Это документация по вэб сервису, основанного на принципах RESTful API (ver. 1 beta).

Функциональность данного API реализована на фреймворке Symfony 5 в контроллере UserController. Для тестирования использовался встроенный вэб-сервер Symfony по адресу http://127.0.0.1:8000. Запросы отсылались программой Postman.

Ключевые моменты:

1. Все ответы возвращаются в формате JSON.
2. Все запросы принимаются в формате JSON (в теле запроса) или в качестве параметров (только логин и пароль).
3. Проверка формата запроса.
4. Проверка вводимых данных.
5. Заголовки ответа согласно спецификации HTTP.
6. Понятные ответы при ошибках запроса или обработки.
7. Все действия в коде детально прокомментированы.
8. Код проверен по стандарту PSR-0, PSR-1, PSR-2.

Проверка регистрации.

При подтверждении пользователя (логин и пароль) ему выдается токен, наличие которого проверяется каждый раз при вызове следующих методов:

- allUsers (получить всех пользователей)

- getById (получить пользователя по Id)

- userGet (получить ответ на json-запрос пользователей по полям nickname или email)

Проверка реализована методом **checkAccess** и выполняется следующим образом:

Проверяется наличие заголовка AccessHash в запросе. Если заголовок отсутствует, функция checkAccess возвращает False в метод, в котором была вызвана проверка. Метод не выполняется и возвращает сообщение об ошибке.

Описание методов класса **UserController**.

1. Метод **allUsers**

Адрес: /user метод: GET

Возвращает колонки *first_name, last_name, nickname, email*, age всех существующих пользователей в базе.

Пример ответа:

[

```
    {
        "first_name": "Ivan",
        "last_name": "Dubrovskiy",
        "nickname": "DuB",
        "email": "dubrovskiy.i@mail.com",
        "age": "28"
    },
    {
        "first_name": "Oleg",
        "last_name": "Petrovich",
        "nickname": "OlegPet",
        "email": "opetrov@mail.com",
        "age": "43"
    },
    {
       { Остальные записи }
    }
]
```

Выполнение функции:

1. При запросе по адресу /user формируется и исполняется запрос в базу данных.
2. Выводит все существующие записи.

2. Метод **userById**

Адрес: /user/id метод: GET

Возвращает строки id, first_name, last_name, nickname, email, age найденных пользователей по id.

Необходимый id передается в адресе и принимается методом в качестве параметра. Пример запроса:

/user/26

Ответ будет следующим:

```
{
    "id": "26",
    "first_name": "laravel",
    "last_name": "8",
    "nickname": "lara",
    "email": "lara@fabian.com",
    "age": "0"
}
```

Выполнение функции.

1. Проверяется, является ли переданный параметр числом. Если нет, возвращается ошибка (Ошибка 1).
2. Выполняется поиск строк в базе, соответствующие переданному id.
3. Если результат не пустой, возвращаем результат
4. Если запись не обнаружена, возвращаем сообщение об ошибке (Ошибка 2).

Сообщения об ошибках:

Ошибка 1. Переданный параметр не число.

```
{
    "error": "Id must be integer"
}
```

Ошибка 2. Пользователь не найден.

```
{
    "query": "search for id=288",
    "error": "NOT FOUND"
}
```

3. Метод **addUser**

Адрес: /user метод: PUT

Создает нового пользователя и записывает данные в базу.

Выполнение функции.

1. Тело запроса декодируется в php-массив. Если формат запроса неверен (не json) функция вернет false, условие не выполнится и будет возвращено сообщение об ошибке (Ошибка 1).
2. Проверяем обязательные поля: сравниваем имена полей с шаблоном. Если имена отличаются, возвращаем сообщение об ошибке (Ошибка 2).
3. Проверяем, являются ли поля заполненными (все поля регистрации обязательные). Если нет или пустые, возвращаем сообщение (Ошибка 3).
4. Проверяем, введено ли в поле age число. Если нет, выводим ошибку (Ошибка 4).
5. Проверяем корректность ввода email. Если формат некорректен, возвращаем ошибку (Ошибка 5).
6. Перед записью проверяем, существует ли email либо nickname в базе (дабы исключить вывод нескольких пользователей по запросам email или nickname). Если не уникальны, возвращаем ошибку (Ошибка 6).

