MobileDogs

Выпуск 1.0

Kuzmin Ilia, Nikiforova Darya

июн. 01, 2024

Contents:

Документация проекта MobileDogs:

1. Описание базы данных

src/database.md

2. Описание функций для пользователей

src/users/users.md

3. Описание функций для ошейников

src/devices/devices.md

4. Описание функций для заданий

src/tasks/tasks.md

. automodule:: mobiledogs

members

Глава 1

Идея проекта:

Проект «Mobile Dogs» представляет собой систему для отслеживания бездомных собак с помощью носимых устройств (ошейников). Каждый пользователь может выбрать себе некоторое количество бездомных собак для отслеживания. Персонал устанавливает новые ошейники на бездомных собак. Пользователи могут наблюдать за собаками, давать задания другим пользователям, а также получать данные о собаках через приложение. Персонал устанавливает новые ошейники на бездомных собак. Есть базовые станции LoRa для связи носимых устройств (ошейников). Есть регистрация пользователей и ошейников. Носимые устройства регистрирует только персонал.

Глава 2

Сценарии:

1. Регистрация пользователей:
    - Пользователи могут зарегистрироваться, указав свои данные и получив уникальный идентификатор.
2. Регистрация ошейников:
    - Персонал может зарегистрировать новые ошейники, присваивая им уникальные идентификаторы и привязывая к конкретным собакам.
3. Оповещение пользователей:
    - Пользователи получают данные о местоположении и состоянии собак через приложение.
4. Модерирование:
    - Персонал следит за состоянием ошейников, информацией о пользователях и собаках, и при необходимости, модерирует контент.
5. Задание от одних пользователям другим:
    - Пользователи могут давать задания другим пользователям по уходу за определенными собаками, кормлением или выполнению других действий.
6. Привязка ошейников и пользователей:
    - Пользователи могут «привязывать» зарегистрированные ошейники к своему аккаунту для отслеживания конкретных собак.

Глава 3

Необходимые запросы для реализации сценариев:

1. POST request для регистрации пользователей
2. PUT request для изменения пользователя
3. DELETE request для удаления пользователя
4. POST request для добавления задания
5. PUT request для изменения задания
6. DELETE request для удаления задания
7. POST request для добавления ошейника
8. PUT request для изменения ошейника
9. DELETE request для удаления ошейника
10. GET request для получения списка всех ошейников
11. POST request для добавления координат
12. PUT request для изменения координат
13. DELETE request для удаления координат
14. GET request для возвращения всех координат ошейника по заданному времени
15. POST request для добавления связи пользователя с ошейником
16. PUT request для изменения связи пользователя с ошейником
17. DELETE request для удаления связи пользователя с ошейником

Глава 4

Инструкции для установки зависимостей и запуска приложения:

4.1 Установка зависимостей

1. Убедитесь, что у вас установлены Python версии 3.6 или выше, а также pip версии 10.0 или выше.
2. Создайте виртуальное окружение, чтобы изолировать зависимости приложения от других проектов. Используйте команду:

```
python3 -m venv venv
```

3. Активируйте виртуальное окружение:

```
source venv/bin/activate
```

4. Установите зависимости из файла requirements.txt. Запустите команду:

```
pip install -r requirements.txt
```

4.2 Доп. установки и настройки

1. Настройте ELK
2. Настройте Filebeat

4.3 Запуск приложения

1. Клонируйте репозиторий:

```
sudo apt update
sudo apt install git
git clone URL
```

2. Перейдите в каталог приложения (~/MobileDogs/src).

