Internal Flow (What Happens Behind Scenes)

When POST runs:

1. Payload created
2. ApiMethods.request() called
3. Supertest builds HTTP call
4. Headers added
5. Request sent
6. Response received
7. Status validated
8. Schema validated (if provided)
9. bookingId stored

POST API – postCreateBooking()

Your Code:

```
async postCreateBooking(overrides = {}, statuscode) {
```

🔎 Line Explanation:

- `async` → Makes function asynchronous (returns Promise).
- `postCreateBooking` → Method name (business-level action).
- `overrides = {}` → Default empty object if no overrides passed.
- `statuscode` → Expected HTTP status code.
- `{}` → Function body begins.

```
const dynamicPayload = await this.api.buildPayload(
    payloads.postbooking.booking,
    overrides
);
```

🔎 Explanation:

- `const dynamicPayload` → Creates constant variable.
- `await` → Waits for promise to resolve.
- `this.api` → Instance of ApiMethods class.
- `buildPayload()` → Utility function to merge base payload with overrides.
- `payloads.postbooking.booking` → Base JSON structure.
- `overrides` → Values from feature file.

🔥 Why?

This prevents modifying original payload and allows dynamic test data.

```
const response = await this.api.request({
```

- `response` → Stores API response.
- `await` → Wait until HTTP request completes.
- `this.api.request()` → Calls centralized HTTP engine.

```
method: "POST",
```

→ HTTP method type.

```
url: endpoints.url,
```

→ Base URL (https://restful-booker.herokuapp.com/)

```
endpoint: endpoints.getbookingid,
```

→ Relative path (booking)

```
body: dynamicPayload,
```

→ Request body sent to server.

```
expectedStatus: statuscode
```

→ Status validation expected.

```
});
```

→ End of request object.

```
this.world.bookingId = response.body.bookingid;
```

🔎 Explanation:

- `this.world` → Cucumber World object.
- `bookingId` → Custom property.
- `response.body.bookingid` → Extract ID from response.

Why?

So next steps can use this ID.

```
return response.body;
```

→ Return only response body (not full response object).

---

2 GET API – getBookingDetails()

```
async getBookingDetails(enterBookingID = null, statuscode = 200) {
```

- `enterBookingID = null` → Optional parameter.
- `statuscode = 200` → Default expected status.

```
const bookingId = enterBookingID || this.world?.bookingId;
```

Explanation:

- `||` → Logical OR.
- If external ID passed → use that.
- Otherwise → use stored ID.
- `?.` → Optional chaining (prevents crash).

```
if (!bookingId) {
    throw new Error("Booking ID not available");
}
```

→ Defensive programming.
Prevents invalid API call.

```
const response = await this.api.request({
```

→ Call centralized engine.

```
method: "GET",
```

→ HTTP GET.

```
endpoint: endpoints.getbookingdetils + bookingId,
```

→ Example:

```
booking/123
```

Dynamic endpoint creation.

```
expectedStatus: statuscode
```

→ Validate HTTP response.

```
return response.body;
```

→ Return JSON object.

🟡 3️⃣ PUT API – updateBookingDetailsByID()

```
async updateBookingDetailsByID(overrides = {}, statuscode = 200) {
```

→ Update method with default parameters.

```
const bookingId = this.world?.bookingId;
```

→ Get stored ID from scenario memory.

```
if (!bookingId) {
    throw new Error("Booking ID not available for update");
}
```

→ Safety validation.

```
const token = await this.api.tokengenerator();
```

🔎 Explanation:

- Calls authentication endpoint.
- Returns token string.
- Required for PUT request.

```
const dynamicPayload = await this.api.buildPayload(
    payloads.putbooking.updatebooking,
    overrides
);
```

→ Build updated JSON payload.

```
const response = await this.api.request({
```

→ Call engine.

```
method: "PUT",
```

→ HTTP PUT method.

```
token: token,
```

→ Inject authentication token.
Internally sets:

```
req.set("Cookie", `token=${token}`);
```

```
return response.body;
```

→ Return updated booking.

🔴 4️⃣ DELETE API – deleteBookingDetailsByID()

```
async deleteBookingDetailsByID(statuscode = 201) {
```

→ Default expected 201.

```
const bookingId = this.world?.bookingId;
```

→ Fetch stored ID.

```
if (!bookingId) {
    throw new Error("Booking ID not available for delete");
}
```

→ Defensive check.

```
const token = await this.api.tokengenerator();
```

→ Generate authentication token.

```
const response = await this.api.request({
```

→ Send DELETE request.

```
method: "DELETE",
```

→ HTTP DELETE method.

```
endpoint: endpoints.getbookingdetils + bookingId,
```

→ Dynamic URL:

```
booking/123
```

```
this.world.bookingId = null;
```

→ Clear scenario state.

Why?
Because resource is deleted.

```
return response.body;
```

→ Return respons