Сообщения об ошибках.

Ошибка 1. Неподдерживаемый формат вводных данных.

```
{
    "error": "Unsupported format data"
}
```

Ошибка 2. Некорректное или отсутствующее поле.

```
{
    "error": "Missing or incorrect field"
}
```

Ошибка 3. Пустое или заполненное пробелами поле.

```
{
    "error": "Invalid or empty value: required all fields to fill"
}
```

Ошибка 4. В поле age введены строковые данные.

```
{
    "error": "Age must be an integer"
}
```

Ошибка 5. Неверный формат ввода email.

```
{
    "error": "invalid email"
}
```

Ошибка 6. Такой email или nickname уже существует.

```
{
    "error": "email or nickname already exists"
}
```

4. Метод **userGet**

Адрес: /userget метод: GET

Возвращает список найденных пользователей по колонкам nickname или email.

Запрос формируется в формате json. Пример запроса:

```
{
    "type": "nickname",
    "list": [
        "vito",
        "jigga",
        "Dub"
    ]
}
```

Ответ будет следующий:

```
{
    "type": "nickname",
    "list": [
        {
            "first_name": "Vito",
            "last_name": "Ferarri",
            "nickname": "vito",
            "email": "vferr@mail.com"
```

```
        },
        {
            "first_name": "Vasya",
            "last_name": "Jiguly",
            "nickname": "Jigga",
            "email": "jigvasya@mail.com"
        },
        {
            "first_name": "Ivan",
            "last_name": "Dubrovskiy",
            "nickname": "DuB",
            "email": "dubrovskiy.i@mail.com"
        }
    ]
}
```

Выполнение функции.

1. Тело запроса декодируется в php-массив. Если формат запроса неверен (не json) функция вернет false, условие не выполнится и будет возвращено сообщение об ошибке (Ошибка 1).
2. Далее проверяется наличие параметра type. Если параметр отсутствует, будет возвращено сообщение об ошибке (Ошибка 1).
3. Проверяется тип запроса (type). Если он не совпадает с допустимыми, условие не выполнится и будет возвращено сообщение об ошибке (Ошибка 1).
4. Проверяется наличие параметра списка данных (list). Если параметр отсутствует, будет возвращено сообщение об ошибке (Ошибка 2).
5. Далее ищутся совпадения в базе данных по значениям из списка (list), найденные значения записываются в массив и возвращаются пользователю.

Сообщения об ошибках:

Ошибка 1. Отсутствует параметр type или неверный формат данных:

```
{
    "error": "Unsupported or missing request type"
}
```

```
Ошибка 2. Отсутствует список данных запроса или пустой.
{
    "error": "Request is empty or missing"
}
```

5. Метод **loginAction**

Адрес: /user метод: PATCH

Проверяет совпадения логин(никнейм) и пароль в базе, генерирует/обновляет токен при совпадении.

Выполнение функции.

1. Из параметров запроса извлекается и проверяется на наличие и валидность name и password. Если проверка не пройдена, возвращается ошибка (Ошибка 1, Ошибка 2).
2. Отправляем запрос в базу. Если запись найдена, генерируем случайное бинарное значение и преобразуем в хэш. Если нет, возвращаем сообщение об ошибке (Ошибка 3).
3. Записываем хэш в соответствующую пользователю запись и выводим сообщение об успехе.

Сообщения об ошибках.

Ошибка 1. Отсутствует или некорректное значение name.

```
{
    "error": "Nickname field data invalid or is empty"
}
```

Ошибка 2. Отсутствует или некорректное значение name.

```
{
    "error": "Password field data invalid or is empty"
}
```

Ошибка 3. Неверный логин/пароль.

```
{
    "error": "Nickname or password not found"
}
```