OpenActive Open Booking Standard v0.3 (draft)

This document represents a **minimal early draft** of a standard for booking physical activities. It is highly likely to change completely between now and the first published version, so this document should only be used to guide swift implementations where the quick win of booking proof-of-concepts and shared learning are high priorities.

Note this has been superseded by the following:
https://app.swaggerhub.com/apis/theodi/openactive-booking-draft/0.4

Scope

The scope of this specification is to cater for the "v1" requirements included in this Use Cases and Requirements document in minimal form. These are summarised as follows:

V1 Goals:

- Improvements to RPDE feed to enable booking
- Get price and availability for a specific event
- Paid for or free bookings of sessions
- Bookings made as new or returning accounts, with minimal user data required
- Guest booking, where a booking is made which does not create an account
- Facilitate integration with any third party payment processor
- Identify the platform during the booking
- Provide a mechanism to make terms & conditions available, and acknowledge consent during booking.
- Two-phase commit booking (i.e. "lease" and "book")
- Booking of sessions and facilities. While a **session** can be a yoga class, organised run or any other kind of physical activity happening at a specific place at specific times, a **facility** is a squash court, football pitch or any other venue that can be booked out at any time
- Include "Book multiple events" requirement in design thinking, but not in the specification

V1 Non-Goals:

- API Authentication mechanism
- Payment processing - payments can be offloaded to the client, which can then notify the provider of transaction and credit them accordingly
- Shopping baskets
- Special pricing models for existing memberships
- Waiting lists
- Group booking. This is a single customer booking multiple spaces for friends or family
- Coverage of every single possible use case of booking

OpenActive

Objectives

- Easy to understand
- Simple to implement (does not require complex libraries)
- Minimalistic (focus on removing complexity)
- Use existing standards where possible
- Allow or a checkout experience that is as seamless as possible

Concepts

Account Creation

In order to create an account via the Booking Standard, a customer must provide details. There is a minimum set of details required for all providers as well as the ability for providers to require extra fields from a set defined below

Minimum required fields

- emailAddress

Extra fields - following this standard

- givenName
- familyName

GDPR

In order to conform to GDPR accounts that were created during unsuccessful bookings must not be persisted. For this reason, the booking standard mandates that newly created accounts start with an expiry of **30 minutes**. If the account does not successfully complete a booking before the expiry has elapsed, the account *must* be deleted by the provider. Suggested mechanisms for this deletion are provided in the “Other Processes” section

Event

A bookable session can occur on multiple dates. For example, a weekly yoga session could occur every Monday for a month. Each of these occurrences is called a subEvent, which inherits the properties of its parent Event.

An Event can have multiple spaces, each of which is available to be booked.

Broker

The third party organisation hosting the booking user experience (e.g. Change4Life). See schema.org definition.

Customer

The end user making the purchase. See schema.org definition.

Seller

The activity provider offering the activities. See schema.org definition.

Implementation

Errors

All errors conform to RFC 7807 https://tools.ietf.org/html/rfc7807, and have content type "application/problem+json"

Workflow

The following is a list of actions which need to implemented in order to make a successful booking. Each action is documented with its associated REST endpoint, example requests and response objects where necessary. These objects are in JSON format.

Authentication

Note that all API requests specified in this document (including Get Latest Event Data) should require authentication. In the absence of an existing authentication pattern, HTTP authentication is recommended.

1. Get Latest Event Data

For a given Event, get the latest data as it appears in the RPDE feed.

In order for an Event or SubEvent to be considered "bookable", it must contain the following:

Property	Status	Notes
id	required	The URI which may be used to retrieve a subset of the latest Event object
identifier	required	
startDate	required	
endDate	optional	

remainingAttendeeCapacity	optional	If not specified unlimited availability is assumed
offer	At least one “offer” is required	An array of offers
isAccessibleForFree	optional	Boolean
duration	optional	
maximumAttendeeCapacity	optional	If not specified unlimited availability is assumed
beta:orderPostUrl	required	The post endpoint to use to create an order, e.g. `http://www.book.com/api/orders`
beta:anonymousLeaseDuration	optional	Duration of extended lease available given in ISO8601 format. If provided the client may choose to use the lease to reserve the space while the customer is entering personal details and payment details. In this case the “customer” property is not required with the first POST to /order. If not provided, the lease is available for the purposes of reservation as part of a payment flow only. This means that no POST to /order should be made until the complete customer and payment details have been received and validated, and that the PATCH to /order should be expected within 180 seconds of the initial POST. Should the lease expire, the customer details used to create the lease must also be removed from the provider’s system. If not provided, the use of the lease to facilitate a “shopping basket” is strictly prohibited.

Any other fields supported by the event object specification can be included and may be used to provide additional context to customers by clients of a booking standard implementation.

Use of the “id” to retrieve the latest Event

To allow the retrieval of the latest Event data, the “id” of the Event must be a globally unique URI that resolves to an up-to-date contents of the “data” property for that Event in the RPDE feed.

