PaymentService QA Testing Guide (no APIM yet)

Overview

APIM is not in place yet, but this does **not** block QA. The Function App is directly reachable — QA tests it by calling the Function App URL directly with a **function key**. APIM is only a future outer gateway; it does not change how the Function authenticates on its own.

```
Now:     QA / Postman --(function key)⟶ Function App          ⟵ testable today
Future:  Browser → CMS --(subscription key)⟶ APIM --(function key)⟶ Function App
```

The two layers are additive. Adding APIM later does not require any PaymentService code change; the function-key layer stays in effect.

The checkout model (read this first)

PaymentService only handles **money (Stripe)** and **tax (Avalara)**. It never writes to CRM — CustomerService does that. Checkout has **two stages**:

Stage	Endpoint(s)	What it does	Money	Avalara
1. Quote (tax estimate)	`POST /quote`	Calculates sales tax + total for the cart so the checkout page can show them while the buyer enters their address. Nothing is saved; no payment.	none	transient estimate (not persisted)
2. Checkout (place order + collect payment)	`POST /checkout`, `POST /checkout/confirm`	Calculates tax on the **full** order (persisted) and collects payment via Stripe. On confirm, verifies the payment and **commits the full-order tax**. How much is charged now is chosen by `paymentPlan`.	see below	full-order invoice, committed on confirm

The `paymentPlan` on `/checkout`

`/checkout` is a **single endpoint with two modes**, selected by the `paymentPlan` field. Both modes are otherwise identical (same full-order tax invoice, same `/checkout/confirm` commit step); only the amount charged now differs:

`paymentPlan`	Charged now	Status	Balance
`"deposit"` (default)	refundable **$200 deposit**	**the live flow today**	collected offline by sales over the phone
`"full"`	the **full amount**	**reserved / not wired in** — future online full-payment / **Affirm** path	none left

> There is no separate `/finalize` endpoint. "Pay in full online" is simply `/checkout` with `paymentPlan: "full"` — it is the same order, just a different amount. Omitting `paymentPlan` defaults to `"deposit"`.

What actually happens at checkout today

1. Buyer builds a cart and enters an address → front-end calls `/quote` repeatedly to display the live tax and total.
2. Buyer clicks pay → front-end calls `/checkout` (default `paymentPlan: "deposit"`: full tax is calculated and saved; a Stripe PaymentIntent for the **$200 refundable deposit** is created) and collects the deposit in the browser with Stripe.js.
3. Front-end calls `/checkout/confirm` → PaymentService verifies the deposit succeeded and **commits the full tax**.
4. Stripe fires a webhook → PaymentService notifies **CustomerService**, which creates the **CRM order** and sends the **order-confirmation email**.
5. The **balance is collected offline** by the sales team over the phone (after a home assessment). This is the intended model for at least the next ~2 years — there is **no online balance collection**.

The `"full"` plan exists only so the future online full-payment / Affirm path can be turned on without adding a new endpoint. QA can smoke-test it, but it is not part of the live flow.

"One button, several API calls" (important for QA)

A frequent question: *the buyer only clicks **Place your order** once — where do multiple API calls come from?* **The number of API calls has nothing to do with the number of button clicks.** The front-end makes calls automatically, both before and inside that single click:

- `/quote` **needs no button.** It is the live "Sales tax / Order total" number on the page. While the buyer types their address, the front-end quietly calls `/quote` in the background to refresh that number. The buyer never clicks anything for it.
- **The single "Place your order" click runs three steps in a row**, all inside one JavaScript click handler (the buyer just sees a spinner, then the confirmation page):
    1. `POST /checkout` — backend creates the tax invoice + a Stripe PaymentIntent for the deposit, returns a `clientSecret`.
    2. **Stripe.js** submits the card number straight to Stripe (`stripe.confirmCardPayment` with that `clientSecret`). **The card number goes only to Stripe — it never touches our backend** (PCI reasons).
    3. `POST /checkout/confirm` — after Stripe reports success, the backend re-verifies with Stripe and commits the full tax.

So: **button click ≠ API call.** One click can chain several backend calls; the card entry step in the middle goes to Stripe, not to us.

How is the balance collected? (common misconception)

There is **no endpoint for the balance.** In the live model:

- The **balance is collected offline** by sales over the phone; sales then updates the order status **in CRM by hand**. No PaymentService endpoint is called for the balance.
- "The balance was already received offline, just record it" is a **CRM status update** (CustomerService / sales), **not** a PaymentService call — that path has no card swipe.

