



Partner API

Quote fares, create bookings and track them in an operator's Almighty Dispatch account.

VERSION 1.2 · 25 OCTOBER 2026

Partner API documentation

Lets a partner system quote a fare, create a booking and follow a booking's status (including the driver and vehicle once allocated) in **one operator's own Almighty Dispatch account**. You can **ask** for the status whenever you like, and/or have the status **pushed** to you by webhook (see "Webhooks"). Both work at the same time: use whichever suits your system. Machine-readable copies: `openapi.yaml` and the Postman collection that come with this document.

Base URL

```
https://www.almightydispatch.com/api/v1
```

If this ever changes, partners are told in advance.

Authentication

Every call needs the operator's API key, sent in a **header** (never in the URL, so it cannot leak into logs):

```
Authorization: Bearer <key>
x-api-key: <key>
```

Either header works.

- The operator creates the key in Dispatch → **Settings** → **Integrations** → **Dispatch API** → **+ Create Key**. The full key is shown **once**; it cannot be recovered later. One key per partner, labelled with the partner's name, so it can be revoked on its own.
- One key belongs to **one operator**. Quotes, bookings and lookups only ever touch that operator's data.
- The operator can **Revoke** a key at any time. A revoked or unknown key gets `401`.
- Keep the key on your server. Never put it in a web page or mobile app.

Conventions

- JSON in, JSON out. Send `Content-Type: application/json` on POST.
- Money is a number in the operator's currency (GBP for UK operators), for example `92`.
- Dates are `YYYY-MM-DD`, times `HH:mm`, in the operator's local time.
- Errors are `{ "error": "message" }` with an HTTP status from the table below.

Status	Meaning
200 / 201	OK / booking created

Status	Meaning
400	Missing or invalid field (the message says which)
401	Missing, invalid or revoked API key
404	Unknown route, vehicle class or booking
409	The same <code>externalRef</code> is still being processed
429	Rate limit reached: wait and retry (<code>Retry-After</code> header)
500	Server error. Quotes and lookups are safe to retry; for a booking, retry with the same <code>externalRef</code>

Limits

120 requests per minute per key, and 60 new bookings per hour per key. Keep status polling to about once a minute per live booking. Keys that misuse the service are revoked.

1. Quote a fare — `POST /fare`

Uses the same pricing engine that charges cards on the operator's own booking widget, so a quote here equals what the operator charges.

Request body

Field	Type	Required	Description
<code>pickup</code>	string	yes	Full address with postcode (max 300 characters). Postcodes drive zones, airport surcharges and fixed prices
<code>dropoff</code>	string	yes	As <code>pickup</code>
<code>distance</code>	number	no	Road distance in the operator's unit (miles for UK). With <code>0</code> or missing you get only the minimum fare, base fee or a fixed price
<code>pickupLat</code> , <code>pickupLng</code> , <code>dropoffLat</code> , <code>dropoffLng</code>	number	no	Decimal degrees. If sent together with <code>distance</code> , the distance is checked against them (see Create a booking)
<code>date</code>	string	no	<code>YYYY-MM-DD</code> . Default today. Used for tariffs, night, weekend and bank-holiday rules
<code>time</code>	string	no	<code>HH:mm</code> . Default <code>12:00</code>
<code>vehicleClassId</code>	string	no	Omit to get a quote for every active vehicle class
<code>promoCode</code>	string	no	Normal promo codes only

```
{
  "pickup": "10 Station Road, Hornchurch RM12 6AA",
  "dropoff": "Heathrow Airport Terminal 5, TW6 2GA",
  "distance": 32.4,
  "date": "2026-11-02",
  "time": "06:30"
}
```