Note that the properties returned by this URI may be a subset of the data in the “data” property of the RPDE feed if feed augmentation has taken place outside of the source booking system, however all properties provided are considered to override anything in the feed. Where augmentation has occurred, the URI should not provided properties that match those in the feed unless they are expected to have identical output in normal operation. New properties can be defined for this purpose, if they are expected to provide different output.

Use of the “id” to retrieve the latest Offer

To allow the retrieval of the latest Offer data, the “id” of the Offer must be a globally unique URI that resolves to an up-to-date contents of the Offer within the Event in the RPDE feed.

Only “Offers” which specify an “id” are bookable, this “id” can be used within the OrderItem to reference the same bookable object.

Bookable free Offers

To allow free opportunities to be booked through the same mechanism as paid opportunities, the following format must be used for a free bookable Offer. Note the “price” is set to the string “0”.

```
{
  "type": "Offer",
  "id": "http://www.book.com/api/sessions/9209#offer",
  "price": "0"
}
```

Free Events

For *universally free* events (distinct from those that have one or more free Offers as well as paid Offers) that require booking in advance, but are accessible for free to anyone, the following property must also be set:

```
    "isAccessibleForFree": true
```

Events that do not require booking in advance / Just turn up and pay

For events that may or may not be bookable, but are accessible without necessarily requiring booking in advance (i.e. “just turn up and pay”), the following property should also be set.

```
    "publicAccess": true
```

Just turn up for free

For events where no booking is required (though it may be available via an Offer) and the participant can “just turn up for free”, the above two properties can be used in combination:

```
    "isAccessibleForFree": true
    "publicAccess": true
```

Request

GET /sessions/{session ID}

Response with Paid session

Session found
STATUS 200

```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "type": "Event",
  "identifier": 9209,
  "id": "http://www.book.com/api/sessions/9209",
  "organizer": {
    "type": "Organization",
    "identifier": 225,
  },
  "name": "Speedball",
  "beta:formattedDescription": "<p>An action packed, fast paced game
              that incorporates netball, handball and
              football.<\/p>",
  "startDate": "2018-01-27T11:00:00Z",
  "remainingAttendeeCapacity": 1,
  "maximumAttendeeCapacity": 10,
  "beta:orderPostUrl": "http://www.book.com/api/orders",
  "beta:anonymousLeaseDuration": "PT15M",
  "offers": [
    {
      "id": "http://www.book.com/api/sessions/9209#!/offers/159",
      "type":"Offer",
      "identifier": 159,
      "name": "London Unlimited ",
      "description": "This is a 30 day unlimited London game pass.
              This pass renews each month but you can cancel at any
              time.",
      "price": "39.00",
      "priceCurrency":"GBP"
    }
  ]
}
```

Response with free session

Note that for the case where each event only has one offer, the offer ID can be referenced as a simple resource or a hash URI rather than a collection. The following two examples are equally valid, and the hash URI is preferred as it is simpler to implement:

- `"http://www.book.com/api/sessions/9209/offer"`.
- `"http://www.book.com/api/sessions/9209#offer"`.

GET http://www.book.com/api/sessions/9210

Session found
STATUS 200

```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "type": "Event",
  "identifier": 9210,
  "id": "http://www.book.com/api/sessions/9210",
  "organizer": {
    "type": "Organization",
    "identifier": 225,
  },
  "name": "Slowball",
  "beta:formattedDescription": "<p>Something else.<\/p>",
  "startDate": "2018-01-27T11:00:00Z",
  "remainingAttendeeCapacity": 3,
  "maximumAttendeeCapacity": 20,
  "beta:orderPostUrl": "http://www.book.com/api/orders",
  "offers": {
      "type": "Offer",
      "id": "http://www.book.com/api/sessions/9210#offer",
      "price": "0"
  },
  "isAccessibleForFree": true,
  "publicAccess": true
}
```

Session does not exist:

HTTP/1.1 404 Not Found
Content-Type: application/problem+json
Content-Language: en

```
{
  “type”: “session_does_not_exist”,
  “title”: “This session does not exist”
}
```

1b. Get Latest Offer Data

For completeness, the offer endpoint referenced in the offer ID can be implemented as an actual endpoint. Note that this is not functionally required for the booking workflow (the /orders endpoint just uses the offer ID url as an ID, and the Get Latest Event Data includes Offers). The modelling specification allows such IDs to optionally resolve to further machine-readable information about the resource, so it is not required to resolve.

Using hash URIs for offers (e.g. “#!/offers/159”) is an alternative to implementing this endpoint.

