API референсного билетного сервиса

Общая информация.

В данном документе приведено описание API билетного сервиса, который может быть использован в качестве референсного при реализации своего билетного шлюза партнёрами компании Ticketland. API сервиса спроектирован по подобию RESTful-сервисов. Кодировка обмениваемых сообщений - `UTF-8`. Обмен запросами и ответами производится по протоколу `HTTP(S)`. Всё описание API разделено на три основные группы: методы для работы с конструктивом, методы для работы с мероприятиями и методы для работы с билетами.

- Список изменений
- Запросы и ответы.
- Аутентификация и авторизация.
- Типы данных.
    - Дата/Время.
    - Дробное число.
- Сообщение об ошибке.
- Конструктив.
    - Метод constructive.
- Репертуар.
    - Метод repertoire.
    - Метод modifiedRepertoire.
- Билеты.
    - Схема продажи билета.
    - Метод tickets.
    - Метод lockTicket.
    - Метод unlockTicket.
    - Метод lockedTickets.
    - Метод createOrder.
    - Метод printableOrderData.
    - Метод confirmOrder.
    - Метод orderedTickets.
    - Метод removeOrder.
    - Метод returnTickets.
    - Метод salesReport.

Список изменений

Дата	Описание
13 января 2019	Добавлен отдельный метод возврата returnTickets билетов, согласно закону от 18.07.2019 №193-ФЗ В методе salesReport для операции возврата (return) поле price теперь возвращает возвращенную клиенту стоимость билета
26 марта 2015	Добавил информацию о зарезервированных кодах ошибок в разделе "Сообщение об ошибке".
10 апреля 2015	Добавил свойство "organizerId" в элемент show в примере ответа метода repertoire.
2 июня 2015	Добавил описание метода modifiedRepertoire.
4 июня 2015	Добавил свойство "time" в примеры запросов методов confirmOrder и removeOrder

Запросы и ответы.

Сервис должен поддерживать два типа запроса:

- `GET` - клиент хочет получить некоторые данные;
- `POST` - клиент хочет выполнить некоторую операцию, при которой может потребоваться передача данных в теле запроса;

В каждом `GET`-запросе от клиента, помимо прочего могут быть указаны следующие заголовки:

- `Accept` - указывает ожидаемый формат сообщения в теле ответа. Например, `application/json`. Реализация данного сервиса, как минимум, должна поддерживать обмен сообщениями в формате `JSON`. Данный заголовок является точкой расширения: в будущем, если понадобится, реализация сервиса может включить обмен сообщениями в формате `XML`. В таком случае, все, что понадобится сделать на клиентской стороне - это указать в запросе "`Accept: application/xml`". Клиент может не указать данный заголовок, в таком случае форматом по-умолчанию считается `JSON`.
- `Authorization` - аутентификационная информация клиента (см. Аутентификация и авторизация).

В каждом `POST`-запросе от клиента, могут быть указаны следующие заголовки:

- `Accept` (см. выше)
- `Content-Type` - указывает на формат сообщения, передаваемого в теле запроса. Реализация данного сервиса всегда должна быть готова принять запрос с сообщением в формате `JSON (application/json)`. Клиент может не указать данный заголовок, в таком случае форматом по-умолчанию считается `JSON`.
- `Authorization` - аутентификационная информация клиента (см. Аутентификация и авторизация).

Если название параметра в запросе заканчивается двумя квадратными скобками "[]" - это означает, что данный параметр может быть указан в запросе несколько раз. Пример: `GET http://www.example.com/constructive?segment[]=building&segment[]=hall`

В каждом ответе, помимо прочего, сервис должен возвращать следующие заголовки:

- `Content-Type` - указывает на формат сообщения, переданного в теле ответа.

Аутентификация и авторизация.

Для аутентификации клиента, в каждом запросе, помимо прочего, должен быть указан заголовок `Authorization`. Предлагается использовать простую Basic-аутентификацию. В сочетании с шифрованием канала связи, это даст минимально необходимый уровень безопасности. В конечном счете, выбор методики аутентификации остается за разработчиком данного сервиса. Клиенту остается лишь передавать в заголовке `Authorization` то значение, которое будет согласовано с владельцем данного сервиса.

Если клиент не предоставил данный заголовок, либо заголовок не содержит значения, то сервис должен вернуть ответ с кодом 401. Если предоставленные клиентом аутентификационные данные не верны, или же клиент не авторизован на выполнение запрошенной операции, то сервис должен вернуть ответ с кодом 403.