Response 200

Field	Type	Description
<code>night</code>	boolean	Night surcharge applied
<code>fares[]</code>	array	One entry per vehicle class
<code>fares[].vehicleClassId</code>	string	Use this id when booking
<code>fares[].name</code>	string	For example "Saloon"
<code>fares[].seats</code>	number	Seats in the class
<code>fares[].total</code>	number	The fare for the journey
<code>fares[].breakdown[]</code>	array	{ "label": "32 mi x 1x", "amount": 110.4 } lines that add up to total

```
{
  "night": false,
  "fares": [
    { "vehicleClassId": "4f5f1045-...", "name": "Saloon", "seats": 4, "total": 118.4,
      "breakdown": [ { "label": "32 mi x 1x", "amount": 110.4 }, { "label": "Heathrow Drop-off", "amount": 8 } ] }
  ]
}
```

Call this without `vehicleClassId` to **discover the operator's vehicle class ids**: there is no separate list endpoint. If a fixed price matches the two postcodes (for example RM1 → Heathrow) it overrides mileage.

2. Create a booking — `POST /bookings`

Creates a **Pending** booking in the operator's Dispatch board. The fare is **always computed by the server** from the journey; any price you send is ignored. The operator then confirms and dispatches it.

Request body (`customer` and `booking` objects)

Field	Type	Required	Description
<code>customer.name</code>	string	yes	The passenger (max 120 characters)

Field	Type	Required	Description
<code>customer.phone</code>	string	yes	International format, for example <code>+447700900123</code> . The customer record is found or created by phone number
<code>customer.email</code>	string	no	
<code>booking.pickup</code> , <code>booking.dropoff</code>	string	yes	Full address with postcode
<code>booking.pickupLat/Lng</code> , <code>booking.dropoffLat/Lng</code>	number	yes	Decimal degrees
<code>booking.distance</code>	number	yes	Driving distance, greater than 0, in the operator's unit. Must be consistent with the coordinates: not less than 95% of the straight line between them and not more than 3 times it plus 10. Otherwise <code>400</code>
<code>booking.date</code> , <code>booking.time</code>	string	yes	<code>YYYY-MM-DD</code> , <code>HH:mm</code>
<code>booking.vehicleClassId</code>	string	yes	One id, from <code>/fare</code>
<code>booking.externalRef</code>	string	yes	Your own unique reference (max 100 characters). Sending it again never creates a second booking (see Safe retries)
<code>booking.stops</code>	array	no	Strings or <code>{ "address": "..." }</code> . Stops do not change the fare, so quote with the full route distance
<code>booking.passengers</code> , <code>bags</code> , <code>luggage</code>	number	no	
<code>booking.flightNo</code>	string	no	
<code>booking.notes</code>	string	no	Visible to the passenger side. There is no driver-only notes field
<code>booking.isReturn</code>	boolean	no	Creates a second linked booking with the route reversed. The quoted per-leg fare applies to each leg
<code>booking.returnDate</code> , <code>booking.returnTime</code>	string	with <code>isReturn</code>	
<code>booking.paymentMethod</code>	string	no	<code>Cash</code> (default) or <code>Card on Arrival</code> . Online card payment is not taken through this API

```
{
  "customer": { "name": "Jane Smith", "phone": "+447700900123", "email": "jane@example.com" },
  "booking": {
    "pickup": "10 Station Road, Hornchurch RM12 6AA",
    "dropoff": "Heathrow Airport Terminal 5, TW6 2GA",
    "pickupLat": 51.5545, "pickupLng": 0.2209,
    "dropoffLat": 51.4700, "dropoffLng": -0.4543,
    "distance": 32.4,
    "date": "2026-11-02", "time": "06:30",
    "vehicleClassId": "uuid from /fare",
    "externalRef": "PARTNER-12345",
    "passengers": 2, "bags": 2, "flightNo": "BA123"
  }
}
```