Request

GET /sessions/{session ID}/offers/{offer ID}

Response

Session found
STATUS 200

```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "id": "http://www.book.com/api/sessions/9209/offers/159",
  "beta:orderPostUrl": "http://www.book.com/api/orders",
  "type":"Offer",
  "identifier": 159,
  "name": "London Unlimited ",
  "description": "This is a 30 day unlimited London game pass.
            This pass renews each month but you can cancel at any
            time.",
  "price": "39.00",
  "priceCurrency":"GDP"
}
```

2. Lease

Reserve an Event’s space for a finite amount of time.

On POST /orders, an order object is created with an expiry time (which is expected to be equal to 180 seconds unless `beta:anonymousLeaseDuration` was set). While this reservation is in orderStatus “OrderPaymentDue”, the number of spaces in that Event decreases by 1 (i.e. the space is taken up). If the order is not confirmed (i.e. changed to state “OrderDelivered”) by the time of the expiry, the order is permanently deleted and the number of spaces in that Event increases by 1 (i.e. the space is released).

This does not require use of an Account, as a “customer” can be specified directly in the order.

If `beta:anonymousLeaseDuration` was set, the “customer” is provided as part of the PATCH Book call to complete the order. Otherwise, the “customer” is provided at the point of Order creation in the first POST.

If the lease times out before it is booked, the customer details provided with the lease must be discarded by the booking system.

Request

Note the “broker” is the third party, and the “customer” is the end user. Both are optional at this stage, although must be added to the order before it is paid.

Also note that “Free” orders still require confirmation. This extra step allows for email verification to take place between the lease and book steps.

Example 1

POST /orders
```
{
  "type": "Order",
  "broker":{
    "type":"Organisation",
    "name":"Change4Life"
  },
  "orderedItem": {
    "type": "OrderItem",
    "acceptedOffer": "http://www.book.com/api/events/1234#!/offer/2134",
    "orderQuantity": 1,
    "orderedItem": "http://www.book.com/api/events/1234#!/subEvent/123"
  },
  "customer":{
    "type":"Person",
    "givenName":"Joe",
    "familyName":"Smith",
    "email":"joe@smith.com"
  }
}
```

Example 2

POST /orders

```
{
  "type": "Order",
  "broker":{
    "type":"Organisation",
    "name":"Change4Life"
  },
  "orderedItem": {
    "type": "OrderItem",
    "acceptedOffer": "http://www.book.com/api/events/1234#offer",
    "orderQuantity": 1
  },
  "customer":{
    "type":"Person",
    "givenName":"Joe",
    "familyName":"Smith",
    "email":"joe@smith.c wwkkom"
  }
}
```

Response

“paymentDueDate” is an ISO-8601 datetime with a time zone designator, which is the lease expiry date.

Note that a successful lease response expands the orderedItems into their original events including the selected offer.

In the case where the customer and the offer match an existing Order that has not yet expired, return the existing Order and update the expiry date. If an existing offer is returned, the response must be status code 200.

Lease successful (paid session):

HTTP/1.1 201 Created
Content-Type: application/json
Content-Language: en
Location: /api/orders/535

```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "type": "Order",
  "id": "http://www.book.com/api/orders/535"
  "identifier": 535,
  "orderedItem": {
    "type": "OrderItem",
    "acceptedOffer":
"http://www.book.com/api/sessions/1234#!/offer/2134",
    "orderQuantity": 1
    "orderedItem": {
      "type": "Event",
      "identifier": 9209,
      "id": "http://www.book.com/api/sessions/9209",
      "organizer": {
        "type": "Organization",
        "identifier": 225,
      },
      "name": "Speedball",
      "description": "An action packed, fast paced game that
incorporates netball, handball and football.",
      "startDate": "2018-01-27T11:00:00Z",
      "offers": {
        "id": "http://www.book.com/api/sessions/9209#!/offer/2134",
        "type":"Offer",
        "identifier": 159,
        "name": "London Unlimited ",
        "description": "This is a 30 day unlimited London game pass.
This pass renews each month but you can cancel at any time.",
        "price": "39.00",
        "priceCurrency":"GDP"
      }
    }
  },
  "orderStatus": "OrderPaymentDue",
  "paymentDueDate": "2018-02-20T11:00:00Z",
  "orderDate": "2018-02-20T11:00:00Z",
  "partOfInvoice": {
    "type": "Invoice",
    "paymentStatus": "PaymentDue",
    "totalPaymentDue": {
      "type": "MonetaryAmount",
      "value": "10.00",
      "currency": "GBP"
    }
  }
}
```

Lease successful (free session):

HTTP/1.1 201 Created
Content-Type: application/json
Content-Language: en
Location: /api/orders/535

```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "type": "Order",
  "id": "http://www.book.com/api/orders/535"
  "identifier": 535,
  "orderedItem": {
    "type": "OrderItem",
    "acceptedOffer": "http://www.book.com/api/sessions/9209#offer",
    "orderQuantity": 1
    "orderedItem": {
      "type": "Event",
      "identifier": 9209,
      "id": "http://www.book.com/api/sessions/9209",
      "organizer": {
        "type": "Organization",
        "identifier": 225,
      },
      "name": "Speedball",
      "description": "An action packed, fast paced game that
incorporates netball, handball and football.",
      "startDate": "2018-01-27T11:00:00Z",
      "offers": {
        "id": "http://www.book.com/api/sessions/9209#offer",
        "type":"Offer",
        "price": "0"
      }
    }
  },
  "orderStatus": "EmailValidationRequired",
  "paymentDueDate": "2018-02-20T11:00:00Z",
  "orderDate": "2018-02-20T11:00:00Z"
}
```

