Министерство Образования Российской Федерации
УрФУ
Образовательных информационных технологий

Лабораторная работа №3
Тема: «Внедрение Dependency Injection и SQLAlchemy в Litestar»

Выполнил:
Нарсеев А.В.

Екатеринбург, 2025

Цель работы

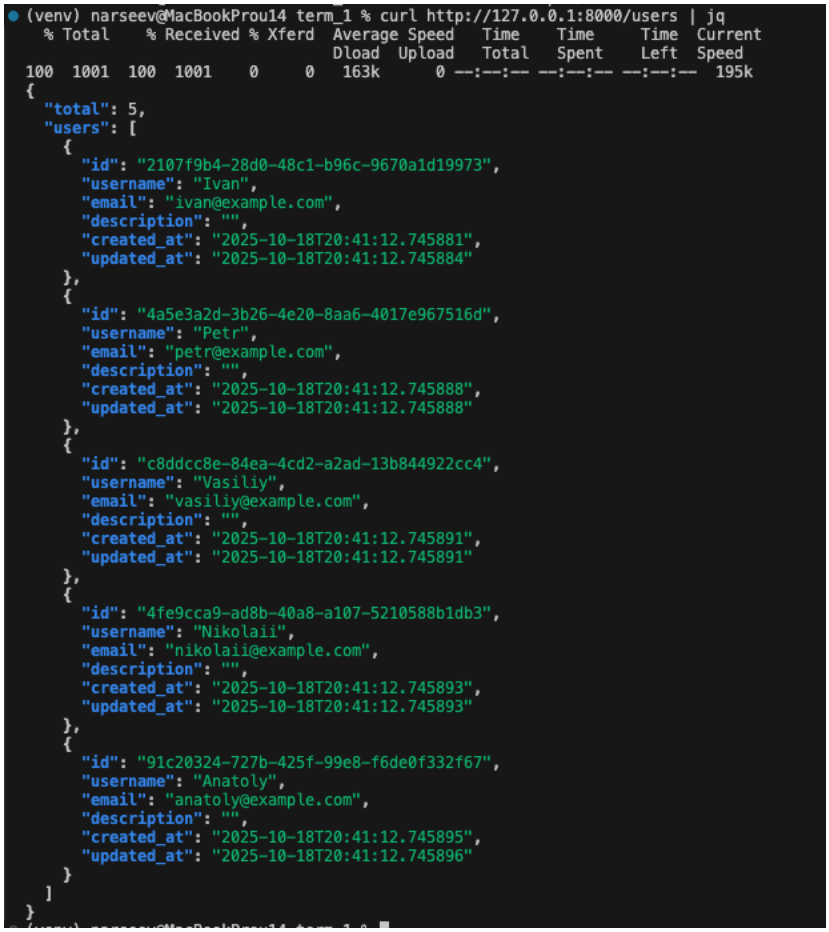
Освоить принципы Dependency Injection и интеграцию SQLAlchemy ORM в веб-приложении на базе фреймворка Litestar написав CRUD.

Задачи

В результате был реализован CRUD для пользователей

1. Изначально был **GET /users** со списком пользователей (тут скрин из браузера, далее будет терминал):

Потом увидел «*. Задание со звездочкой», поэтому был добавлен возврат количества юзеров:



В базе то же самое:

```
labdb=# select * from users;
                  id                  | username |        email        |         created_at         |         updated_at         | description
--------------------------------------+----------+---------------------+----------------------------+----------------------------+-------------
 2107f9b4-28d0-48c1-b96c-9670a1d19973 | Ivan     | ivan@example.com    | 2025-10-18 20:41:12.745881 | 2025-10-18 20:41:12.745884 |
 4a5e3a2d-3b26-4e20-8aa6-4017e967516d | Petr     | petr@example.com    | 2025-10-18 20:41:12.745888 | 2025-10-18 20:41:12.745888 |
 c8ddcc8e-84ea-4cd2-a2ad-13b844922cc4 | Vasiliy  | vasiliy@example.com | 2025-10-18 20:41:12.745891 | 2025-10-18 20:41:12.745891 |
 4fe9cca9-ad8b-40a8-a107-5210588b1db3 | Nikolaii | nikolaii@example.com| 2025-10-18 20:41:12.745893 | 2025-10-18 20:41:12.745893 |
 91c20324-727b-425f-99e8-f6de0f332f67 | Anatoly  | anatoly@example.com | 2025-10-18 20:41:12.745895 | 2025-10-18 20:41:12.745896 |
(5 rows)
```

2. **GET /users/{id}**

```
(venv) narseev@MacBookProu14 term_1 % curl http://127.0.0.1:8000/users/2107f9b4-28d0-48c1-b96c-9670a1d19973 | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   191  100   191    0     0  14601      0 --:--:-- --:--:-- --:--:-- 14692
{
  "id": "2107f9b4-28d0-48c1-b96c-9670a1d19973",
  "username": "Ivan",
  "email": "ivan@example.com",
  "description": "",
  "created_at": "2025-10-18T20:41:12.745881",
  "updated_at": "2025-10-18T20:41:12.745884"
}
```