Response 201

Field	Type	Description
<code>customerId</code>	string	
<code>bookings[]</code>	array	One entry per booking created (two for a return trip)
<code>bookings[].ref</code>	number	The operator's booking number. Store it
<code>bookings[].returnRef</code>	number or null	The second leg's number for a return trip
<code>bookings[].bookingId</code> , <code>vehicleId</code> , <code>vehicleName</code>	string	
<code>fare</code>	number	The fare, per leg
<code>breakdown[]</code>	array	As in the quote
<code>duplicate</code>	boolean	Only present (<code>true</code> , status <code>200</code>) when this <code>externalRef</code> was already booked

```
{ "customerId": "...", "bookings": [ { "ref": 6141, "returnRef": null, "vehicleName": "Saloon", "bookingId": "...", "vehicleId": "..." } ],
  "fare": 118.4, "breakdown": [ { "label": "32 mi x 1x", "amount": 110.4 } ] }
```

Safe retries. If a call times out, send it again with the same `externalRef` : you get the original booking back, never a duplicate. A `409` means the first attempt is still being processed: retry in a few seconds.

3. Check a booking — GET /bookings

Query with either `?ref=6141` (the operator's number) or `?externalRef=PARTNER-12345` (your own reference). You can only look up bookings created with your own key; any other number returns `404` . The driver's fare is never returned.

Response 200

Field	Type	Description
<code>ref</code>	number	
<code>status</code>	string	See "Booking status" below
<code>date</code> , <code>time</code>	string	<code>time</code> may come with seconds (<code>06:30:00</code>): read the first five characters
<code>pickup</code> , <code>dropoff</code>	string	
<code>vehicleName</code>	string	
<code>fare</code>	number	
<code>paymentMethod</code>	string	
<code>isReturn</code>	boolean	
<code>returnDate</code> , <code>returnTime</code>	string or null	
<code>driver</code>	object or null	Once the job is with a driver (<code>Dispatched</code> onward): <code>{ "firstName": "..." }</code> , plus <code>phone</code> only if the operator switched it on for your key. <code>null</code> before that
<code>vehicle</code>	object or null	Once allocated: <code>{ "make", "model", "colour", "registration" }</code> (some fields can be missing). <code>null</code> before that

```
{ "ref": 6141, "status": "Driver Accepted", "date": "2026-11-02", "time": "06:30:00",  
  "pickup": "...", "dropoff": "...", "vehicleName": "Saloon", "fare": 118.4,  
  "paymentMethod": "Cash", "isReturn": false, "returnDate": null, "returnTime": null,  
  "driver": { "firstName": "Sam" },  
  "vehicle": { "make": "Toyota", "model": "Prius", "colour": "Silver", "registration": "AB12 CDE" } }
```

The driver's fare, bank details and surname are never returned.

4. Cancel or change a booking — `POST /requests`

Ask the operator to cancel or change one of **your own** bookings. **Nothing changes until the operator approves it** in Dispatch. You can poll for the answer (`GET /requests`) and/or receive it by webhook (`request.approved` , `request.rejected`).

Request body

Field	Type	Required	Description
<code>type</code>	string	yes	<code>cancel</code> or <code>amend</code>
<code>ref</code> or <code>externalRef</code>	number / string	one of them	The operator's booking number, or your own reference

Field	Type	Required	Description
<code>reason</code>	string	no	Shown to the operator (max 500 characters)
<code>changes</code>	object	for <code>amend</code>	Any of: <code>date</code> (YYYY-MM-DD), <code>time</code> (HH:mm), <code>pickup</code> , <code>dropoff</code> , <code>passengers</code> , <code>bags</code> , <code>luggage</code> , <code>flightNo</code> , <code>notes</code> , <code>stops</code> (list of addresses)

```
{ "type": "amend", "externalRef": "PARTNER-12345", "reason": "Flight delayed", "changes": { "time": "10:30", "flightNo": "BA124" } }
```

Response 201 : `{ "id": "...", "status": "pending" }`

Rules: - Only bookings created with your key. - Not possible once the passenger is on board, or the booking is completed or cancelled (`409`). An amend is also refused once the driver has arrived. - One pending request per booking and type (`409` if you send a second). - The fare is **not** recalculated for a change. The operator adjusts it if needed. - You cannot change the vehicle class, the customer or the fare through a request. - Up to 30 requests per hour per key.