No available spaces:

```
HTTP/1.1 409 Conflict
Content-Type: application/problem+json
Content-Language: en

{
   “type”: “no_spaces_available”,
   “title”: “There are no spaces available”
}
```

Email address invalid:

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json
Content-Language: en

{
   “type”: “email_address_invalid”,
   “title”: “Email address is invalid”
}
```

2b. Get Order Status

To aid debugging and allow for lease validation, a GET request returns the current Order.

Request

```
GET  /orders/535
```

Response

The response will be identical to that of the Lease or Book call, depending on the status of the Order.

3. Book

Fulfill a lease and confirm payment

A successful call to this endpoint should set the paymentDueDate to null and, where applicable, generate a barcode number (“confirmationNumber”) used to access the session. It will also set the booking account’s expiry to null.

Note that for entirely free bookings, the customer e-mail address must be validated by the broker between the lease and book step. This may be achieved, for example, by sending a confirmation code to the email address.

Request for paid event

PATCH /orders/{order ID}

PATCH /orders/535

```
{
  "type": "Order",
  "partOfInvoice": {
    "type": "Invoice",
    "accountId": 1232312,
    "paymentStatus": "PaymentComplete",
    "paymentMethod": "http://purl.org/goodrelations/v1#Stripe",
    "paymentMethodId": "[StripeToken]",
    "totalPaidByCustomer": {
      "type": "MonetaryAmount",
      "value": "10.00",
      "currency": "GBP"
    },
    "totalPaidToProvider": {
      "type": "MonetaryAmount",
      "value": "8.00",
      "currency": "GBP"
    }
  }
}
```

Response

Booking successful:
STATUS 200

```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "type": "Order",
  "identifier": 535
  "orderedItem": {
    "type": "OrderItem",
    "orderQuantity": 1
    "acceptedOffer":
"http://www.book.com/api/sessions/9209#!/offers/2134",
    "orderedItem": {
      "type": "Event",
      "identifier": 9209,
      "id": "http://www.book.com/api/sessions/9209",
      "organizer": {
        "type": "Organization",
        "identifier": 225,
      },
      "name": "Speedball",
      "description": "An action packed, fast paced game that
incorporates netball, handball and football.",
      "startDate": "2018-01-27T11:00:00Z",
      "offers": {
        "id": "http://www.book.com/api/sessions/9209#!/offers/2134",
        "type":"Offer",
        "identifier": 159,
        "name": "London Unlimited ",
        "description": "This is a 30 day unlimited London game pass.
This pass renews each month but you can cancel at any time.",
        "price": "39.00",
        "priceCurrency":"GDP"
      }
    }
  }
  "orderDate": "2018-02-20T11:00:00Z",
  "orderStatus": "OrderDelivered",
  "partOfInvoice": {
    "type": "Invoice",
    "accountId": 1232312,
    "paymentStatus": "PaymentComplete",
    "paymentMethod": "http://purl.org/goodrelations/v1#Stripe",
    "paymentMethodId": "AppName",
    "totalPaidByCustomer": {
      "type": "MonetaryAmount",
      "value": "10.00",
      "currency": "GBP"
    },
    "totalPaidToProvider": {
      "type": "MonetaryAmount",
      "value": "8.00",
      "currency": "GBP"
    }
  },
  "confirmationNumber":"T112434234"
}
```

Request for free event

PATCH /orders/{order ID}

PATCH /orders/535

```
{
  "type": "Order",
  "orderStatus": "OrderDelivered"
}
```

Response

<u>Booking successful</u>:
STATUS 200

```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "type": "Order",
  "identifier": 535,
  "orderedItem": {
    "type": "OrderItem",
    "acceptedOffer": "http://www.book.com/api/sessions/9209#offer",
    "orderQuantity": 1
    "orderedItem": {
      "type": "Event",
      "id": "http://www.book.com/api/sessions/9209",
      "identifier": 9209,
      "organizer": {
        "type": "Organization",
        "identifier": 225,
      },
      "name": "Speedball",
      "description": "An action packed, fast paced game that
incorporates netball, handball and football.",
      "startDate": "2018-01-27T11:00:00Z",
      "offers": {
        "id": "http://www.book.com/api/sessions/9209#offer",
        "type":"Offer",
        "price": "0"
      }
    }
  }
  "orderDate": "2018-02-20T11:00:00Z",
  "orderStatus": "OrderDelivered",
  "confirmationNumber":"T112434234"
}
```

Lease expired:

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json
Content-Language: en

{
   “type”: “lease_has_expired”,
   “title”: “It is not possible to book this offer as the lease has expired”
}
```

