**Чек-лист для тестирования API (Swagger-Petstore)**

Чек-лист покрывает функциональные и негативные сценарии для API, включая методы GET, POST, PUT, DELETE.

Ссылка на API - <https://petstore.swagger.io/#/store>

**Endpoints**

**Секция Pet** — Все о вашем питомце.

**Endpoint:** GET /pet/{petId}

**Описание:** Возвращает данные питомца, если он существует, по petId.

**Параметры:** “petId” (str, required)

**Endpoint:** GET /pet/findByStatus

**Описание:** Возвращает список питомцев с указанным статусом

**Параметры: “**status” (str, required) - ["available", "pending", "sold"]

**Endpoint:** POST /pet/{petId}/uploadImage

**Описание:** Загружает изображение для питомца, по petId

**Параметры:** “petId” (int, required), “additionalMetadata” (str, optional), “file” (file, optional)

**Endpoint:** POST /pet

**Описание:** Создает нового питомца с указанными параметрами в теле запроса

**Параметры:** “body” (object, required)

**Endpoint:** POST /pet/{petId}

**Описание:** Обновляет данные существующего питомца, используя форму

**Параметры:** “petId” (int, required), “name” (str, optional), “ status” (str, optional)

**Endpoint:** PUT /pet

**Описание:**

**Параметры:**

**Endpoint:** DELETE /pet/{petId}

**Описание:** Удаляет питомца, по “petId”.

**Параметры:** “api\_key” (str, optional), “pet\_id” (int, required)

**Секция Store** — Доступ к заказам в магазине питомцев.

**Endpoint:** GET /store/inventory

**Описание:** Возвращает инвентарь питомца

**Параметры:** No params

**Endpoint:** GET /store/order/{orderId}

**Описание:** Поиск заказа на покупку по “orderId” питомца

**Параметры:** “orderId” (int, required)

**Endpoint:** POST /store/order

**Описание:** Оформить заказ на питомца.

**Параметры: “**body” (obj, required)

**Endpoint:** DELETE /store/order/{orderId}

**Описание:** Удалить заказ на покупку по “orderId”.

**Параметры:** “orderId” (int, required)

**Секция User** — Операции над пользователем.

**Endpoint:** GET /user/{username}

**Описание:** Получить информацию о пользователе по “username”.

**Параметры:** username (str, required)

**Endpoint:** GET /user/login

**Описание:** Вход пользователя в систему по параметрам - “username”, “password”.

**Параметры: “**username” (str, required), “password” (str, required)

**Endpoint:** GET /user/logout

**Описание:** Выход из текущего сеанса пользователя.

**Параметры:** No params

**Endpoint:** POST /user/createWithList

**Описание:** Создает список пользователей с заданным входным массивом содержащий конкретный набор параметров

**Параметры: “**body” (array[object] (id, username, firstName, lastName, email, password, phone, userStatus), required)

**Endpoint:** POST /user

**Описание:** Создает нового пользователя с указанными полями в теле, для пользователя, который уже вошел в систему

**Параметры: “**body” (obj(id, username, firstName, lastName, email, password, phone, userStatus), required)

**Endpoint:** PUT /user/{username}

**Описание:** Обновляет информацию о пользователе, для пользователя, который уже вошел в систему

**Параметры: “**username” (str, required), “body” (obj - (id, username, firstName, lastName, email, password, phone, userStatus), optional).

**Endpoint** DELETE /user/{username}

**Описание:** Удаляет пользователя, по username, для пользователя, который уже вошел в систему.

**Параметры:** “username” (str, required)

**Список проверок для каждого эндпоинта**

**Секция Pet**

**Endpoint:** GET /pet/{petID}

**Описание:** Возвращает данные питомца по указанному petId.

**Параметры: “**petId” (str, required)

**Positive:**

* Проверка корректного ответа для существующего “petId”. (например, “petId”=5)

**Negative:**

* Проверка ответа при запросе несуществующего “petId” (например, “petId”=9999).
* Проверка ответа при невалидном petId (например, “petId”=”abc”).
* Выполнение запроса без параметров.
* Изменить метод отправляемый на эндпоинт (например, PUT вместо GET).

**Endpoint:** GET /pet/findByStatus

**Описание:** Возвращает список питомцев с указанным статусом ("available", "pending", "sold").

**Параметры: “**status” (str, required) - ["available", "pending", "sold"]

**Positive:**

* Проверка корректного ответа для питомцев “status”=“available”.
* Проверка корректного ответа для питомцев “status”=“pending”.
* Проверка корректного ответа для питомцев “status”=“sold”.