Типы данных.

Дата/Время.

Для обмена датой/временем предлагается использовать следующий формат:

```
yyyy-MM-ddTHH-mm-ss
```

yyyy - год (четыре цифры), MM - месяц (две цифры), dd - число (две цифры), HH - час (24-х часовая система, две цифры), mm - минута (две цифры), ss - секунда (две цифры).

В качестве тайм-зоны следует использовать некую оговоренную тайм-зону, согласованную представителями компании Ticketland и партнёра, реализующего данный сервис.

Пример: 2015-03-23T18-45-00

Дробное число.

Любое дробное число следует указывать в виде строки строго с двумя знаками после разделителя, в качестве которого выступает точка.

Пример: "250.00"

Сообщение об ошибке.

В случае любой ошибки, возникшей при выполнении запроса (за исключением ошибок, связанных с аутентификацией), сервис должен вернуть ответ с кодом 500, а в теле ответа должно быть сообщение, информирующее о произошедшей ошибке.

Пример сообщения (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "code": 300, // (целое число) код ошибки
 "message": "Не удалось создать заказ" // (строка) текстовое сообщение об ошибке
}
```

Следует иметь ввиду, что коды от 1 до 100 зарезервированы под определенные конкретные ситуации, предусмотренные интеграционной системой компании Ticketland. Эти коды будут перечислены здесь позже. Соответственно, для описания каких-то своих специфических ошибок, разработчик сервиса должен воспользоваться кодами в незарезервированном подмножестве от 101 и далее. Если же возникла ситуация (ошибка), упомянутая в описании зарезервированных кодов, то разработчик сервиса в этом случае должен вернуть зарезервированный код, соответствующий возникшей ситуации. Также, разработчик сервиса может использовать один и тот же код (от 101 и выше) для информирования о различных ситуациях. Например, код 300 может описывать все ошибки, связанные с заказом, код 400 - ошибки работы с репертуаром и так далее.

Конструктив.

Конструктив - это общее название всего, что связано с местом проведения мероприятия. Здание, зал (или сцена), секции и места, все это вместе - конструктив. Для получения информации о конструктиве в API представлен один метод: `constructive`.

Метод constructive.

Пример вызова: `GET http://www.example.com/constructive?hallId=15&hallVersion=2442&segment[]=building&segment[]=hall&segment[]=section&segment[]=place`

Входные параметры:

- `hallId` (строка) - идентификатор зала (сцены), в котором проходит мероприятие.
- `hallVersion` (строка) - версия этого зала (сцены). Как известно, в одном и том же зале могут проходить совершенно различные мероприятия, причем, нередко, рассадка и количество мест в этом зале различается для разных мероприятий. Например, для зала с идентификатором 15 может быть 10 версий различных схем рассадок. Следовательно, конкретную версию рассадки в конкретном зале всегда можно определить парой атрибутов: `hallId` и `hallVersion`. Атрибут `hallVersion` должен быть уникальным в рамках своего зала и однозначно определять свой уникальный вариант рассадки в этом зале.
- `segment[]` (строка) - указание, какой (или какие) сегмент конструктива вызывающая сторона желает получить в ответе. Возможные значения: `building` - информация о здании, в котором расположена сцена, `hall` - информация о самом зале (сцене), `section` - информация о секциях в зале и `place` - информация о местах в зале. Как видно из примера, параметр `segment[]` может быть указан несколько раз, это следует трактовать как желание вызывающей стороны получить несколько сегментов конструктива.

Параметры `hallId` и `hallVersion` не являются обязательными, однако, всегда должны быть указаны либо в паре, либо не указаны вовсе. Если они не указаны, значит в ответе ожидается информация по всем залам, с учетом указанных `segment[]`. В свою очередь параметр `segment[]` должен быть указан как минимум один раз. Если параметр `segment[]` не был указан ни разу, то сервис может завершить обработку запроса с ошибкой.

Пример ответа:

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
   /*
  Сегмент "buildings": массив объектов, в каждом из которых описано отдельное здание. Данный сегмент должен присутствовать в ответе, если в запросе был указан параметр segment[] со значением building.
  Как правило, в театрах, обычно, одно здание.
  Если в запросе были указаны параметры hallId и hallVersion, то данный массив должен включать в себя лишь один объект, описывающий здание, в котором находится запрашиваемый зал.
 */
   "buildings": [
      {
         "id": "1", // (строка) идентификатор здания (театра); значение должно быть уникальным в пределах всех зданий в базе данных
         "name": "Большой Театр" // (строка) название здания (театра)
      }
```

```
  ],

 /*
  Сегмент "halls": массив объектов, каждый из которых описывает свойства зала (сцены). Данный сегмент должен присутствовать в ответе, если в запросе был указан параметр segment[] со значением hall.
  Если в запросе были указаны параметры hallId и hallVersion, то данный массив должен включать в себя лишь один объект, описывающий запрошенный зал.
