Postman Fundamentals – 1-Day Practical Guide (QA Reality)

Scope: Just enough Postman to be effective in real projects
Focus: CRUD APIs, Collections, Environments, Variables
Rule: ❌ No hardcoding. Ever.

---

0. First, a hard truth (read this)

If you:

- Hardcode URLs
- Hardcode tokens
- Copy-paste request bodies per API

👉 You are **not using Postman properly**. You're just sending HTTP calls.

This guide fixes that in **one day**.

---

1. What you must learn today (no extras)

You only need **3 Postman concepts**:

1. **Collections** – group APIs logically
2. **Environments** – control where APIs run
3. **Variables** – remove hardcoding

Everything else is noise for now.

---

2. Simple CRUD API we'll use

Assume a basic REST API:

```
POST   /users
GET    /users/{id}
PUT    /users/{id}
DELETE /users/{id}
```

Base URL changes per environment.

---

3. Collections (how seniors structure it)

❌ Bad structure

```
Random APIs
Login API
Some test API
```

✅ Correct structure

```
Postman Basics – CRUD
 ├── Create User (POST)
 ├── Get User (GET)
 ├── Update User (PUT)
 └── Delete User (DELETE)
```

A **collection = one feature / one service**.

---

4. Environments (this is mandatory)

Why environments exist

Because this changes:

- Base URL

- Tokens
- Test data IDs

Example environments

Variable	Dev	QA
baseUrl	https://dev.api.com	https://qa.api.com
token	dev-token	qa-token
userId	(empty)	(empty)

Create these in **Postman** → **Environments**.

---

5. Variables (this removes hardcoding)

Variable syntax

{{variableName}}

Base URL usage

❌ Hardcoded

https://qa.api.com/users

✅ Correct

{{baseUrl}}/users

---

6. CRUD requests – how they work together

6.1 Create User (POST)

URL

```
{{baseUrl}}/users
```

Body (raw JSON)

```
{
  "name": "Test User",
  "email": "testuser@mail.com"
}
```

Tests tab (VERY important)

```
var response = pm.response.json();
pm.environment.set("userId", response.id);
pm.test("User created", function () {
    pm.response.to.have.status(201);
});
```

➡️ This stores userId dynamically.

6.2 Get User (GET)

URL

```
{{baseUrl}}/users/{{userId}}
```

Test

```
pm.test("User fetched", function () {
    pm.response.to.have.status(200);
});
```

6.3 Update User (PUT)

URL

```
{{baseUrl}}/users/{{userId}}
```

Body

```
{
  "name": "Updated User"
}
```

6.4 Delete User (DELETE)

URL

```
{{baseUrl}}/users/{{userId}}
```

Test

```
pm.test("User deleted", function () {
   pm.response.to.have.status(200);
});
```

---

7. How to run it properly

1. Select environment (Dev / QA)
2. Run collection in order:
    - Create → Get → Update → Delete
3. No manual ID copy-paste

If you’re copying IDs manually → **you failed the basics**.

---

8. Sample collection file (for understanding)

File name:

/postman-basics/collection.json

```
{
 "info": {
  "name": "Postman Basics – CRUD",
  "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
 },
 "item": [
  {
   "name": "Create User",
   "request": {
    "method": "POST",
    "url": "{{baseUrl}}/users",
    "body": {
     "mode": "raw",
     "raw": "{\"name\": \"Test User\", \"email\": \"test@mail.com\"}"
    }
   }
  },
  {
   "name": "Get User",
   "request": {
    "method": "GET",
    "url": "{{baseUrl}}/users/{{userId}}"
   }
  }
 ]
}
```

You don't need to memorize this — just understand the structure.

---

9. What reviewers look for (important)

✅ Uses environment variables
✅ IDs passed dynamically
✅ Collection can run end-to-end

❌ Hardcoded URLs
❌ Manual copy-paste
❌ One request = one collection

---

10. Final takeaway

> **Postman is a testing tool, not a curl replacement.**

If your collection can't run end-to-end without manual work — it's not production-ready.

If you want next:

- Auth token handling (login → token → reuse)
- Negative CRUD test cases
- Postman → Newman → CI flow

Say what you want.