Check the answer — `GET /requests?id=<id>` (or `?ref=<booking number>` for the latest 20):

```
{ "id": "...", "type": "amend", "status": "approved", "ref": 6141, "externalRef": "PARTNER-12345", "note": null, "createdAt": "2026-11-01T10:00:00Z", "decidedAt": "2026-11-01T10:04:00Z" }
```

`status` is `pending` , `approved` or `rejected` ; `note` is the operator's reason when they rejected it. An approved cancel also sends `booking.cancelled` .

Sandbox (test keys)

Ask the operator for a **test key** (it starts with `tdk_test_`). It works against the same web address, with simulated bookings:

- `POST /fare` gives real quotes.
- `POST /bookings` returns a booking number (from 900000001 up) but **creates nothing real**: no booking on the operator's board, no customer, no messages.
- The booking then moves by itself: `Pending` , `Confirmed` after 20 seconds, `Dispatched` after 1 minute, `Driver Accepted` after 100 seconds, `Arrived` after 140, `In Progress` after 180, `Completed` after 4 minutes. `GET /bookings` shows it with a test driver (Sam) and car (Toyota Prius, SB21 TST), and `"sandbox": true` .
- If the operator set up a webhook address on the test key, the same events are pushed on that timeline (within about a minute of each step).
- `POST /requests` is approved at once, so you can test cancel and change handling.
- Return trips are not simulated (one leg only). A test key never touches live data, so use it freely.

When you are ready, the operator replaces the test key with a normal one. Nothing else in your code changes.

Booking status

A booking normally moves in this order:

Status	Meaning
Pending	Received, waiting for the operator to confirm. Every booking created through this API starts here
Confirmed	The operator accepted it
Dispatched	Offered or sent to a driver
Driver Accepted	A driver accepted and is on the way
Arrived	The driver is at the pickup
In Progress	Passenger on board
Completed	Journey finished
Cancelled	Cancelled by the operator (or by the passenger through the operator)

Pool and Pool-Claimed can also appear while the operator offers the job to several drivers.

Webhooks (push updates)

Instead of (or as well as) asking for the status, the operator can give your key a web address and Almighty Dispatch will **POST a signed message to it** each time one of your bookings changes. The operator sets this up in Dispatch → Settings → Integrations → Dispatch API → **Webhook** on your key: your **https** address, and a **signing secret** (shown once) that they send you privately.

Events

event	When
booking.confirmed	The operator accepted the booking
booking.driver_allocated	A driver was assigned and the job was sent to them. Includes driver and vehicle
booking.driver_accepted	The driver accepted and is on the way
booking.driver_changed	A different driver was assigned before pickup
booking.driver_arrived	The driver is at the pickup
booking.passenger_on_board	Passenger on board, journey started
booking.completed	Journey finished
booking.cancelled	The booking was cancelled

event	When
booking.updated	The pickup date or time changed
request.approved	The operator approved your cancel or change request. Body has request (id , type , status , note) and booking (ref , externalRef)
request.rejected	The operator rejected it; request.note may say why
webhook.test	Sent when the operator presses "Send test"

Message (Content-Type: application/json)

```
{
  "id": "0b6f8d0e-6c1e-4a52-9d46-1f6a3c2f8f11",
  "createdAt": "2026-11-02T05:41:09.000Z",
  "event": "booking.driver_allocated",
  "booking": { "ref": 6141, "externalRef": "PARTNER-12345", "status": "Dispatched",
    "date": "2026-11-02", "time": "06:30", "pickup": "...", "dropoff": "...", "vehicleName": "Saloon" },
  "driver": { "firstName": "Sam" },
  "vehicle": { "make": "Toyota", "model": "Prius", "colour": "Silver", "registration": "AB12 CDE" }
}
```

driver and vehicle are null until a driver is allocated. driver.phone appears only if the operator ticked that option for your key. For a return trip each leg is its own booking with its own ref .