3. Создайте новую виртуальную среду и активируйте:

```
python -m venv venv
source .venv/bin/activate
```

4. Запустите приложение с помощью команды:

```
uvicorn main:app --reload
```

5. Запустите Filebeat в отдельном окне и проверить статус работы всех серверов (в файле `/etc/filebeat/filebeat.yml` в `filebeat.inputs` в `parth` ~/MobileDogs/src/log/*.json)

```
sudo systemctl status elasticsearch.service
sudo systemctl status kibana.service
sudo systemctl status logstash.service
sudo filebeat -e
```

6. Чтобы выйти из приложения, нажмите `Ctrl + C` в терминале, в котором оно запущено. Чтобы деактивировать виртуальное окружение, запустите команду: `deactivate`

psycopg2-binary<=2.8.3 ecs_logging<=2.1.0 fastapi<=0.103.0 Flask<=1.1.1 markdown-it-py<=2.2.0 matplotlib<=3.1.3 myst-parser<=1.0.0 networkx<=2.4 passlib<=1.7.4 python-dateutil==2.8.1 python-jsonrpc-server<=0.3.4 python-language-server<=0.31.7 python-logstash-async<=3.0.0 pydantic<=2.5.3 Requests<=2.31.0 Sphinx<=5.3.0 sphinxcontrib-applehelp<=1.0.2 sphinxcontrib-devhelp<=1.0.2 sphinxcontrib-htmlhelp<=2.0.5 sphinxcontrib-jsmath<=1.0.1 sphinxcontrib-qthelp<=1.0.7 sphinxcontrib-serializinghtml<=1.1.10 SQLAlchemy<=2.0.30 sqlalchemy-orm<=1.2.10 uvicorn<=0.30.0 pathlib h5py>=2.9.0 clyent==1.2.1 nbformat==5.4.0 requests==2.28.1 jedi>=0.16

LICENSE README.md setup.py MobileDogs.egg-info/PKG-INFO MobileDogs.egg-info/SOURCES.txt MobileDogs.egg-info/dependency_links.txt MobileDogs.egg-info/requires.txt MobileDogs.egg-info/top_level.txt src/__init__.py src/conf.py src/database.py src/database_manager.py src/main.py

psycopg2-binary<=2.8.3 ecs_logging<=2.1.0 fastapi<=0.103.0 Flask<=1.1.1 markdown-it-py<=2.2.0 matplotlib<=3.1.3 myst-parser<=1.0.0 networkx<=2.4 passlib<=1.7.4 python-dateutil==2.8.1 python-jsonrpc-server<=0.3.4 python-language-server<=0.31.7 python-logstash-async<=3.0.0 pydantic<=2.5.3 Requests<=2.31.0 Sphinx<=5.3.0 sphinxcontrib-applehelp<=1.0.2 sphinxcontrib-devhelp<=1.0.2 sphinxcontrib-htmlhelp<=2.0.5 sphinxcontrib-jsmath<=1.0.1 sphinxcontrib-qthelp<=1.0.7 sphinxcontrib-serializinghtml<=1.1.10 SQLAlchemy<=2.0.30 sqlalchemy-orm<=1.2.10 uvicorn<=0.30.0 pathlib h5py>=2.9.0 clyent==1.2.1 nbformat==5.4.0 requests==2.28.1 jedi>=0.16

[test] pytest coverage

src

Глава 5

Описание базы данных

В базе данных есть 5 таблиц:

1. `users`

- **id:** Идентификатор пользователя (первичный ключ).
- **login:** Логин пользователя (уникальный).
- **password:** Хэшированный пароль пользователя.
- **email:** Адрес электронной почты пользователя (уникальный).
- **gender:** Пол пользователя (необязательно).
- **phone:** Номер телефона пользователя (необязательно).
- **birthday:** Дата рождения пользователя (необязательно).
- **registration_date:** Дата регистрации пользователя.
- **deletion_date:** Дата удаления (архивирования) пользователя (необязательно).
- **archived:** Флаг, указывающий, был ли пользователь удален (архивирован) (по умолчанию False).

2. `tasks`

- **id:** Идентификатор задачи (первичный ключ).
- **id_user_1:** Идентификатор первого пользователя, связанного с задачей (используется внешний ключ для связи с таблицей `users`).
- **id_user_2:** Идентификатор второго пользователя, связанного с задачей (используется внешний ключ для связи с таблицей `users`).
- **id_task:** Дополнительный идентификатор задачи (необязательно).
- **confirm:** Флаг, указывающий, подтверждена ли задача (по умолчанию False).
- **send_date:** Дата отправки задачи.
- **completion_date:** Дата завершения задачи (необязательно).