 */
   "halls": [
      {
         "id": "15", // (строка) идентификатор зала; значение должно быть уникальным в пределах всех залов в базе данных
         "name": "Основная сцена", // (строка) название зала
  "printName": "Основная сцена", // (строка, не обязательно) название зала на печатаемом билете, в случае, если оно отличается от значения свойства name.
         "buildingId": "1" // (строка) идентификатор здания, в котором расположен зал
      },
      {
         "id": "23",
         "name": "Малая сцена",
  "printName": "Малая сцена",
         "buildingId": "1"
      }
   ],


 /*
  Сегмент "sections": массив объектов, каждый из которых описывает свойства секции. Данный сегмент должен присутствовать в ответе, если в запросе был указан параметр segment[] со значением section.
  Если в запросе были указаны параметры hallId и hallVersion, то данный массив должен включать в себя лишь те секции, которые включены в запрашиваемый зал.
 */
   "sections": [
      {
         "id": "4053", // (строка) идентификатор секции; значение должно быть уникальным в пределах всех секций в базе данных
         "name": "Левая сторона", // (строка) название секции
         "printName": "Лев. сторона", // (строка, не обязательно) название секции на печатаемом билете, в случае, если оно отличается от значения свойства name.
  /*
   Массив объектов, каждый из которых описывает координату точки в двумерной системе координат; не обязательное, но весьма желательное свойство, особенно для больших залов.
   Начало координат в левом верхнем углу. С помощью указанных точек будет изображена фигура, повторяющая контуры секции. может быть указано любое количество точек, но не менее трех.
  */
         "coordinates": [
            {
    "x": 10, // (целое число) координата точки по оси X
    "y": 20 // (целое число) координата точки по оси Y
   },
            {"x": 20, "y": 20},
            {"x": 10, "y": 30},
            {"x": 15, "y": 40},
            {"x": 20, "y": 30}
         ]
      },
      {
         "id": "4055",
         "name": "Правая сторона",
         "printName": "Прав. сторона",
```

```
            "coordinates": [
                {"x": 70, "y": 20},
                {"x": 80, "y": 20},
                {"x": 70, "y": 30},
                {"x": 75, "y": 40},
                {"x": 80, "y": 30}
            ]
        },
        {
            "id": "4079",
            "name": "Амфитеатр",
            "printName": "Амфитеатр",
            "coordinates": [
                {"x": 50, "y": 30},
                {"x": 60, "y": 30},
                {"x": 50, "y": 40},
                {"x": 60, "y": 40}
            ]
        }
    ],

 /*
  Данный сегмент является вспомогательным и должен автоматически появиться в ответе, если в запросе были указаны параметры hallId и hallVersion.
    Этот сегмент представляет из себя массив из одного объекта, описывающего связь между конкретной версией запрошенного зала и секциями, перечисленными выше.
 */
    "hallVersions": [
        {
            "hallId": "15", // (строка) идентификатор запрошенного зала
            "hallVersion": "2442", // (строка) версия запрошенного зала (см. описание входящего параметра hallVersion)
            "sectionIds": ["4053", "4055"] // массив строк, каждая из которых является идентификатором секции, включенной в запрашиваемый зал; может быть пустым, но это лишено смысла
        }
    ],

 /*
  Сегмент "places": массив объектов, каждый из которых описывает свойства места. Данный сегмент должен присутствовать в ответе, если в запросе был указан параметр segment[] со значением place.
  Если в запросе были указаны параметры hallId и hallVersion, то данный массив должен включать в себя лишь те места, которые включены в лишь те секции, которые включены в запрашиваемый зал.
 */
    "places": [
        {
            "id": "20048", // (строка) идентификатор места; значение должно быть уникальным в пределах всех мест в базе данных
            "sectionId": "4053", // (строка) идентификатор секции, в которую включено данное место
            "row": "3", // (строка) ряд
            "rowMetric": "Ряд", // (строка, не обязательно) название ряда на печатаемом билете; если не указано, будет использовано название по усмотрению компании Ticketland.
            "seat": "10", // (строка) место
            "seatMetric": "Место", // (строка, не обязательно) название места на печатаемом билете; если не указано, будет использовано название по усмотрению компании Ticketland.
    /*
     Координата места в двумерной системе координат. Начало координат в левом верхнем углу. Свойство является обязательным для тех мест, которые должны быть отображены на схеме зала.
    */
            "coordinate": {
     "x": 10, // (целое число) координата точки по оси X
```

```
 "y": 20 // (целое число) координата точки по оси Y
}
    },
    {
      "id": "30042",
      "sectionId": "4055",
      "row": "4",
      "rowMetric": "Линия",
      "seat": "12",
      "seatMetric": "Кресло",
      "coordinate": {"x": 20, "y": 30}
```

```
    }
  ]
}
```