- Paying in full **online** is a different thing: it is `/checkout` with `paymentPlan: "full"` (reserved for the future / Affirm), an **alternative to the deposit flow**, not a step after it. It needs the same two calls (`/checkout` + `/checkout/confirm`) as the deposit flow, for the same async-card-payment reason.

What QA needs

#	Item	Where it comes from
1	**Function App base URL**	The deployed address, e.g. `https://func-paysvc-sw-stg-cnc.azurewebsites.net`
2	**Function (host) key**	DevOps: Azure Portal -> Function App -> **App keys** -> copy a host key. Keep it out of source control; store it only in Postman / a secret store.
3	**Test credentials configured**	The deployment app settings must use Stripe **test mode** keys and Avalara **sandbox** credentials.

Authentication model

Endpoint	Auth level	How to call
`GET /api/health`	Anonymous	No key
`GET /api/health/external`	Function	Function key
`POST /api/quote`	Function	Function key
`POST /api/checkout`	Function	Function key
`POST /api/checkout/confirm`	Function	Function key
`POST /api/webhooks/stripe`	Anonymous (Stripe signature)	Test via Stripe Dashboard / CLI only

How to send the function key

- **Header (preferred):** `x-functions-key: <function_key>`
- **Query string:** `?code=<function_key>`

Postman collection

Import `PaymentService-QA.postman_collection.json` (in this folder). Then set the collection variables:

- `baseUrl` - the Function App base URL
- `functionKey` - the host key from DevOps
- `avalaraCompanyCode` - your Avalara company code
- `orderId` - any unique test ID (used by `/checkout`)
- `paymentIntentId` / `avalaraDocCode` - leave blank; fill from the create-step responses before running the confirm steps

Why does Checkout need two calls (create + confirm)?

`/quote` is a single call because nothing is saved and no money moves — it just returns a number. **Checkout** is split into `/checkout` and `/checkout/confirm` because a real payment happens **in the browser, in between**, and the backend cannot block waiting for it.

The three moments of the payment

Step	Who	What happens	Money state
1. create (`/checkout`)	Backend	Avalara saves the full-order tax as an **uncommitted** invoice and Stripe creates a **PaymentIntent** for the amount charged now (a "ready to collect" ticket), returning a `clientSecret`.	**No money moved yet.** The PaymentIntent starts as `requires_payment_method`.
2. pay (front-end Stripe.js)	Customer	The browser uses the `clientSecret` to show the card form. The customer pays; this may include 3-D Secure (bank SMS), may fail, or may be abandoned.	The payment is captured by Stripe **here** - outside our API.
3. confirm (`/checkout/confirm`)	Backend	Verifies with Stripe that the PaymentIntent is truly `succeeded` **and** the amount matches, then **commits** the Avalara tax invoice so it becomes a real, reportable transaction.	Payment confirmed; tax record finalized.

Why it cannot be one call

The gap between create and pay is **asynchronous and out of the server's control** - the customer might pay in two seconds, take a phone call and pay ten minutes later, or never pay at all. The backend cannot block waiting for a card swipe. So it hands a payment ticket to the front-end (create), and only after the customer has paid does it come back to reconcile (confirm).

The security point (important)

`confirm` is not a rubber stamp. It **independently re-checks the amount and status directly against Stripe** - it does not trust the caller's claim that "payment is done". So even a tampered front-end or a forged "already paid" request cannot drive the tax commit unless Stripe actually received the money. **Money and the tax document are strictly linked**, which prevents fake orders and under-reporting tax.

This applies to both `paymentPlan` modes — deposit and full — since they share the exact same create/confirm pair.

In one line: *`/quote` just estimates tax; `/checkout` opens an invoice and collects payment (deposit by default, or full via `paymentPlan`), and `/checkout/confirm` reconciles it after the money actually lands — the uncontrollable, asynchronous card-payment step in between is why checkout needs two calls.*

Test flow

1. Health (anonymous) - confirm deployment is up

```
GET {baseUrl}/api/health
```

Expect `200`. Fastest check that the app is deployed and running.

2. Health external (function key) - confirm Stripe/Avalara connectivity

```
GET {baseUrl}/api/health/external
x-functions-key: <function_key>
```

Expect `200` with Stripe and Avalara dependency statuses. Confirms the credentials in app settings are valid.

