespace
```

```
ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
ass.relname
ass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::VARCHAR, $4::VARCHAR, $
5::VARCHAR, $6::VARCHAR]) AND pg_catalog.pg_table_is_visible(pg_catalog.pg_class.oid) Aass.relname
FROM pg_catalog.pg_class JOIN pg_catalog.pg_namespace ON pg_catalog.pg_namespace.oid = pg_catalog.pg_class.relnamespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::namespace
WHERE pg_catalog.pg_class.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::WHERE pg_catalog.pg_cla
ss.relname = $1::VARCHAR AND pg_catalog.pg_class.relkind = ANY (ARRAY[$2::VARCHAR, $3::VARCHAR, $4::VARCHAR, $5::VARCHAR, $6::VARCHAR
]) AND pg_catalog.pg_table_is_visible(pg_catalog.pg_class.oid) AVARCHAR, $4::VARCHAR, $5::VARCHAR, $6::VARCHAR]) AND pg_catalog.pg_ta
ble_is_visible(pg_catalog.pg_class.oid) AND pg_catalog.pg_namespace.nspname != $7::VARCHAR
INFO - 2025-11-30 18:16:34,234 - sqlalchemy.engine.Engine - base - [generated in 0.00236s] ('users', 'r', 'p', 'f', 'v', 'm', 'pg_cat
alog')
2025-11-30 18:16:34,242 INFO sqlalchemy.engine.Engine COMMIT
INFO - 2025-11-30 18:16:34,234 - sqlalchemy.engine.Engine - base - [generated in 0.00236s] ('users', 'r', 'p', 'f', 'v', 'm', 'pg_cat
alog')
2025-11-30 18:16:34,242 INFO sqlalchemy.engine.Engine COMMIT
 'f', 'v', 'm', 'pg_catalog')
2025-11-30 18:16:34,242 INFO sqlalchemy.engine.Engine COMMIT
2025-11-30 18:16:34,242 INFO sqlalchemy.engine.Engine COMMIT
INFO - 2025-11-30 18:16:34,242 - sqlalchemy.engine.Engine - base - COMMIT
✅ Database tables created successfully
INFO:     Application startup complete.
```

Тестирование приложения:

1) Создание пользователя

```
PS C:\Users\lyaho> Invoke-RestMethod -Uri "http://localhost:8000/users" -Method GET


users       : {@{id=1; email=test@example.com;
              username=testuser;
              created_at=2025-11-30T13:17:35.904246Z;
              updated_at=2025-11-30T13:17:35.904246Z}}
total_count : 1
page        : 1
count       : 10
total_pages : 1
```

2) Получение пользователя по ID

```
PS C:\Users\lyaho> Invoke-RestMethod -Uri "http://localhost:8000/users/1" -Method GET


id         : 1
email      : test@example.com
username   : testuser
created_at : 2025-11-30T13:17:35.904246Z
updated_at : 2025-11-30T13:17:35.904246Z
```

3) Обновление пользователя

```
PS C:\Users\lyaho> $updateBody = @{
>>     username = "updateduser"
>>     email = "updated@example.com"
>> }
PS C:\Users\lyaho>
PS C:\Users\lyaho> Invoke-RestMethod -Uri "http://localhost:8000/users/1" -Method PUT -Body ($updateBody | ConvertTo-Json) -ContentTy
pe "application/json"


id         : 1
email      : updated@example.com
username   : updateduser
created_at : 2025-11-30T13:17:35.904246Z
updated_at : 2025-11-30T13:19:50.196612Z
```

4) Создание нескольких пользователей