Репертуар.

Под репертуаром подразумевается информация обо всех мероприятиях, о которых известно билетной системе. Для получения информации о мероприятиях в API представлен один метод: `repertoire`.

Метод repertoire.

Пример вызова: `GET http://www.example.com/repertoire?fromInclusive=2015-03-23T00-00-00&tillExclusive=2015-05-23T00-00-00`

Входные параметры:

- `fromInclusive` (дата/время, не обязательно) - минимально возможное время (включительно) начала мероприятий, информация о которых будет представлена в ответе; мероприятия могут начинаться от указанного времени; если не указано, то это следует трактовать как отсутствие необходимости фильтрации отдаваемых мероприятий по данному критерию.
- `tillExclusive` (дата/время, не обязательно) - максимально возможное время (исключительно) начала мероприятий, информация о которых будет представлена в ответе; мероприятия должны начинаться строго раньше указанного времени; если не указано, то это следует трактовать как отсутствие необходимости фильтрации отдаваемых мероприятий по данному критерию.

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 /*
  Сегмент "organizers": массив объектов, каждый из которых описывает организаторов шоу, приведенных в ответе (ниже). Здесь должны быть перечислены только те организаторы, на которых ссылаются шоу, приведенные в следующем сегменте.
 */
   "organizers": [
      {
         "id": "500", // (строка) идентификатор организатора; значение должно быть уникальным в пределах всех организаторов в базе данных
         "name": "ООО 'Лучшие спектакли'", // (строка) название организатора
      },
      {
         "id": "510",
         "name": "ООО 'Рога и Копыта'"
      }
   ],

 /*
  Сегмент "shows": массив объектов, каждый из которых описывает шоу, в рамках которых будут проходить мероприятия, приведенные в ответе (ниже). Здесь должны быть перечислены только те шоу, на которые ссылаются мероприятия, приведенные в следующем сегменте.
 */
   "shows": [
      {
         "id": "1000", // (строка) идентификатор шоу; значение должно быть уникальным в пределах всех шоу в базе данных
         "name": "Щелкунчик", // (строка) название шоу
         "type": "Балет", // (строка) тип шоу
         "minAge": 12, // (целое число, не обязательно) минимально допустимый возраст зрителя
```

```
  "organizerId": "500" // (строка) идентификатор организатора
      },
      {
        "id": "1002",
        "name": "Ромео и Джульетта",
        "type": "Опера",
        "minAge": 16,
  "organizerId": "510"
      }
   ],

 /*
  Сегмент "performances": массив объектов, каждый из которых описывает конкретное мероприятие, проходящее в рамках своего шоу.
 */
    "performances": [
      {
        "id": "20048", // (строка) идентификатор мероприятия; значение должно быть уникальным в пределах всех мероприятий в базе данных
        "hallId": "15", // (строка) идентификатор зала, в котором проходит мероприятие
        "hallVersion": "2442", // (строка) версия зала
        "showId": "1000", // (строка) идентификатор шоу, в рамках которого проходит мероприятие
        "beginTime": "2015-05-28T18-00-00" // (дата/время) время начала мероприятия
      },
      {
        "id": "20059",
        "hallId": "15",
        "hallVersion": "2442",
        "showId": "1002",
        "beginTime": "2015-04-14T20-00-00"
```