- **task_text:** Текст задачи.
- **archived:** Флаг, указывающий, была ли задача удалена (архивирована) (по умолчанию False).

3. `dogs_collar`

- **id:** Идентификатор ошейника (первичный ключ).
- **uni_num_dog:** Уникальный номер ошейника.
- **name_dog:** Кличка собаки.
- **feeling_hungry:** Флаг, указывающий, голодна ли собака (необязательно).
- **health_status:** Состояние здоровья собаки (необязательно).
- **registration_date:** Дата регистрации ошейника.
- **deletion_date:** Дата удаления (архивирования) ошейника (необязательно).
- **archived:** Флаг, указывающий, был ли ошейник удален (архивирован) (по умолчанию False).

4. `users_dogs_collar`

- **id:** Идентификатор связи между пользователем и ошейником (первичный ключ).
- **id_user:** Идентификатор пользователя (используется внешний ключ для связи с таблицей `users`).
- **id_collar:** Идентификатор ошейника (используется внешний ключ для связи с таблицей `dogs_collar`).
- **binding_date:** Дата связи между пользователем и ошейником.
- **unbinding_date:** Дата разрыва связи между пользователем и ошейником (необязательно).
- **archived:** Флаг, указывающий, была ли связь удалена (архивирована) (по умолчанию False).

5. `coordinates`

- **id:** Идентификатор координат (первичный ключ).
- **collar_id:** Идентификатор ошейника (используется внешний ключ для связи с таблицей `dogs_collar`).
- **latitude:** Широта.
- **longitude:** Долгота.
- **timestamp:** Время фиксации координат.

Схемы отношений:

- `users` имеет отношение **«один ко многим»** с `tasks` (через `id_user_1` и `id_user_2`).
- `users` имеет отношение **«один ко многим»** с `users_dogs_collar` (через `id_user`).
- `dogs_collar` имеет отношение **«один ко многим»** с `users_dogs_collar` (через `id_collar`).
- `dogs_collar` имеет отношение **«один ко многим»** с `coordinates` (через `collar_id`).

5.1 Функциональность:

База данных может быть использована для:

- **Управления пользователями:** Регистрации, входа, изменения профиля, архивирования.
- **Планирования задач:** Создание, подтверждение, отправка, завершение, архивирование.
- **Отслеживания собак:** Регистрация ошейников, запись координат, управление связью между ошейником и пользователем.

5.2 Дополнительные замечания:

- `archived`-флаги позволяют архивировать данные, а не полностью удалять их. Это может быть полезно для восстановления данных или для аналитики.
- `TIMESTAMP` с `default='now()'` используется для автоматического заполнения поля датой и временем при добавлении новой записи.
- Внешние ключи обеспечивают целостность данных, гарантируя, что записи связаны между собой правильно.

Глава 6

Функции для работы с ошейниками

1. `create_dog_collar(collar: DogCollarCreate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за создание новых ошейников.

Параметры:

- `collar: DogCollarCreate`: Данные нового ошейника, которые передаются в виде объекта `DogCollarCreate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.add_dog_collar` для добавления нового ошейника в базу данных.
2. Если функция `db_manager.add_dog_collar` успешно добавляет ошейник, функция записывает информацию об успешном добавлении в лог и возвращает ID ошейника и сообщение об успешном добавлении.
3. Обрабатывает любые исключения, возникшие в процессе добавления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

2. `update_dog_collar(collar_id: int, collar_update: DogCollarUpdate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за обновление данных существующего ошейника.

Параметры:

- `collar_id: int`: ID ошейника, который нужно обновить.
- `collar_update: DogCollarUpdate`: Данные для обновления, которые передаются в виде объекта `DogCollarUpdate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.update_dog_collar` для обновления данных ошейника в базе данных.

2. Если функция `db_manager.update_dog_collar` успешно обновляет данные, функция записывает информацию об успешном обновлении в лог и возвращает сообщение об успешном обновлении.
3. Обрабатывает любые исключения, возникшие в процессе обновления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

3. `delete_dog_collar(collar_id: int, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за удаление (архивацию) ошейника.