Headers

Header	Value
X-AD-Event	The event name
X-AD-Delivery	Unique id of this message (same as id)
X-AD-Signature	t=<unix seconds>,v1=<hex>

Verify the signature. v1 is the lowercase hex HMAC-SHA256, keyed with your signing secret, of the text <t> .<raw request body> . Compute it over the **raw body exactly as received**, compare in constant time, and reject messages whose t is more than 5 minutes old. Reject anything that does not match: your address is public, the signature is what proves the message is ours.

```
// Node.js
import crypto from 'node:crypto'
function verify(rawBody, header, secret) {
  const t = /t=(\d+)/.exec(header)?.[1], v1 = /v1=([0-9a-f]+)/.exec(header)?.[1]
  if (!t || !v1 || Math.abs(Date.now() / 1000 - Number(t)) > 300) return false
  const mine = crypto.createHmac('sha256', secret).update(`${t}.${rawBody}`).digest('hex')
  return mine.length === v1.length && crypto.timingSafeEqual(Buffer.from(mine), Buffer.from(v1))
}
```

```
# Python
import hmac, hashlib, time, re
def verify(raw_body: bytes, header: str, secret: str) -> bool:
    t = re.search(r"t=(\d+)", header); v1 = re.search(r"v1=([0-9a-f]+)", header)
    if not t or not v1 or abs(time.time() - int(t.group(1))) > 300: return False
    mine = hmac.new(secret.encode(), f"{t.group(1)}.encode() + raw_body, hashlib.sha256).hexdigest()
    return hmac.compare_digest(mine, v1.group(1))
```

Your reply. Answer with any `2xx` status within 8 seconds. Anything else (or no answer) counts as a failure. Redirects are not followed. The response body is ignored.

Retries. A failed message is retried after 1 minute, 5 minutes, 30 minutes, 2 hours, 6 hours and 12 hours (7 attempts in all), then dropped; the operator is alerted. Use `X-AD-Delivery / id` to ignore a message you have already processed: a retry can arrive after you did process the first one. Messages can arrive out of order, so use the `booking.status` inside the message and ignore anything older than what you already have (or ask `GET /bookings`).

Address rules. `https` only, port 443, a public website name (no IP addresses, no private or internal networks).

What this API does not do (yet)

- **No live driver position** (map tracking). You get the driver's first name, car and registration, not GPS.
- **No address search, geocoding or distance calculation.** You supply addresses, coordinates and distance.
- **No list endpoint** for vehicle classes (use `/fare` without `vehicleClassId`).
- **No automatic approval for live keys.** A cancel or change request waits for the operator. (Test keys approve at once.)
- **No change of vehicle class, customer or fare** through a request: contact the operator.

Getting started

1. The operator creates a labelled key (Settings → Integrations → Dispatch API) and sends it to you privately.
2. Call `POST /fare` with no `vehicleClassId` to get the vehicle class ids.
3. Optional: ask the operator to set up your webhook address and send you the signing secret.
4. Build and test against a **test key** (no real bookings are made). Then ask the operator for a normal key and swap it in.
5. Go live. The operator can see **Last Used** on the key to confirm traffic.

Quick test

```
curl -s -X POST "https://www.almightydispatch.com/api/v1/fare" \
-H "x-api-key: YOUR_KEY" -H "Content-Type: application/json" \
-d '{"pickup":"RM12 6AA","dropoff":"Heathrow Airport TW6 2GA","distance":32,"date":"2026-11-02","time":"12:00"}'
```

Changelog

- **1.2** (25 Oct 2026): cancel and change requests (`POST /requests` , `GET /requests` , webhooks `request.approved` / `request.rejected`), and sandbox (test) keys.
- **1.1** (25 Oct 2026): webhooks (push updates), and `driver` / `vehicle` in the booking status.
- **1.0** (23 Oct 2026): first release.