```
    }
  ]
}
```

Метод modifiedRepertoire.

Метод предназначен для получения списка мероприятий, в которых "что-то изменилось" с точки зрения билетного сервиса. Данный метод призван уменьшить нагрузку на сеть и вычислительные ресурсы при постоянном запрашивании (polling) информации о мероприятиях у билетного сервиса. С мена названия мероприятия, даты или времени его проведения, количества оставшихся свободных мест - всё это может быть причиной, по которой идентификатор мероприятия попадает в результат вызова данного метода. В конечном счете, разработчик билетного сервиса сам определяет список факторов, позволяющих мероприятию считаться "модифицированным". Этот метод не является обязательным для реализации. В рамках подготовки к интеграции с разрабатываемым билетным сервисом его разработчик должен сообщить о том, был ли им реализован данный метод или нет.

Пример вызова: `GET http://www.example.com/modifiedRepertoire?modificationTag=hx982c3d331f`

Входные параметры:

- `modificationTag` (строка, не обязательно) - метка, возвращенная билетным сервисом в результате последнего вызова данного метода; разработчик сервиса определяет, чем будет эта метка: строка с датой/временем, например "2015-06-02T12:34:45" или же, некий уникальный идентификатор последней транзакции в базе данных билетного сервиса, например "2312da42f" и т.д. В любом случае, следует понимать, что в этом параметре передается та информация, которая, во-первых, будет понятна билетному сервису и, во-вторых, будет им интерпретирована как некая "метка", которую следует учесть при определении "модифицированности" того или иного мероприятия, идентификаторы которых будут перечислены в результате вызова данного метода. Также следует иметь ввиду, что если данный аргумент не был указан при вызове метода, то в результате вызова метода должны быть перечислены идентификаторы всех мероприятий, доступных к продаже. Очевидно, что данный аргумент не указывается, например, при первом вызове данного метода, когда вызывающая сторона элементарно "не знает" какой `modificationTag` следует передать.

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 /*
  Метка, которую билетная система готова получить при следующем вызове данного метода. На основании ЭТОЙ метки, которая будет передана в СЛЕДУЮЩЕМ вызове, билетная система сможет принять решение о том, какие мероприятия изменились с момента завершения ТЕКУЩЕГО вызова данного метода.
 */
   "modificationTag": "20150602_194452", // (строка)

 /*
  Массив идентификаторов мероприятий, изменившихся с момента, указанного в аргументе modificationTag, переданного при при вызове данного метода. Массив может быть пустым, в таком случае, предполагается, что ни одно мероприятие не изменилось.
 */
   "performances": ["20048", "20049"] // (массив строк)
}
```

Билеты.

Работу с билетами можно разделить на две части: импорт информации о свободных билетах и выполнение билетных операций. Здесь сразу следует пояснить, что у билета составной идентификатор. Он состоит из двух идентификаторов: `performanceId` (идентификатор мероприятия) и `placeId` (идентификатор места в зале). С помощью этих двух идентификаторов всегда можно четко обозначить продаваемый билет. При всех операциях с билетами будут передаваться эти два идентификатора.

Для получения информации о свободных билетах в API представлен метод `tickets`.

Схема продажи билета.

Для продажи билета сначала его следует заблокировать с помощью метода `lockTicket`. Данный метод возвращает идентификатор корзины (в которую помещен билет). Этот идентификатор, затем, следует передать в метод `createOrder`, что бы создать заказ на основе корзины. После этого будет вызван метод `printableOrderData`, что бы получить данные для печати билетов, входящих в этот заказ. В завершении (после получения денег от покупателя) будет вызван метод `confirmOrder`, означающий подтверждение (выкуп) заказа.

Для удаления билета из корзины вызывается метод `unlockTicket`. Для удаления заказа (независимо от того, был он подтвержден или нет) вызывается метод `removeOrder`.

Метод tickets.

Пример вызова: `GET http://www.example.com/tickets?performanceId=20048`

Входные параметры:

- `performanceId` (строка) - идентификатор мероприятия, информацию о свободных билетах которого требуется вернуть.

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 /*
  Сегмент "tickets": массив объектов, каждый из которых описывает билет, доступный к продаже.
 */
   "tickets": [
      {
         "placeId": "20048", // (строка) идентификатор места в зале
         "performanceId": "20059", // (строка) идентификатор мероприятия
         "price": "250.55" // (дробное число) цена билета
      },
      {
         "placeId": "20049",
         "performanceId": "20059",
         "price": "100.00"
      }
   ]
}
```

Метод lockTicket.

Добавление билета в корзину. Если билет уже находится в этой или другой корзине, то метод должен вернуть сообщение об ошибке.

Пример вызова: `POST http://www.example.com/lockTicket`

Пример запроса (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "performanceId": "20059", // (строка) идентификатор мероприятия
 "placeId": "20049", // (строка) идентификатор места в зале
 "basketId": "a34b3498f928c" // (строка, не обязательно) идентификатор корзины. Если не задан, то это следует трактовать как первое добавление билета в корзину, в таком случае, должна быть создана новая корзина и её идентификатор должен быть возвращен в ответном сообщении. Если же данный параметр указан, то билет должен быть добавлен в существующую корзину.
}
```