```
PS C:\Users\lyaho> $users = @(
>>     @{email = "user2@example.com"; username = "user2"; password = "pass123"},
>>     @{email = "user3@example.com"; username = "user3"; password = "pass123"},
>>     @{email = "admin@example.com"; username = "admin"; password = "admin123"}
>> )
PS C:\Users\lyaho>
PS C:\Users\lyaho> foreach ($user in $users) {
>>     Invoke-RestMethod -Uri "http://localhost:8000/users" -Method POST -Body ($user | ConvertTo-Json) -ContentType "application/jso
n"
>> }


id         : 2
email      : user2@example.com
username   : user2
created_at : 2025-11-30T13:20:00.058637Z
updated_at : 2025-11-30T13:20:00.058637Z

id         : 3
email      : user3@example.com
username   : user3
created_at : 2025-11-30T13:20:00.425297Z
updated_at : 2025-11-30T13:20:00.425297Z

id         : 4
email      : admin@example.com
username   : admin
created_at : 2025-11-30T13:20:00.778659Z
updated_at : 2025-11-30T13:20:00.778659Z
```

5) Проверка пагинации

```
PS C:\Users\lyaho> # С count=2 и page=1
PS C:\Users\lyaho> Invoke-RestMethod -Uri "http://localhost:8000/users?count=2&page=1" -Method GET


users       : {@{id=1; email=updated@example.com;
              username=updateduser;
              created_at=2025-11-30T13:17:35.904246Z;
              updated_at=2025-11-30T13:19:50.196612Z},
              @{id=2; email=user2@example.com;
              username=user2;
              created_at=2025-11-30T13:20:00.058637Z;
              updated_at=2025-11-30T13:20:00.058637Z}}
total_count : 4
page        : 1
count       : 2
total_pages : 2



PS C:\Users\lyaho>
PS C:\Users\lyaho> # С count=2 и page=2
PS C:\Users\lyaho> Invoke-RestMethod -Uri "http://localhost:8000/users?count=2&page=2" -Method GET


users       : {@{id=3; email=user3@example.com;
              username=user3;
              created_at=2025-11-30T13:20:00.425297Z;
              updated_at=2025-11-30T13:20:00.425297Z},
              @{id=4; email=admin@example.com;
              username=admin;
              created_at=2025-11-30T13:20:00.778659Z;
              updated_at=2025-11-30T13:20:00.778659Z}}
total_count : 4
page        : 2
count       : 2
total_pages : 2
```

6)Удаление пользователя

```
PS C:\Users\lyaho> Invoke-RestMethod -Uri "http://localhost:8000/users/1" -Method DELETE

PS C:\Users\lyaho>
```

Проверка базы данных

```
OOOO@MSI MINGW64 ~
$ docker exec -it litestar_lab3_db psql -U litestar_user -d litestar_lab3 -c "SELECT * FROM users"
 id |       email       | username |                        password_hash                         |         created_at         |         updated_at
----+-------------------+----------+--------------------------------------------------------------+----------------------------+----------------------------
  2 | user2@example.com | user2    | $2b$12$7eLh6lhEt9IIgc7BwQKoDOZfaJ9Vw1emeRkdRI6.t7jLrjOx/nHwa | 2025-11-30 13:20:00.058637+00 | 2025-11-30 13:20:00.058637+00
  3 | user3@example.com | user3    | $2b$12$KI.sphc7TITD4sOIOGF3YeLM8LgkE5hRRMEF1yD3Cg78dD1BaGH2G | 2025-11-30 13:20:00.425297+00 | 2025-11-30 13:20:00.425297+00
  4 | admin@example.com | admin    | $2b$12$8NbEzRVtNI6qQvaUGOD3b.UUFXxiUeHr4OAZSkQkgGWhJHAiKuSM6 | 2025-11-30 13:20:00.778659+00 | 2025-11-30 13:20:00.778659+00
(3 rows)
```

Задача со звездочкой:

