**Задача**:

Протестировать функционал, задача которого создать пользователя и определить его возрастную группу.

Сервис работает по протоколу https, параметры передаются в теле сообщения в формате JSON

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **api** | **Тело** | **Ответ при успехе** |
| POST | http://host:80/CreateUser | {  "Name": "String"  } | 201 ОК  {  "id": "String"  } |
| GET | http://host:80/GetUser | {  "id": "String"  } | 200 ОК  {  "Name": "String",  "Age": int32,  } |
| POST | http://host:80/SetUserAge | {  "id": "String",  "Age ": Int32  } | 200 ОК |
| GET | http://host:80/{id}/GetAgeGroupById |  | 200 ОК  {  "AgeGroup": "String"  } |

Правило определения группы (AgeGroup):

Young – age от 0 до 18

Adult – age от 19 до 100

Здравствуйте!

Задание мне представляется как формат «свободного оформления» результатов с элементами креативной доработки.

Наличие реального сервера добавила бы широты рассуждения над процессом тестирования, особенно интересуют вопросы, которые у нас остаются автоматически за скобками, а вообще само задание можно даже рассмотреть с точки зрения ТЗ, оценить перспективы разработки, но в любом случае напрашиваются вопросы:

1. есть ли логирование и на каком уровне оно настроено, что пишется, после отправки api;
2. что с БД, если api выдает ошибку, но в БД есть запись;
3. зачем прописано https если ходить предлагается по http;
4. возможно, как вариант улучшения прописать документацию в качестве openapi;
5. возраст через int32, не очень понятно какая у этого цель…почему не int8, int16 или почему не как дата;
6. параметры в разном регистре- представляются очень спорным решением;
7. поле age в get-user опционально.

**Чек-лист проверок, разделяем по API**

1. Корректность структуры данных
2. Создание нового пользователя

**POST** http://host:80/CreateUser, значения для JSON “name”

* "Name":валидные данные
* "Name":повторно валидные данные (идемпотентность метода)
* "Name": пустое значение null (не заполнено)
* Валидация данных в полях (корректные и некорректные данные)
* Пустой JSON
* SQL injection

1. Проверка создания пользователя

**GET** http://host:80/GetUser

* Пустой список (если возможно)
* Запрос данных по валидному ID, проверка возвращения корректных данных
* Запрос данных по несуществующему ID, но в валидном формате
* Запрос данных по невалидному ID
* В случае передачи параметра с некорректным значением возвращается код ошибки (например 400) с описанием ошибки в теле ответа

1. Проверка создания возраста пользователя

**POST** http://host:80/SetUserAge

* "Аge":валидные данные по валидному ID
* "Аge":валидные данные по несуществующему ID
* "Аge":валидные данные по валидному ID, проверка регистрозависимости
* "Аge":проверка граничных значений и эквивалентных классов по валидному ID, int32
* Пустой JSON
* SQL injection

1. Проверка определения группы пользователя

**GET** http://host:80/{id}/GetAgeGroupById

* "ID":существующий ID
* "ID":несуществующий ID
* Пустое значение, null
* SQL injection