Параметры:

- `collar_id: int`: ID ошейника, который нужно удалить.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.archive_dog_collar` для архивирования ошейника в базе данных.
2. Если функция `db_manager.archive_dog_collar` успешно архивирует ошейник, функция записывает информацию об успешном удалении в лог и возвращает сообщение об успешном удалении.
3. Обрабатывает любые исключения, возникшие в процессе удаления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

4. `get_all_dog_collars(db: Session = Depends(get_db))`

Описание: Эта функция возвращает список всех ошейников.

Параметры:

- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.query_all_dog_collars` для получения всех ошейников из базы данных.
2. Если функция `db_manager.query_all_dog_collars` успешно возвращает список ошейников, функция записывает информацию об успешном получении в лог и возвращает список ошейников.
3. Обрабатывает любые исключения, возникшие в процессе получения, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

5. `create_coordinate(coordinate: CoordinateCreate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за создание новых координат.

Параметры:

- `coordinate: CoordinateCreate`: Данные новых координат, которые передаются в виде объекта `CoordinateCreate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.add_coordinate` для добавления новых координат в базу данных.
2. Если функция `db_manager.add_coordinate` успешно добавляет координаты, функция записывает информацию об успешном добавлении в лог и возвращает ID координат и сообщение об успешном добавлении.

3. Обрабатывает любые исключения, возникшие в процессе добавления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

6. `update_coordinate(coordinate_id: int, coordinate_update: CoordinateUpdate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за обновление данных существующих координат.

Параметры:

- `coordinate_id: int`: ID координат, которые нужно обновить.
- `coordinate_update: CoordinateUpdate`: Данные для обновления, которые передаются в виде объекта `CoordinateUpdate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.update_coordinate` для обновления данных координат в базе данных.
2. Если функция `db_manager.update_coordinate` успешно обновляет данные, функция записывает информацию об успешном обновлении в лог и возвращает сообщение об успешном обновлении.
3. Обрабатывает любые исключения, возникшие в процессе обновления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

7. `delete_coordinate(coordinate_id: int, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за удаление координат.

Параметры:

- `coordinate_id: int`: ID координат, которые нужно удалить.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.delete_coordinate` для удаления координат из базы данных.
2. Если функция `db_manager.delete_coordinate` успешно удаляет координаты, функция записывает информацию об успешном удалении в лог и возвращает сообщение об успешном удалении.
3. Обрабатывает любые исключения, возникшие в процессе удаления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

8. `get_track(collar_id: int, start_time: str, end_time: str, db: Session = Depends(get_db))`

Описание: Эта функция возвращает список координат для заданного ошейника за определенный период времени.

Параметры:

- `collar_id: int`: ID ошейника.
- `start_time: str`: Начальное время периода.
- `end_time: str`: Конечное время периода.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.get_track` для получения списка координат из базы данных.

2. Если функция `db_manager.get_track` успешно возвращает список координат, функция записывает информацию об успешном получении в лог и возвращает список координат.
3. Обрабатывает любые исключения, возникшие в процессе получения, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

9. `create_user_dog_collar(binding: UserDogCollarCreate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за создание новой связи между пользователем и ошейником.

Параметры:

- `binding: UserDogCollarCreate`: Данные новой связи, которые передаются в виде объекта `UserDogCollarCreate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.add_user_dog_collar` для добавления новой связи в базу данных.
2. Если функция `db_manager.add_user_dog_collar` успешно добавляет связь, функция записывает информацию об успешном добавлении в лог и возвращает ID пользователя, ID ошейника и сообщение об успешном добавлении.
3. Обрабатывает любые исключения, возникшие в процессе добавления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

10. `update_user_dog_collar(binding_id: int, binding_update: UserDogCollarUpdate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за обновление данных существующей связи между пользователем и ошейником.