Account does not exist:

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json
Content-Language: en

{
   “type”: “account_does_not_exist”,
   “title”: “It is not possible to book this offer as the account does not exist. Perhaps the account has
expired”
}
```

4. Cancel order

Cancelling an order is a case of updating the order's status:

Request for free event:

PATCH /orders/535
```
{
  "type": "Order",
  "orderStatus": "OrderCancelled"
}
```

Response

Booking successful:
STATUS 200
```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "type": "Order",
  "identifier": 535,
  "orderedItem": {
    "type": "OrderItem",
    "acceptedOffer": "http://www.book.com/api/sessions/9209#offer",
    "orderQuantity": 1
    "orderedItem": {
      "type": "Event",
      "id": "http://www.book.com/api/sessions/9209",
      "identifier": 9209,
      "organizer": {
        "type": "Organization",
        "identifier": 225,
      },
      "name": "Speedball",
      "description": "An action packed, fast paced game that 
incorporates netball, handball and football.",
      "startDate": "2018-01-27T11:00:00Z",
      "offers": {
        "id": "http://www.book.com/api/sessions/9209#offer",
        "type":"Offer",
        "price": "0"
      }
    }
  }
  "orderDate": "2018-02-20T11:00:00Z",
  "orderStatus": "OrderCancelled",
}
```

Lease expired:

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json
Content-Language: en

{
   “type”: “lease_has_expired”,
   “title”: “It is not possible to book this offer as the lease has expired”
}
```

Account does not exist:

```
HTTP/1.1 422 Unprocessable Entity
Content-Type: application/problem+json
Content-Language: en

{
   “type”: “account_does_not_exist”,
   “title”: “It is not possible to book this offer as the account does not exist. Perhaps the account has
expired”
}
```

Other Processes

1. Account Expiry

In the “Concepts” section, it was explained that new accounts that have not successfully completed a booking *must* be deleted after a **30 minute** expiry in order to conform to GDPR rules.

One suggested mechanism for implementing this is a cron job which runs the following SQL (assuming a SQL based implementation):

```
DELETE FROM accounts
WHERE expiry <= CURRENT_TIMESTAMP;
```

2. Lease Expiry

In the “Lease” step in the workflow, it was explained that reservations which are leased are given an expiry time. If not booked by the time the expiry is elapsed, they are to be expired, at which point the reserved space is deallocated.

Similar to account expiry, one suggested mechanism for implementing this is a cron job which runs the following SQL (assuming a SQL based implementation):

```
DELETE FROM reservations
WHERE expiry <= CURRENT_TIMESTAMP;
```

Appendix: Outstanding Issues and Notes

The following notes and issues have been moved the the end of this document for reference. For those implementing the specification in its present state, they are outside of scope.

Issues

Issue 1: Use of persistent "Order" semantics vs use of transient "Reservation" semantics, for a Lease	
Order	**Reservation**
Similar to Stripe, an Order is created as during the checkout process with a unique identifier and a "Reserved" or "Unpaid" status. If the Order is left unpaid, it is automatically removed after a timeout, and once paid it will remain indefinitely. Refunds can be applied, changing the status of the order. 1. Creates one object that persists throughout the lifecycle of the purchase, from reservation, to payment, to refund. This simplifies debugging and audit. 2. Reduces the complexity of the implementing system, as in the simple case only a single database table is required which handles orders throughout their lifecycle. 3. Only a single REST path needs to be implemented at /orders to allow creation, payment and cancellation of orders (as opposed to /reservations) 4. An "Order" to reserve an item could semantically still be considered an order. 5. The "customer" can be added to the Order both at point of reservation, or at point of payment, with further customer details captured after payment is taken, with the same semantics in all cases. 6. For free activities, a reservation is not required and an order can be immediately created. This simplifies the semantics for free activity booking, as a POST to /orders is sufficient. 7. It makes future shopping basket semantics easier, as you can refund the	A Reservation is created during the checkout process, which is converted to a Booking following payment. If the Reservation is left unpaid, it is automatically removed after a timeout. Refunds can be applied to the Booking, changing the status to "Refunded". 1. Reservation has a clearer semantic meaning than an Order in a Reserved state 2. A well defined step between Reservation and Booking serves as a clear trigger for persistent object creation on the provider system (e.g. account creation).

items in the order, and the items themselves persist after purchase. 8. Lack of clear boundary between reservation and booking objects allows the specification to more easily adapt to future requirements such as changing an existing booking (e.g moving the time of a Squash court booking, which is permissible in some systems without generating a new invoice).	