**Negative:**

* Проверка ответа при несуществующем “status” (например, status=“test”).
* Проверка ответа при корректном “status”, но с добавлением пробелов до и после параметра (например, “status”=“ sold ”)
* Проверка ответа при пустом значении параметра “status” (например, status=””)
* Проверка ответа при невалидном “status” (например, “status”=1).
* Выполнение запроса без параметров.
* Изменить метод отправляемый на эндпоинт (например, PUT вместо GET)

**Endpoint:** POST /pet/{petId}/uploadImage.

**Описание:** Загружает изображение для питомца по “petId”.

**Параметры: “**petId”: int - required, “additionalMetadata”: str, “file”: file

**Positive:**

* Проверка корректного ответа для существующего “petId”, c добавлением параметров “additionalMetadata” и “file” (например, “petId”=1, “additionalMetadata”=”TestMeta”, “file”=”cat.png”)
* Проверка корректного ответа для существующего “petId”, c добавлением параметра “additionalMetadata” (например, “petId”=5, “additionalMetadata”=”Meta”)
* Проверка корректного ответа для существующего “petId”, с добавлением параметра “file” (например, “petId”=10, “file”=”dog.png”)
* Убедиться что file добавился к питомцу перейдя по эндпоинту GET /pet/{petID}

**Negative:**

* Проверка корректного ответа для существующего “petId”, без добавления дополнительных параметров (например, “petId”=12)
* Проверка ответа при несуществующем “petId”, с добавлением корректных параметров “additionalMetadata” и “file” (например, “petId”=9999, “additionalMetadata”=”New Meta”, “file”=”bird.jpg”)
* Проверка ответа при несуществующем “petId”, без добавления дополнительных параметров (например, “petId”=100000)
* Проверка ответа при невалидном petId, с добавлением невалидных параметров “additionalMetadata” и “file” (например, “petId”=”20”, “additionalMetadata”=12, “file”=3)
* Выполнение запроса без параметров
* Проверка загрузки файла неподдерживаемого формата (например, “file”=”script.exe”)
* Проверка загрузки файла превышающего допустимый размер
* Изменить метод отправляемый на эндпоинт (например, PUT вместо POST)

**Endpoint:** POST /pet

**Описание:** Создает нового питомца с указанными параметрами в теле запроса

**Параметры:** “body” (object, required)

**Positive:**

* Проверка корректного ответа при передаче валидного “body”, с заполнением всех полей (например, как в примере из Swagger <https://petstore.swagger.io/#/pet/addPet>)
* Проверка корректного ответа при передаче валидного “body”, содержащего:
* Несколько URL в поле “photoUrls” (например, [“url1”, “url2”]
* Несколько тегов в массиве “tags” (например, [{“id”: 1, “name”: “tag1”}, {“id”: 2, “name”: “tag2”}]

**Negative:**

1. **Проверка на отсутствие полей:**

* Проверка на отсутствие полей в “body”:
* Отсутствует поле “id”
* Отсутствует поле “category”
* Отсутствует поле “name”
* Отсутствует поле “photoUrls”
* Отсутствует поле “tags”
* Отсутствует поле “status”

1. **Невалидные типы данных:**

* “id” содержит строку вместо числа
* “category[id]” содержит строку вместо числа
* “name” содержит число вместо строки
* “photoUrls” содержит объект вместо вместо массива строк
* “tags” содержит строку вместо массива объектов
* “tags[].id” содержит строку вместо числа
* “status” содержит значение, не являющееся строкой (например, число)

1. **Некорректная структура “body”:**

* Полностью отсутствует тело запроса “body”
* Передача запроса с нарушением структуры вложенности
* Лишние поля в “body” (например, “extraField”=”Hello”)

1. **Граничные значения:**

* Поле “id” содержит отрицательное значение (например, -1)
* Поле “id” содержит максимально допустимое значение (например, 999999999)
* Поле “name” содержит максимально длинную строку (например, 1000 символов)
* Поле “photoUrls” содержит максимально допустимое количество URL (например, 1000 элементов)
* Поле “tags” содержит максимально допустимое количество объектов (например, 1000 объектов)

1. **Некорректные значения:**

* Поле “status” содержит значение, отличное от “available”, “pending”, или “sold” (например, “unknown”)
* Поле “tags[].id” содержит отрицательное значение.