Параметры:

- `binding_id: int`: ID связи, которую нужно обновить.
- `binding_update: UserDogCollarUpdate`: Данные для обновления, которые передаются в виде объекта `UserDogCollarUpdate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.update_user_dog_collar` для обновления данных связи в базе данных.
2. Если функция `db_manager.update_user_dog_collar` успешно обновляет данные, функция записывает информацию об успешном обновлении в лог и возвращает сообщение об успешном обновлении.
3. Обрабатывает любые исключения, возникшие в процессе обновления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

11. `delete_user_dog_collar(binding_id: int, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за удаление связи между пользователем и ошейником.

Параметры:

- `binding_id: int`: ID связи, которую нужно удалить.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.archive_user_dog_collar` для удаления связи из базы данных.
2. Если функция `db_manager.archive_user_dog_collar` успешно удаляет связь, функция записывает информацию об успешном удалении в лог и возвращает сообщение об успешном удалении.
3. Обрабатывает любые исключения, возникшие в процессе удаления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

6.1 Дополнительные замечания:

- `db_manager`: Класс, который предоставляет функции для взаимодействия с базой данных.
- `log_message`: Функция для записи сообщений в лог-файл.
- `HTTPException`: Это исключение используется для генерации ответа HTTP с кодом ошибки.

6.2 Пример использования:

```
# Создание нового ошейника
collar_data = DogCollarCreate(uni_num_dog="1234567890", name_dog="Rex", ...)
collar_id, message = create_dog_collar(collar_data, db)

# Обновление данных ошейника
collar_update_data = DogCollarUpdate(name_dog="Rexy", ...)
update_dog_collar(collar_id, collar_update_data, db)

# Удаление (архивирование) ошейника
delete_dog_collar(collar_id, db)

# Получение списка всех ошейников
all_collars = get_all_dog_collars(db)

# Создание новых координат
coordinate_data = CoordinateCreate(collar_id=collar_id, latitude=55.7558, longitude=37.
↪6173)
coordinate_id, message = create_coordinate(coordinate_data, db)

# Обновление данных координат
coordinate_update_data = CoordinateUpdate(latitude=55.7560, longitude=37.6175)
update_coordinate(coordinate_id, coordinate_update_data, db)

# Удаление координат
delete_coordinate(coordinate_id, db)

# Получение списка координат для ошейника за определенный период
track = get_track(collar_id, "2023-10-27T10:00:00Z", "2023-10-27T12:00:00Z", db)

# Создание связи между пользователем и ошейником
binding_data = UserDogCollarCreate(id_user=123, id_collar=collar_id)
userid,dogid,message = create_user_dog_collar(binding_data, db)

# Обновление данных связи
```

(continues on next page)

(продолжение с предыдущей страницы)

```
binding_update_data = UserDogCollarUpdate(id_user=123, id_collar=901)
update_user_dog_collar(binding_id, binding_update_data, db)

# Удаление связи
delete_user_dog_collar(binding_id, db)
```

Глава 7

Функции для работы с заданиями

1. `create_task(task: TaskCreate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за создание новых заданий.

Параметры:

- `task: TaskCreate`: Данные нового задания, которые передаются в виде объекта `TaskCreate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.add_task` для добавления нового задания в базу данных.
2. Если функция `db_manager.add_task` успешно добавляет задание, функция записывает информацию об успешном добавлении в лог и возвращает ID задания, сообщение об успешном добавлении и текст задания.
3. Обрабатывает любые исключения, возникшие в процессе добавления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.
4. `update_task(task_id: int, task_update: TaskUpdate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за обновление данных существующего задания.

Параметры:

- `task_id: int`: ID задания, которое нужно обновить.
- `task_update: TaskUpdate`: Данные для обновления, которые передаются в виде объекта `TaskUpdate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.update_task` для обновления данных задания в базе данных.
2. Если функция `db_manager.update_task` успешно обновляет данные, функция записывает информацию об успешном обновлении в лог и возвращает сообщение об успешном обновлении.

3. Обрабатывает любые исключения, возникшие в процессе обновления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.
4. `delete_task(task_id: int, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за удаление (архивацию) задания.