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "basketId": "a34b3498f928c", // (строка) идентификатор корзины, в которую добавлен билет
 "ttlInSeconds": 900 // (целое число) время (в секундах) в течении которого данный билет будет находиться в заблокированном состоянии до того, как билет будет разблокирован автоматически
}
```

Метод unlockTicket.

Удаление билета из корзины. Если билет уже был удален из корзины ранее, метод должен отработать идемпотентно и вернуть такой же результат, как если бы билет присутствовал в корзине и был успешно удален из неё.

Пример вызова: `POST http://www.example.com/unlockTicket`

Пример запроса (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "performanceId": "20059", // (строка) идентификатор мероприятия
 "placeId": "20049", // (строка) идентификатор места в зале
 "basketId": "a34b3498f928c" // (строка) идентификатор корзины.
}
```

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{/*
 Просто пустой JSON.
 Если на момент вызова метода билета не было в корзине ответ также должен представлять из себя пустой JSON.
*/}
```

Метод lockedTickets.

Получение списка билетов, находящихся в корзине. Если корзина с указанным идентификатором не существует, следует вернуть сообщение об ошибке.

Пример вызова: GET `http://www.example.com/lockedTickets?basketId=a34b3498f928c`

Входные параметры:

- `basketId` (строка) - идентификатор корзины.

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "tickets": [ // массив объектов, каждый из которых описывает билет, находящийся в корзине; может быть пустым.
  {
   "performanceId": "20059", // (строка) идентификатор мероприятия
   "placeId": "20049" // (строка) идентификатор места
  }
 ]
}
```

Метод createOrder.

Создание заказа на основе корзины. Метод может вернуть сообщение об ошибке только в случае, если, вообще, не удалось создать заказ в силу каких-то ограничений или ошибок, например, был указан неверный идентификатор корзины. Если же при создании заказа возникли ошибки лишь с некоторыми билетами, входящими в корзину, то метод должен проинформировать об этом в теле своего ответа (см. пример ниже).

Пример вызова: `POST http://www.example.com/createOrder`

Пример запроса (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "basketId": "a34b3498f928c", // (строка) идентификатор корзины.
 "customer": { // Информация о покупателе. Данное свойство может отсутствовать в запросе.
  "id": "4991", // (строка) идентификатор покупателя
  "surname": "Сидоров", // (строка, не обязательно) фамилия покупателя
  "name": "Иван", // (строка, не обязательно) имя покупателя
  "patronymic": "Петрович", // (строка, не обязательно) отчество покупателя
  "phone": "79250000000", // (строка, не обязательно) номер телефона покупателя
  "email": "sidorov@example.com" // (строка, не обязательно) адрес электронной почты покупателя
 },
 "ticketExtras": [ // массив объектов с дополнительной информацией о билетах, входящих в корзину, на основании которой будет создан заказ; может быть пустым.
  {
   "performanceId": "20059", // (строка) идентификатор мероприятия
   "placeId": "20049", // (строка) идентификатор места
   "price": "100.00" // (дробное число) цена билета по данным клиента; в случае расхождения этой цены с данными сервиса, он может заблокировать создание заказа или заблокировать добавление билета в создаваемый заказ
  }
 ]
}
```

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "orderId": "10002", // (строка) идентификатор заказа
 "ttlInSeconds": 172800, // (целое число) время (в секундах), в течении которого заказ можно подтвердить (оплатить). По истечении этого времени, заказ может быть автоматически аннулирован.
 "tickets": [ // массив с информацией о билетах, вошедших (или тех, которые должны были войти) в заказ; каждому билету в корзине должен соответствовать один элемент в этом массиве
  {
   "performanceId": "20059", // (строка) идентификатор мероприятия
   "placeId": "20049", // (строка) идентификатор места
   "error": { // информация о причине (ошибке) по которой билет не вошел в заказ; если билет попал в заказ, то данное свойство должно отсутствовать.
    "code": 105, // (целое число) код ошибки
    "message": "Несовпадение цены" // (строка) текстовое сообщение об ошибке
   }

  }
 ]

}
```

Метод printableOrderData.