Issue 2: Shopping basket vs. collection of leases	
Collection of leases	**Shopping basket**
The Order can be created at the point of payment, and a collection of leases can be passed to the booking endpoint to facilitate it 1. Advantage: Simplicity of implementation 2. Disadvantage: Lease timeouts may need resetting independently 3. For cases where leases cannot be anonymous, customer details must be provided with each individual lease. An additional error case is introduced where these customers details may not match.	The Order can be created at the point of first lease, and the order accrues leases as the user continues their journey. The order then exists at the point of payment, and can easily be paid. 1. Advantage: The order timeout can be reset with each action per user session 2. Disadvantage: Increase of complexity 3. Use of a central “Order” reduces the need to create an Account in the case where leases cannot be anonymous, and so enables “guest checkout” for this use case.

Issue 3: Anonymous leases	
Allow anonymous leases	**Disallow anonymous leases**
Leases can be acquired without customer data being provided 1. Open to abuse if not correctly implemented by broker e.g. by bots (for implements which do not use authentication) 2. Assurance to customer that later inputted user data will not have been entered in vain	Leases can only be acquired with customer data provided 1. If user data capture is at the last step of the customer journey, the user may be disappointed if they are not able complete the booking

3. Committing to an Event before any user data capture will likely decrease chance for customer drop-off	

Issue 4: bookableItemId vs. Offer URI vs Offer Identifier	
bookableItemId	**Offer URI / Identifier**
v0.2 had the notion of a bookableItemId which would be added to an Offer or an Event to allow them to be bookable. Hence Offers did not need to be used for free events. 1. This makes simple implementations more simple, but adds complexity to clients (as there are more variations of workflow to handle).	V0.3 moved towards the data model requiring Offers for free and paid bookable Events (see discussion at 35:30), and using these as the primary identifiers to trigger the booking workflow. 1. The Offer URI creates a parsing problem, as the /orders endpoint needs to parse the Offer URI to retrieve the relevant Event ID and Offer ID. Another approach is to use an Offer identifier without a mandated structure (schema.org's "identifier" instead of JSON-LD "@id" URI). 1. It might be more straightforward to allow for any form of ID to represent the Offer and not limit it to a URI, however the advantage of a URI form is that it allows the embedding of potentially complex data that can be passed blindly by the client from the Offer into the /orders call, while still prescribing an element of structure.

Booking Multiple Events

Note: Discussion points for "Book multiple events" included in design thinking
The semantics of this specification can be easily extended to book multiple events with the following considerations:

1. Note that the **orderedItem** in the top level **Order** can also support an array of items, which supports the creation of a shopping basket.
2. For orders that consist of only isAccessibleForFree items, the order is instantly confirmed. If an order consists of a mix of isAccessibleForFree and offer items, the offer expiry process will apply and the customer will lose their space on the isAccessibleForFree sessions when the order is cancelled.
3. In the simple case, for single item isAccessibleForFree implementation, only a single item POST for orders must be supported.

Add order items to the basket by making a POST to the order's "orderedItems" with a new acceptedOffer:

POST /orders/535/orderedItems
```
{
  "type": "OrderItem",
  "acceptedOffer": "http://www.book.com/api/sessions/9209/offer/2134",
  "orderQuantity": 1
}
```

To remove order items from the basket, a PATCH may be made with an existing acceptedOffer and an orderQuantity of 0, or a DELETE may be used.

PATCH /orders/535/orderedItems/12
```
{
    "orderQuantity": 0
}
```

DELETE /orders/535/orderedItems/12

GET /orders
[TODO - retrieve either future orders or all orders]

Account Creation

Note: Account creation work in progress

Initial thoughts on account creation are included below, however for the initial implementation this is not required, as any accounts can be created as a part of the lease using the "customer" feature of lease creation.

Get Account

Gets account associated with an email address. If found, it returns the ID of the account, if not a 404 Not Found error is returned and a new account must be created.

Request

GET /accounts?email={emailAddress}

Response

Account exists:
STATUS 200

```
{
  "type":"Person",
  "identifier":"abcd1234",
  "email": "example.person@example.com"
}
```

Account does not exist:
HTTP/1.1 404 Not Found
Content-Type: application/problem+json
Content-Language: en

```
{
   “type”: “account_does_not_exist”,
   “title”: “An account with that email does not exist”
}
```

Create Account

Create new account

Request

POST /accounts

```
{
  “email”: “example.person@example.com”
}
```

Response

Account created successfully:
STATUS 201

```
{
  “type”:”Person”,
  “identifier”:”abcd1234”,
  “email”: “example.person@example.com”,
  “accountExpiry”: “2018-03-02T18:53:00.000Z”
}
```

Account with email already exists:

```
HTTP/1.1 409 Conflict
Content-Type: application/problem+json
Content-Language: en

{
   “type”: “account_with_email_already_exists”,
   “title”: “Account with that email already exists”
}
```

Confirmation Email Suppression

Note: Suppressing Confirmation E-mails

Further research is required on the subject of confirmation email suppression of the booking system (to allow the broker to send custom e-mail messages).