3. Quote (function key) - pure tax estimate

No id, no Stripe, nothing persisted — just tax + total. Safe to call as many times as you like.

```
POST {baseUrl}/api/quote
x-functions-key: <function_key>
Content-Type: application/json

{
  "customerEmail": "qa@dcbel.com",
  "currencyCode": "USD",
  "shipTo": { "line1": "500 W 2nd St", "line2": null, "city": "Austin", "stateCode": "TX", "postal
  "billTo": null,
  "items": [
    { "productCode": "DCBEL-R16", "description": "dcbel r16", "quantity": 1, "unitPrice": 5000.00,
  ],
  "shippingAmount": 100.00
}
```

Expect `200` with `{ "taxAmount": ..., "totalAmount": ... }`. There is **no** `paymentIntentId` or `avalaraDocCode` here — quote does not create either.

4. Checkout (function key) - place order + create the payment intent

> **Use a unique orderId per run.** The Avalara doc code is O- and must be unique per company; reusing an id that was already committed returns 502.

Tax is calculated on the **full** order. `paymentAmount` is the amount charged now, and `paymentPlan` selects the mode: `"deposit"` (default, charge the $200 deposit) or `"full"` (reserved — charge the full amount). If you omit

`paymentPlan`, it defaults to `"deposit"`.

```
POST {baseUrl}/api/checkout
x-functions-key: <function_key>
Content-Type: application/json

{
  "orderId": "O-QA-0001",
  "customerEmail": "qa@dcbel.com",
  "currencyCode": "USD",
  "shipTo": { "line1": "500 W 2nd St", "line2": null, "city": "Austin", "stateCode": "TX", "postal
  "billTo": null,
  "items": [
    { "productCode": "DCBEL-R16", "description": "dcbel r16", "quantity": 1, "unitPrice": 5000.00,
  ],
  "shippingAmount": 100.00,
  "paymentPlan": "deposit",
  "paymentAmount": 200.00
}
```

Capture `paymentIntentId` and `avalaraDocCode` from the response for the confirm step.

To smoke-test the reserved full-payment path, send the same request with `"paymentPlan": "full"` and `"paymentAmount"` set to the full order total (`totalAmount` from a quote). Everything else is identical.

Marking a PaymentIntent as paid (test mode, no browser)

The confirm step requires the PaymentIntent to be **succeeded**. A PaymentIntent from `/checkout` starts in `requires_payment_method`. In Stripe **test mode** you can mark it paid directly via the Stripe API with a test-card token (`pm_card_visa` = 4242 4242 4242 4242) — no browser or Stripe.js needed:

```
POST https://api.stripe.com/v1/payment_intents/{paymentIntentId}/confirm
Authorization: <Basic auth with your Stripe test secret key from Stripe Dashboard / DevOps>
Content-Type: application/x-www-form-urlencoded

payment_method=pm_card_visa&return_url=https://example.com/return
```

Response `status` should be `succeeded`. Then run the confirm step below.

5. Confirm checkout (function key) - verify payment + commit full tax

```
POST {baseUrl}/api/checkout/confirm
x-functions-key: <function_key>
Content-Type: application/json

{
```

```
  "orderId": "O-QA-0001",
  "paymentIntentId": "<from step 5>",
  "avalaraDocCode": "<from step 5>",
  "expectedAmount": 200.00,
  "avalaraCompanyCode": "<your Avalara company code>"
}
```

`expectedAmount` must match the amount charged in step 5: the **deposit** ($200) for `paymentPlan: "deposit"`, or the **full total** for `paymentPlan: "full"`. In the live deposit flow the balance is then collected offline by sales.

6. Stripe webhook - special, do NOT test from Postman

```
POST {baseUrl}/api/webhooks/stripe
```

This endpoint is anonymous but validated by a **Stripe HMAC signature**. A hand-crafted Postman request has no valid signature and will be rejected (or return `503` if the webhook secret is not configured). Test it properly with:

- **Stripe Dashboard:** register the webhook URL, then use **Send test event** for `payment_intent.succeeded` / `payment_intent.payment_failed`, or
- **Stripe CLI:** `stripe listen --forward-to {baseUrl}/api/webhooks/stripe`

Negative tests (prove the key is enforced)

Case	Expected
`POST /api/quote` **without** key	`401 Unauthorized`
`POST /api/quote` with a **wrong** key	`401 Unauthorized`
`POST /api/checkout` with `paymentPlan: "half"`	`400` validation error (must be `deposit` or `full`)
`GET /api/health` without key	`200` (anonymous by design)

Notes

- The route prefix is the default `/api`.
- Amounts are in **major currency units** (dollars, not cents).
- Keep the function key out of screenshots, tickets, and source control.