Получение данных для печати заказа. Фактически, данный метод должен быть вызван до подтверждения (оплаты) заказа, непосредственно перед печатью билетов, входящих в заказ. Если заказа не существует, метод должен вернуть сообщение об ошибке.

Пример вызова: GET `http://www.example.com/printableOrderData?orderId=10002`

Входные параметры:

- `orderId` (строка) - идентификатор заказа.

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "tickets": [ // массив с информацией, требуемой для печати каждого билета; каждому билету в заказе должен соответствовать один элемент в этом массиве
  {
   "performanceId": "20059", // (строка) идентификатор мероприятия
   "placeId": "20049", // (строка) идентификатор места
   "barcode": { // информация о штрих-коде билета; может отсутствовать, если не удалось её подготовить, но в таком случае, должна быть указана причина в свойстве "error"
    "value": "29384742312031", // (число, выраженное в виде строки) значение штрих-кода
    "type": "interleaved_2_of_5" // (строка, не обязательно) тип штрих-кода; в случае отсутствия, в качестве типа будет принято значение "interleaved_2_of_5"; на данный момент компания Ticketland будет трактовать любой штрих-код как "interleaved_2_of_5", добавляя слева цифру "0", если в значении штрих-кода указано нечетное количество цифр.
   },
   "error": { // информация о причине (ошибке) по которой не удалось подготовить данные для печати билета; если ошибок нет, то данное свойство должно отсутствовать
    "code": 150, // (целое число) код ошибки
    "message": "Штрих-код не найден" // (строка) текстовое сообщение об ошибке
   }
  }
 ]

}
```

Метод confirmOrder.

Подтверждение (оплата) заказа. После вызова этого метода заказ считается проданным. Это финальное состояние успешного жизненного цикла заказа. Метод может вернуть сообщение об ошибке только в случае, если, вообще, не удалось подтвердить заказ целиком в силу каких-то ограничений или ошибок. Если же при подтверждении заказа возникли ошибки лишь с некоторыми билетами, входящими в него, то метод должен проинформировать об этом в теле своего ответа (см. пример ниже).

Пример вызова: POST `http://www.example.com/confirmOrder`

Пример запроса (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "orderId": "10002", // (строка) идентификатор заказа
 "time": "2015-03-23T15-45-23" // (дата/время) время "на часах" вызывающей стороны на момент отправки запроса
}
```

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "tickets": [ // массив с информацией о билетах, входящих в заказ
  {
   "performanceId": "20059", // (строка) идентификатор мероприятия
   "placeId": "20049", // (строка) идентификатор места
   "error": { // информация о причине (ошибке) по которой не удалось подтвердить покупку данного конкретного билета;
если подтверждение прошло успешно, то данное свойство должно отсутствовать
    "code": 250, // (целое число) код ошибки
    "message": "Билет не найден" // (строка) текстовое сообщение об ошибке
   }
  }
 ]
}
```

Метод orderedTickets.

Получение списка билетов, входящих в заказ. Если заказ с указанным идентификатором не найден, следует вернуть сообщение об ошибке. Этот метод может быть вызван как до подтверждения заказа так и после.

Пример вызова: GET `http://www.example.com/orderedTickets?orderId=10002`

Входные параметры:

- `orderId` (строка) - идентификатор заказа.

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "tickets": [ // массив объектов, каждый из которых описывает очередной билет, входящий в заказ
  {
   "performanceId": "20059", // (строка) идентификатор мероприятия
   "placeId": "20049" // (строка) идентификатор места
  }
 ]
}
```

Метод removeOrder.

Удаление заказа. Данный метод может быть вызван как для неподтвержденного, так и для подтвержденного заказа и в обоих случаях должен работать одинаково, приводя к полному удалению заказа. Метод может вернуть сообщение об ошибке только в случае, если, вообще, не удалось удалить заказ целиком в силу каких-то ограничений или ошибок. Если же при удалении заказа возникли ошибки лишь с некоторыми билетами, входящими в него, то метод должен проинформировать об этом в теле своего ответа (см. пример ниже). Если на момент вызова метода заказ уже был удален, то метод должен отработать идемпотентно и вернуть такой же результат, как если бы заказ существовал и был успешно удален.