1. This call takes a query parameter “suppressConfirmation”. If set to “true”, the client wants to handle notifying the customer of confirmation via email or other means. In this case, the provider *should* not send out its own notifications of confirmation e.g. confirmation emails
2. PATCH /orders/535?suppressConfirmation={suppressConfirmation}

Cancellation

Note: Cancellation

Cancel

Cancel order that is in leased or booked state

A successful call to this endpoint should set the paymentDueDate and confirmationNumber to null

Request

PATCH /orders/{order ID}/orderItems/{orderItem ID}

```
{
  “orderQuantity”: 0
}
```

Response

<u>Cancel successful</u>:
STATUS 200

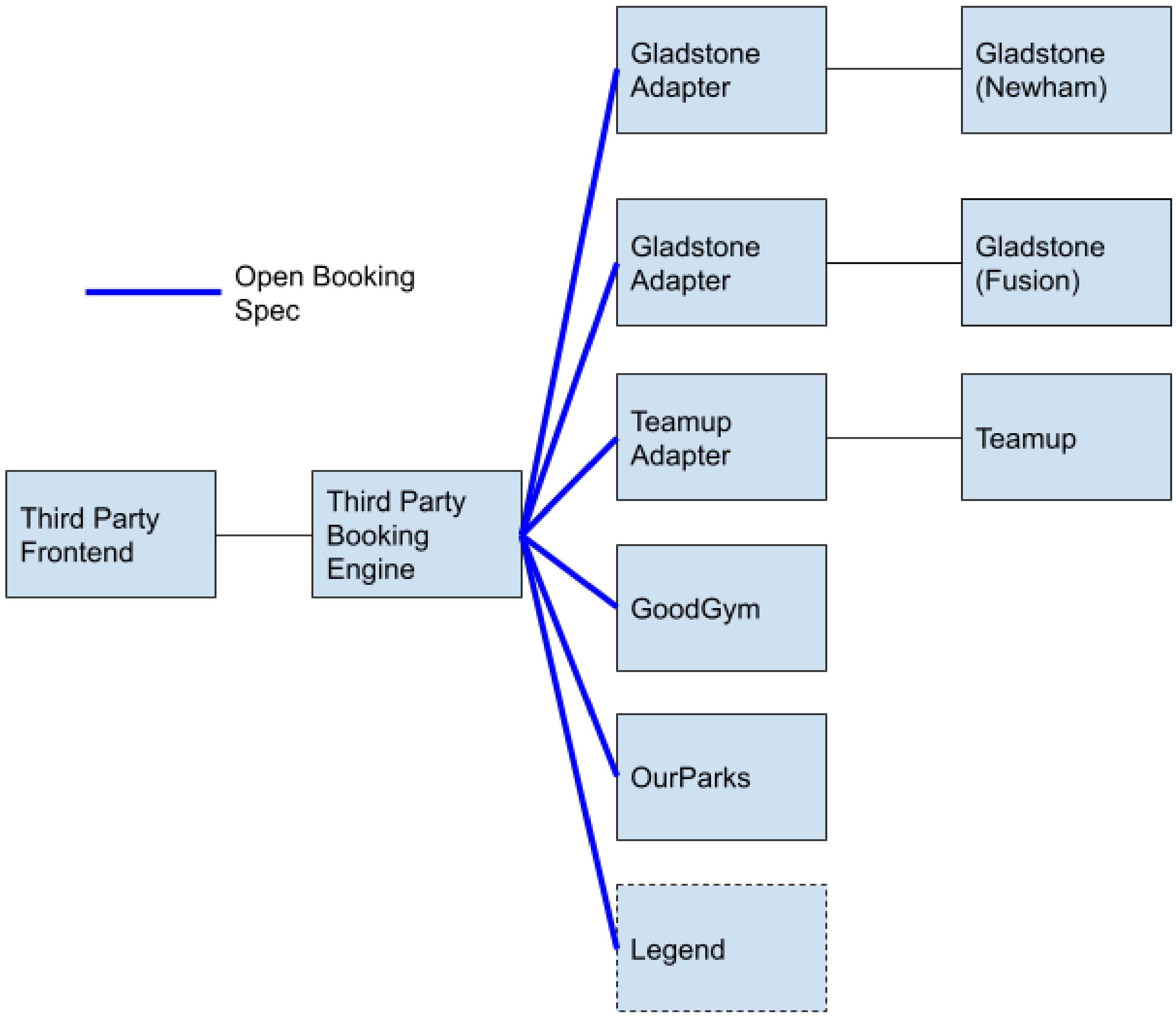
```
{
  "@context": "https://www.openactive.io/ns/oa.jsonld",
  "type": "Order",
  "identifier": 535,
  "orderedItem": {
    "type": "OrderItem",
    "acceptedOffer":
"http://www.book.com/api/sessions/9209/offer/2134",
    "orderQuantity": 1
    "orderedItem": {
      "type": "Event",
      "identifier": 9209,
      "organizer": {
        "type": "Organization",
        "identifier": 225,
      },
      "name": "Speedball",
      "description": "An action packed, fast paced game that
incorporates netball, handball and football.",
      "startDate": "2018-01-27T11:00:00Z",
      "offers": {
        "id": "http://www.book.com/api/events/1234/offer/2134",
        "type":"Offer",
        "identifier": 159,
        "name": "London Unlimited ",
        "description": "This is a 30 day unlimited London game pass.
This pass renews each month but you can cancel at any time.",
        "price": "39.00",
        "priceCurrency":"GDP"
      }
    }
  }, {
    "type": "OrderItem",
    "acceptedOffer":
"http://www.book.com/api/sessions/9209/offer/2134",
    "orderQuantity": 1
    "orderedItem": {
      "type": "Event",
      "id": "http://www.book.com/api/events/1234",
      "identifier": 9209,
      "organizer": {
        "type": "Organization",
        "identifier": 225,
      },
      "name": "Speedball",
      "description": "An action packed, fast paced game that
incorporates netball, handball and football.",
      "startDate": "2018-01-27T11:00:00Z",
      "isAccessibleForFree": true
    }
  }
  "orderStatus": "OrderPaymentDue",
  "paymentDueDate": "2018-02-20T11:00:00Z",
  "orderDate": "2018-02-20T11:00:00Z",
  "partOfInvoice": {
    "type": "Invoice",
    "paymentStatus": "PaymentDue",
    "totalPaymentDue": {
      "type": "MonetaryAmount",
      "value": "-4.00",
      "currency": "GBP"
    }
  }
}
```