1. **Некорректная структура Json:**

* Незакрытые кавычки в значении (например, “name”: “doggie)
* Пропущенная запятая между полями
* Неправильно закрытые скобки (например, вместо “}” использовать “]”)

1. **Изменить метод отправляемый на эндпоинт:**

* Выполнение запроса с методом GET вместо PОST

1. **Создание дубликата питомца:**

* Проверка ответа при передачи дубликата питомца в “body” (например, отправить запрос в “body”, содержащим идентичные данные питомца, который уже существует в системе (например, тот же “id”, “name”, “status”, “photoUrls”, и “tags”)

**Endpoint:** POST /pet/{petId}

**Описание:** Обновляет “name”, “status” питомца, по “petId”

**Параметры:** “petId” (int, required), “name” (str, optional), “ status” (str, optional) - ["available", "pending", "sold"]

**Positive:**

* Проверка корректного ответа при передачи валидного “petId”, с добавлением параметров “name”, “status” (например, ”petId”=10, “name”=”Sharik”, “status”=”sold”)
* Проверка корректного ответа при передачи валидного “petId”, с добавлением параметра “name” (например, “petId”=11, “name”=”Rex”)
* Проверка корректного ответа при передачи валидного “petId”, с добавлением параметра “status” (например, “petId”=3, “status”=”available”)
* Проверка корректного ответа при передачи валидного “petId”, без дополнительных параметров (например, “petId”=18)

**Negative:**

* Проверка ответа при передаче несуществующего “petId”, c добавлением корректных параметров “name”, “status” (например, “petId”=999999, “name”=”Vas’ka”, status=”sold”)
* Проверка ответа при передаче невалидного “petId”, c добавлением корректных параметров “name”, “status” (например, “petId”=”111”, name=”Tom”, “status”=”available”)
* Проверка ответа при передаче валидного “petId”, с добавлением невалидных параметров “name”, “status” (например, “petId”=1, “name”=12, “status”=”Unknown”)
* Выполнение запроса без параметров
* Проверка ответа при передаче валидного “petId”, c добавлением максимально длинного параметра “name” (например, “petId”=12, “name”=строка длиною 100 символов)
* Проверка ответа при передаче валидного “petId” c пустым значением параметра “name” (например, “petId”=12, “name”=””)
* Проверка ответа при передаче валидного “petId” c лишними проблема в параметре “status” (например, petId=”9”, status=” sold ”)
* Пропущенные кавычки в параметрах (например, “name”=Rex)
* Выполнение запроса с методом GET вместо POST

**Endpoint:** DELETE /pet/{petId}

**Описание:** Удаляет питомца, по “petId”.

**Параметры:** “api\_key” (str, optional), “petId” (int, required)

**Positive:**

* Проверка корректного ответа при передаче валидного “api\_key” и “petId”
* Проверка корректного ответа при передаче валидного “petId”, без параметра “api\_key”
* Убедиться что питомец c удаленным “petId” отсутствует в системе

**Negative:**

* Проверка ответа при передаче валидного “api\_key”, и несуществующего “petId” (например, “api\_key”=”valid api\_key”, petId=99999)
* Проверка ответа при передаче невалидного “api\_key”, и валидного “petId” (например, “api\_key”=”invalid api\_key”, “petId”=3)
* Проверка ответа при передаче некорректного “petId” (например, “petId”=”10”)
* Проверка ответа при передаче нуля для параметра “petId” (например, petId=0)
* Проверка ответа при передаче параметра “petId” как отрицательного числа (например, “petId”=-10)
* Проверка ответа при попытке повторного удаления одного и того же питомца (например, отправить запрос на удаление для уже удаленного “petId”)
* Выполнение запроса без параметров
* Выполнение запроса с методом POST вместо DELETE

**Секция Store**

**Endpoint:** GET /store/inventory

**Описание:** Возвращает инвентарь питомца

**Параметры:** No params

**Positive:**

* Проверка корректного ответа при запросе на эндпоинт

**Negative:**

* Проверить ответ при передаче дополнительного параметра (например, “extra\_field”=”extra\_value”)
* Выполнение запроса с методом POST вместо GET

**Endpoint:** GET /store/order/{orderId}

**Описание:** Поиск заказа на покупку по “orderId” питомца. Для валидного ответа orderId должен быть в пределах от 1 до 10 включительно, иначе вернется Exception

**Параметры:** “orderId” (int, required)

**Positive:**

* Проверка корректного ответа при передаче валидного “orderId” (например, “orderId”=5)
* Проверка корректного ответа при передаче минимального валидного “orderId” (например, “orderId”=1)
* Проверка корректного ответа при передаче максимально валидного “orderId” (например, “orderId”=10)