3. **POST /users**

Добавляем нового пользователя и смотрим БД

```
(venv) narseev@MacBookProu14 term_1 % curl -X POST http://127.0.0.1:8000/users \
  -H "Content-Type: application/json" \
  -d '{"username":"Sergey","email":"sergey@example.com","description":"Новый пользователь"}' | jq

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   332  100   230  100   102  28395  12592 --:--:-- --:--:-- --:--:-- 41500
{
  "id": "6b75f931-bc0e-4d50-8edc-4287bab0e379",
  "username": "Sergey",
  "email": "sergey@example.com",
  "description": "Новый пользователь",
  "created_at": "2025-10-25T15:28:34.496381",
  "updated_at": "2025-10-25T15:28:34.496383"
}
```

В базе он появился:

```
labdb=# select * from users;
                  id                  | username |        email        |         created_at         |         updated_at         |    description
--------------------------------------+----------+---------------------+----------------------------+----------------------------+--------------------
 2107f9b4-28d0-48c1-b96c-9670a1d19973 | Ivan     | ivan@example.com    | 2025-10-18 20:41:12.745881 | 2025-10-18 20:41:12.745884 |
 4a5e3a2d-3b26-4e20-8aa6-4017e967516d | Petr     | petr@example.com    | 2025-10-18 20:41:12.745888 | 2025-10-18 20:41:12.745888 |
 c8ddcc8e-84ea-4cd2-a2ad-13b844922cc4 | Vasiliy  | vasiliy@example.com | 2025-10-18 20:41:12.745891 | 2025-10-18 20:41:12.745891 |
 4fe9cca9-ad8b-40a8-a107-5210588b1db3 | Nikolaii | nikolaii@example.com| 2025-10-18 20:41:12.745893 | 2025-10-18 20:41:12.745893 |
 91c20324-727b-425f-99e8-f6de0f332f67 | Anatoly  | anatoly@example.com | 2025-10-18 20:41:12.745895 | 2025-10-18 20:41:12.745896 |
 6b75f931-bc0e-4d50-8edc-4287bab0e379 | Sergey   | sergey@example.com  | 2025-10-25 15:28:34.496381 | 2025-10-25 15:28:34.496383 | Новый пользователь
(6 rows)
```

4. **PUT /users/{id}**

Обновляем данные пользователя:

```
(venv) narseev@MacBookProu14 term_1 % curl -X PUT http://127.0.0.1:8000/users/6b75f931-bc0e-4d50-8edc-4287bab0e379 \
  -H "Content-Type: application/json" \
  -d '{"username": "SergeyUpdated", "description": "Обновленный пользователь"}' | jq

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   344  100   249  100    95  32875  12542 --:--:-- --:--:-- --:--:-- 49142
{
  "id": "6b75f931-bc0e-4d50-8edc-4287bab0e379",
  "username": "SergeyUpdated",
  "email": "sergey@example.com",
  "description": "Обновленный пользователь",
  "created_at": "2025-10-25T15:28:34.496381",
  "updated_at": "2025-10-25T15:29:39.954677"
}
```

В БД тоже изменения есть:

```
labdb=# select * from users;
                  id                  |   username    |        email        |         created_at         |         updated_at         |       description
--------------------------------------+---------------+---------------------+----------------------------+----------------------------+--------------------------
 2107f9b4-28d0-48c1-b96c-9670a1d19973 | Ivan          | ivan@example.com    | 2025-10-18 20:41:12.745881 | 2025-10-18 20:41:12.745884 |
 4a5e3a2d-3b26-4e20-8aa6-4017e967516d | Petr          | petr@example.com    | 2025-10-18 20:41:12.745888 | 2025-10-18 20:41:12.745888 |
 c8ddcc8e-84ea-4cd2-a2ad-13b844922cc4 | Vasiliy       | vasiliy@example.com | 2025-10-18 20:41:12.745891 | 2025-10-18 20:41:12.745891 |
 4fe9cca9-ad8b-40a8-a107-5210588b1db3 | Nikolaii      | nikolaii@example.com | 2025-10-18 20:41:12.745893 | 2025-10-18 20:41:12.745893 |
 91c20324-727b-425f-99e8-f6de0f332f67 | Anatoly       | anatoly@example.com | 2025-10-18 20:41:12.745895 | 2025-10-18 20:41:12.745896 |
 6b75f931-bc0e-4d50-8edc-4287bab0e379 | SergeyUpdated | sergey@example.com  | 2025-10-25 15:28:34.496381 | 2025-10-25 15:29:39.954677 | Обновленный пользователь
(6 rows)
```