TODO: Add POST for processing refund

Appendix: Design Notes

The notes below are a log of the high level decisions taken during the initial specification design. These serve as a guide for discussion points and assumptions made.

- To enable the following architecture:



    - Over time we expect adapters to be redundant and booking systems to implement the specification directly.
- To allow for the following functionality:
    - **Get Space Availability:** To retrieve the explicit number of spaces available at the point where the customer is selecting options and entering payment details
        - Using a subset of the modelling specification to represent the event, availability, and associated offers would allow reuse of existing work.
        - This endpoint should not be required to output the equivalent level of detail to the open opportunity data, to allow for optimisation.
    - **Lease:** Reserves a space(s) without account ID required
        - NB. This is not always implemented, so the workflow should allow for this
        - The lease may fail if there are constraints about the concurrent bookings
            - Error codes should be enumerated, especially for the lease
        - The lease should have a timeout (suggest 15 minutes)

        - Should the lease only be possible after Create or Get, as otherwise it's hard to spot duplicate member bookings?
    - **[Baskets vs Leases]:** In order to assign a payment to multiple bookings, an "order" must be created that encompasses the bookings, and allows them to be assigned to payment.
        - **Option 1:** The order can be created at the point of payment, and a collection of leases can be passed to the booking endpoint to facilitate it
            - Advantage: Simplicity of implementation
            - Disadvantage: Lease timeouts may need resetting independently
        - **Option 2:** The order can be created at the point of lease, and the order accrues leases as the user continues their journey. The order then exists at the point of payment, and can easily be paid.
            - Advantage: The order timeout can be reset with each action per user session
            - Disadvantage: Increase of complexity
    - **Create or Get Third Party Account:** Gets an existing third party account based on e-mail address, or otherwise creates a new account and returns the account identifier
        - Need to define *minimum* required fields to create a member
            - Suggest email address only
        - Need to define custom required fields for each endpoint:
            - Forename, Surname
            - Date of Birth
            - Ethnicity
            - Gender
            - Postcode
        - For existing members, is a new third-party account created, or is the existing member's account utilised
        - Duplicate bookings for the same email address may optionally be not possible
            - Multi-party bookings (booking for more than one person)
                - Can primary booker make multi-party-booking?
        - GDPR consideration: Accounts should not remain in the system if they were created for the sake of a booking which subsequently failed.
            - Alternative routes: timeout to remove accounts that have no bookings; additional endpoint to roll back account creation.
            - Such a timeout should be reset on each call
    - **- Third Party Payment Step -**
    - **Book:** (or "confirm payment") confirms the payment has occurred and registers the amount taken
        - Note the amount paid by the customer, and the amount deposited to the provider should both be captured
        - Triggers the sending of confirmation emails {has side effects}
            - There should be the functionality for these confirmation emails to be suppressed to streamline the user experience
        - The book API call should produce a barcode or other data required for site entry
    - **Cancel:** Allow a booking to be cancelled

        - Should allow cancellation with or without notification
        - Note this should not be used for system-level transaction rollback, as it is intended to be used for user cancellation
    - **[Authenticate existing members]:** OAuth flow to allow authentication of existing members
        - This should be descoped / not in V1
    - **[Waiting lists]**