Параметры:

- `task_id: int`: ID задания, которое нужно удалить.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.archive_task` для архивирования задания в базе данных.
2. Если функция `db_manager.archive_task` успешно архивирует задание, функция записывает информацию об успешном удалении в лог и возвращает сообщение об успешном удалении.
3. Обрабатывает любые исключения, возникшие в процессе удаления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

7.1 Дополнительные замечания:

- `db_manager`: Класс, который предоставляет функции для взаимодействия с базой данных.
- `log_message`: Функция для записи сообщений в лог-файл.
- `HTTPException`: Это исключение используется для генерации ответа HTTP с кодом ошибки.

7.2 Пример использования:

```
# Создание нового задания
task_data = TaskCreate(id_user_1=123, id_user_2=456, id_task=789, task_text="Купить хлеб
↪")
task_id, message, task_text = create_task(task_data, db)

# Обновление данных задания
task_update_data = TaskUpdate(confirm=True, task_text="Купить молоко")
update_task(task_id, task_update_data, db)

# Удаление (архивирование) задания
delete_task(task_id, db)
```

Глава 8

Функции для работы с пользователями

1. `create_user(user: UserCreate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за регистрацию новых пользователей.

Параметры:

- `user: UserCreate`: Данные нового пользователя, которые передаются в виде объекта `UserCreate`.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Хеширует пароль пользователя с помощью функции `get_password_hash`.
2. Создает словарь `user_data` из объекта `user`, исключая поле `deletion_date`.
3. Заменяет поле `password` в словаре `user_data` на хешированный пароль.
4. Вызывает функцию `db_manager.add_user` для добавления нового пользователя в базу данных.
5. Если функция `db_manager.add_user` возвращает `user_id` равный 0, то это означает ошибку. В этом случае функция записывает ошибку в лог с помощью функции `log_message` и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.
6. Если функция `db_manager.add_user` успешно добавляет пользователя, функция записывает информацию об успешной регистрации в лог и возвращает `user_id` и сообщение об успешной регистрации.
7. Обрабатывает любые исключения, возникшие в процессе регистрации, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.
8. `update_user(user_id: int, user_update: UserUpdate, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за обновление данных существующего пользователя.

Параметры:

- `user_id: int`: ID пользователя, которого нужно обновить.
- `user_update: UserUpdate`: Данные для обновления, которые передаются в виде объекта `UserUpdate`.

- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.update_user` для обновления данных пользователя в базе данных.
2. Если функция `db_manager.update_user` успешно обновляет данные, функция записывает информацию об успешном обновлении в лог и возвращает сообщение об успешном обновлении.
3. Обрабатывает любые исключения, возникшие в процессе обновления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.
4. `delete_user(user_id: int, db: Session = Depends(get_db))`

Описание: Эта функция отвечает за удаление (архивацию) пользователя.

Параметры:

- `user_id: int`: ID пользователя, которого нужно удалить.
- `db: Session`: Сессия базы данных, полученная из контекста зависимости `get_db`.

Действия:

1. Вызывает функцию `db_manager.archive_user` для архивирования пользователя в базе данных.
2. Если функция `db_manager.archive_user` успешно архивирует пользователя, функция записывает информацию об успешном удалении в лог и возвращает сообщение об успешном удалении.
3. Обрабатывает любые исключения, возникшие в процессе удаления, записывает их в лог и генерирует исключение `HTTPException` с кодом 500 и описанием ошибки.

8.1 Дополнительные замечания:

- `db_manager`: Класс, который предоставляет функции для взаимодействия с базой данных.
- `log_message`: Функция для записи сообщений в лог-файл.
- `get_password_hash`: Функция для хеширования паролей.
- `HTTPException`: Это исключение используется для генерации ответа HTTP с кодом ошибки.

8.2 Пример использования:

```
# Регистрация нового пользователя
user_data = UserCreate(login="JohnDoe", password="password123", email="johndoe@example.
↪com", ...)
user_id, message = create_user(user_data, db)

# Обновление данных пользователя
user_update_data = UserUpdate(phone="88001234567")
update_user(user_id, user_update_data, db)

# Удаление (архивирование) пользователя
delete_user(user_id, db)
```