**Negative:**

* Проверка ответа на минимальной границе “orderId”=0
* Проверка ответа для отрицательного “orderId”=-1
* Проверка ответа на максимальной границе для “orderId”=11
* Проверка ответа на максимальной границе для “orderId”=9
* Проверка ответа при передаче несуществующего “orderId” (например, “orderId”=100000)
* Проверка ответа при передаче некорректного типа “orderId” (например, “orderId”=”Test”)
* Проверка ответа при передаче пустого значения “orderId” (например, “orderId”=””)
* Проверка ответа при передаче дополнительного параметра (например, orderId=14, “extra\_field”=”Test”)
* Выполнение запроса с методом PUT вместо GET

**Endpoint:** POST /store/order

**Описание:** Оформить заказ на питомца.

**Параметры: “**body” (obj, required)

**Positive:**

* Проверка корректного ответа при передаче валидного “body”, c заполнением всех полей (например, как в примере из Swagger <https://petstore.swagger.io/#/store/placeOrder>)
* Убедиться что заказ на питомца появился в системе, перейдя по эндпоинту GET /store/order/{orderId}

**Negative:**

* Проверка ответа при передаче валидного “body”, но поле “status”=”sold”
* Проверка ответа при передаче валидного “body”, но поле “status”=”pending”

1. **Проверка на отсутствие полей в “body”:**

* Отсутствует поле “id”
* Отсутствует поле “petId”
* Отсутствует поле “quantity”
* Отсутствует поле “shipDate”
* Отсутствует поле “status”
* Отсутствует поле “complete”

1. **Невалидные типы данных в “body”:**

* Поле “id” содержит строку вместо числа
* Поле “petId” cодержит строку вместо числа
* Поле “quantity” содержит строку вместо числа
* Поле “shipDate” содержит число вместо даты
* Поле “shipDate” содержит строку вместо даты
* Поле “status” содержит число вместо строки
* Поле “compete” содержит число вместо bool

1. **Некорректная структура “body”:**

* Полностью отсутствует тело запроса “body”
* Передача запроса с нарушением структуры вложенности
* Лишние поля в “body” (например, “extrafield”=”New field”)

1. **Граничные значения:**

* Поле “id” содержит отрицательное значения (например, “id”=-1)
* Поле “id” равно нулю
* Поле “id” содержит максимально допустимое значение (например “id”=99999)
* Поле “petId” содержит отрицательное значение (например, “petId”=-1)
* Поле “petId” равно нулю
* Поле “petId” содержит максимально допустимое значение (например, “petId”=100000)
* Поле “quantity” содержит отрицательное значение (например, “quantity”=-1)
* Поле “quantity” равно нулю
* Поле “quantity” содержит максимально допустимое значение (например, “quantity”=1000000)
* Проверка ответа при передаче валидного “body”, но поле “shipDate” раньше текущей даты (например, сегодня “01-01-25”, а “shipDate”=”31-12-24”)
* Проверка ответа при передаче валидного “body”, но поле “shipDate” сегодняшняя дата (например, сегодня “01-01-24”, “shipDate”=”01-01-24”)

1. **Некорректные значения:**

* Поле “status” содержит значение, отличное от “available”, “pending”, или “sold” (например, “unknown”)

1. **Некорректная структура Json:**

* Незакрытые кавычки в значении (например, “status”=“placed)
* Пропущенная запятая между полями
* Неправильно закрытые скобки (например, вместо “}” использовать “]”)

1. **Изменить метод отправляемый на эндпоинт:**

* Выполнить запрос с методом GET вместо POST

1. **Создание дубликата заказа:**

* Проверка ответа при передаче дубликата заказа в “body” (например, отправить запрос в “body”, содержащим идентичные данные заказа, который уже существует в системе (например, тот же “id”, “petId”, “quantity”, “shipDate”, “status”, и “complete”)

**Endpoint:** DELETE /store/order/{orderId}

**Описание:** Удалить заказ на покупку по “orderId”.

**Параметры:** “orderId” (int, required)

**Positive:**

* Проверка корректного ответа при передаче валидного “orderId”
* Убедиться что при успешном ответе, заказ отсутствует в системе перейдя по эндпоинту GET /store/order/{orderId}

**Negative:**

* Дублирующий запрос на удаление (например, “orderId” содержит уже удаленный до этого “orderId”)
* Проверка ответа при передаче отрицательного “orderId” (например, “orderId”=-1)
* Проверка ответа при передаче нуля (“orderId”=0)
* Проверка ответа при передаче максимально допустимого значения (например, “orderId”=1000000)
* Проверка ответа при передаче пустого значения (например, “orderId”=””)
* Поле “orderId” содержит строку вместо числа (например, “orderId”=”Hello!”)
* Выполнить запрос с пустыми параметрами
* Изменить метод запроса, GET вместо DELETE