Пример вызова: `POST http://www.example.com/removeOrder`

Пример запроса (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "orderId": "10002", // (строка) идентификатор заказа
 "time": "2015-03-23T15-45-23" // (дата/время) время "на часах" вызывающей стороны на момент отправки запроса
}
```

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 /*
  Массив объектов, каждый из которых описывает билет, с которым возникла проблема в ходе удаления заказа.
  Если при удалении заказа, с очередным билетом не возникло никаких проблем, то информация об этом билете НЕ
ДОЛЖНА попасть в данный массив.
  Если на момент вызова метода заказа уже не существовало, то данный массив должен быть пустым.
 */
 "tickets": [
  {
   "performanceId": "20059", // (строка) идентификатор мероприятия
   "placeId": "20049", // (строка) идентификатор места
   "error": { // информация о причине (ошибке) по которой не удалось удалить данный билет из заказа
    "code": 350, // (целое число) код ошибки
    "message": "Невозвратный билет" // (строка) текстовое сообщение об ошибке
   }
  }
 ]
}
```

Метод returnTickets.

Возврат билетов. Данный метод может быть вызван только для подтвержденного заказа. Позволяет вернуть как все билеты, так и отдельные. Метод может вернуть сообщение об ошибке по каждому отдельному билету. Может вызываться несколько раз для одного заказа, пока в заказ содержит невозвращенные билеты. Если на момент вызова метода билет возвращен, то метод должен отработать идемпотентно и вернуть такой же результат, как если бы билет еще не был возвращен и был успешно возвращен.

Пример вызова: POST http://www.example.com/returnTickets

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
  "orderId": "10002", // (строка) идентификатор заказа
  "time": "2015-03-23T15-45-23", // (дата/время) время "на часах" вызывающей стороны на момент отправки запроса
  "tickets": [ // массив объектов, каждый из которых описывает возвращаемый билет
    {
      "performanceId": "20059", // (строка) идентификатор мероприятия
      "placeId": "20049", // (строка) идентификатор места
      "price": "100.00", // (дробное число) цена билета на момент выполнения операции
      "returnPrice": "50.00" // (дробное число) стоимость билета возвращаемая клиенту, согласно закону от 18.07.2019 №193-ФЗ
    }
  ]
}
```

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
    /*
        Массив объектов, каждый из которых описывает билет, с которым возникла проблема в ходе удаления заказа.
        Если при возврате билетов, с очередным билетом не возникло никаких проблем, то информация об этом билете НЕ ДОЛЖНА попасть в данный массив.
        Если на момент вызова метода билет был уже возвращен, то данный массив должен быть пустым.
    */
    "tickets": [
        {
            "performanceId": "20059", // (строка) идентификатор мероприятия
            "placeId": "20049", // (строка) идентификатор места
            "error": { // информация о причине (ошибке) по которой не удалось удалить данный билет из заказа
                "code": 350, // (целое число) код ошибки
                "message": "Невозвратный билет" // (строка) текстовое сообщение об ошибке
            }
        }
    ]
}
```

Метод salesReport.

Получение отчета о продажах билетов. Данный метод вызывается для проведения сверки продаж по данным системы компании Ticketland и данным реализуемого сервиса с целью поиска и корректировки возможных расхождений. Идеологически, этот метод относится к API для работы с билетами.

Пример вызова: GET `http://www.example.com/salesReport?fromInclusive=2015-03-23T00-00-00&tillExclusive=2015-03-24T00-00-00`

Входные параметры:

- `fromInclusive` (дата/время) - время начала (включительно) интересующего периода построения отчета
- `tillExclusive` (/) - ()

Пример ответа (с пояснениями):

```
/* ВСЕ АТРИБУТЫ ОБЯЗАТЕЛЬНЫ, ЕСЛИ НЕ УКАЗАНО ИНОЕ */

{
 "tickets": [ // массив объектов, каждый из которых описывает информацию об очередном билете с которым была выполнена та или иная операция (продажа, возврат)
  {
   "performanceId": "20059", // (строка) идентификатор мероприятия
   "placeId": "20049", // (строка) идентификатор места
   "operationTime": "2015-03-23T15-45-23", // (дата/время) время выполнения операции с билетом
   "operationType": "sale", // тип выполненной операции; возможные значения "sale" - продажа билета, "return" - возврат билета; в течении запрашиваемого периода один и тот же билет может выкупаться и возвращаться несколько раз.
   "price": "250.00" // (дробное число) цена билета на момент выполнения операции
  },
  {
   "performanceId": "20059",
   "placeId": "20050",
   "operationTime": "2015-03-23T18-19-03",
   "operationType": "return",
   "price": "3200.00" // (дробное число) стоимость билета возвращенная клиенту
  }
 ]
}
```