5. **DELETE /users/{id}**

Удаляем пользователя:

```
(venv) narseev@MacBookProu14 term_1 % curl -X DELETE http://127.0.0.1:8000/users/6b75f931-bc0e-4d50-8edc-4287bab0e379
(venv) narseev@MacBookProu14 term_1 %
```

В БД тоже удалился:

```
labdb=# select * from users;
                  id                  | username |        email        |         created_at         |         updated_at         | description
--------------------------------------+----------+---------------------+----------------------------+----------------------------+-------------
 2107f9b4-28d0-48c1-b96c-9670a1d19973 | Ivan     | ivan@example.com    | 2025-10-18 20:41:12.745881 | 2025-10-18 20:41:12.745884 |
 4a5e3a2d-3b26-4e20-8aa6-4017e967516d | Petr     | petr@example.com    | 2025-10-18 20:41:12.745888 | 2025-10-18 20:41:12.745888 |
 c8ddcc8e-84ea-4cd2-a2ad-13b844922cc4 | Vasiliy  | vasiliy@example.com | 2025-10-18 20:41:12.745891 | 2025-10-18 20:41:12.745891 |
 4fe9cca9-ad8b-40a8-a107-5210588b1db3 | Nikolaii | nikolaii@example.com | 2025-10-18 20:41:12.745893 | 2025-10-18 20:41:12.745893 |
 91c20324-727b-425f-99e8-f6de0f332f67 | Anatoly   | anatoly@example.com | 2025-10-18 20:41:12.745895 | 2025-10-18 20:41:12.745896 |
(5 rows)
```

Вопросы

1. **Объясните принцип Dependency Injection (DI) своими словами.** Какую проблему он решает в контексте разработки приложений и какие преимущества дает?
   DI – способ передать зависимости в классы/функции извне, а не создавать их внутри
   Решает проблему изоляции и тестируемости. Код не зависит напрямую от конкретных реализаций
   Преимущества: легкое тестирование, переиспользование, чистая архитектура
2. **Каковы основные обязанности каждого из трех слоев приложения (Repository, Service, Controller)?** Почему такое разделение считается хорошей практикой? Что произойдет, если объединить логику репозитория и контроллера?
   Repository – работа с БД (CRUD, запросы)
   Service – бизнес-логика (правила, валидация, транзакции)
   Controller – прием http, валидация входа, возврат ответа
   Разделение нужно для масштабируемости, разделения ответственности, тестируемости
   Если объединить Repository и Controller то сложнее будет тестировать, логика БД смешается с http

3. **Объясните жизненный цикл зависимости в Litestar.** Как именно создается и когда уничтожается экземпляр сессии базы данных при обработке одного HTTP-запроса?
   Зависимость создается в начале запроса, передается во все нужные слои и уничтожается в конце запроса автоматически при выходе из контекста
   Т.е. одна сессия на запрос
4. **Что такое** async/await **и зачем они используются в данном приложении?** Как асинхронность влияет на производительность при работе с базой данных?
   async/await позволяют выполнять операции не блокирую поток
   Пока БД обрабатывает запрос, сервер может обслуживать другие запросы, что повышает производительность и масштабируемочть
5. **Почему в сигнатурах методов** UserRepository **первым аргументом передается** session**?** Почему бы не создать его внутри репозитория? Кто и когда должен вызывать session.commit() или session.rollback()?
   session передаётся явно, чтобы контролировать транзакции на уровне запроса
   Репозиторий не создаёт её сам, потому что этим управляет DI
   session.commit() или rollback() вызываются в контроллере или middleware - там, где известно, успешно ли завершился запрос
6. **Для чего в методе get_by_filter используется пагинация (count и page)?**
   Чтобы не грузить все записи сразу (экономия памяти, ускорение ответа)
7. **В текущей реализации** UserService **практически не содержит бизнес-логики и является "прокси" для** UserRepository**.** Приведите пример конкретной бизнес-логики (например, проверка уникальности email, хеширование пароля, отправка приветственного письма), которую можно было бы добавить в сервисный слой.
   При добавлении пользователя автоматически добавлять ему дату внесения в базу
8. **Какие HTTP-статусы должны возвращать каждый из эндпоинтов (**get**,** post**,** put**,** delete**) в различных сценариях (успех, ошибка, не найден)?** Обоснуйте свой выбор.

Метод	*Успех*	*Ошибка валидации*	*Не найдено*	*Ошибка сервера*
GET	*200*	*400 Bad Request*	*404 Not Found*	*500*
POST	*201*	*400 Bad Request*		*500*
PUT	*200*	*400 Bad Request*	*404 Not Found*	*500*
DELETE	*204*	*-*	*404 Not Found*	*500*

Ссылки

GitHub: https://github.com/gh-u14/AppDev/