```
async def get_count(self, session: AsyncSession, **kwargs) -> int:
    query = select(func.count(User.id))

    for key, value in kwargs.items():
        if hasattr(User, key) and value is not None:
            query = query.where(getattr(User, key) == value)

    result = await session.execute(query)
    return result.scalar()
```

Ответы на вопросы:

1. Принцип Dependency Injection (DI)

Dependency Injection - это паттерн проектирования, при котором зависимости объекта не создаются внутри самого объекта, а передаются извне. Это решает проблему тесной связности компонентов и упрощает тестирование.

Преимущества:

- Упрощение тестирования (можно передавать mock-объекты)
- Снижение связанности компонентов

- Упрощение повторного использования кода
- Централизованное управление зависимостями

2. Обязанности слоев приложения

Repository - работа с данными, CRUD операции, абстракция над базой данных.

Service - бизнес-логика, валидация, преобразование данных.

Controller - обработка HTTP-запросов, маршрутизация, преобразование данных для клиента.

Если объединить репозиторий и контроллер: нарушится принцип единственной ответственности, усложнится тестирование, бизнес-логика смешается с представлением данных.

3. Жизненный цикл зависимости в Litestar

При HTTP-запросе:

1. Создается сессия БД через `provide_db_session()`
2. Создаются репозиторий и сервис с зависимостями
3. После обработки запроса сессия автоматически закрывается
4. Все зависимости уничтожаются после завершения запроса

4. async/await в приложении

async/await позволяют выполнять неблокирующие операции. При работе с БД это особенно важно, так как запросы к БД - I/O-bound операции. Асинхронность позволяет обрабатывать множество запросов одновременно, повышая производительность.

5. Передача session в UserRepository

Session передается как аргумент для:

- Контроля транзакций на уровне сервиса

- Возможности использования одной сессии для нескольких операций
- Упрощения тестирования

session.commit()/rollback() должен вызывать сервисный слой, так как он управляет бизнес-транзакциями.

6. Пагинация в get_by_filter

Пагинация (count и page) нужна для:

- Ограничения объема возвращаемых данных
- Улучшения производительности
- Удобства клиента (постраничная навигация)

7. Бизнес-логика для UserService

python

```python
async def create(self, session, user_data: UserCreate) -> User:
    # Проверка уникальности email
    existing_user = await self.get_by_email(session, user_data.email)
    if existing_user:
        raise ValueError("Email already exists")

    # Хеширование пароля
    hashed_password = self._hash_password(user_data.password)
    user_data.password = hashed_password

    # Отправка приветственного письма (асинхронно)
    await self._send_welcome_email(user_data.email)

    return await self.user_repository.create(session, user_data)
```

8. HTTP-статусы для эндпоинтов

GET /users/{id}

- 200 - пользователь найден
- 404 - пользователь не найден

POST /users

- 201 - пользователь создан
- 400 - неверные данные
- 409 - конфликт (email уже существует)

PUT /users/{id}

- 200 - обновление успешно
- 404 - пользователь не найден
- 400 - неверные данные

DELETE /users/{id}

- 204 - удаление успешно
- 404 - пользователь не найден

Выводы:

В ходе выполнения лабораторной работы №3 "Внедрение Dependency Injection и SQLAlchemy в Litestar" были успешно освоены принципы Dependency Injection и интеграция SQLAlchemy ORM в веб-приложении на базе фреймворка Litestar.

Была реализована полнофункциональная CRUD-система для управления пользователями с четким разделением на слои:
- Repository слой для работы с базой данных
- Service слой для бизнес-логики
- Controller слой для обработки HTTP-запросов

На практике применен принцип Dependency Injection, который позволил создать гибкую и тестируемую архитектуру приложения. Все зависимости (сессии базы данных, репозитории, сервисы) корректно инжектируются через провайдеры Litestar.

Интеграция SQLAlchemy ORM с PostgreSQL обеспечила надежную работу с данными, включая автоматическое создание таблиц, миграции и

выполнение сложных запросов. Использование асинхронных операций (async/await) значительно повысило производительность приложения при работе с базой данных.

Все запланированные endpoints (GET, POST, PUT, DELETE) работают корректно, что подтверждено тестированием через PowerShell. Приложение успешно создает, читает, обновляет и удаляет пользователей, возвращая соответствующие HTTP-статусы.

Дополнительно реализована пагинация для получения списка пользователей и подсчет общего количества записей.