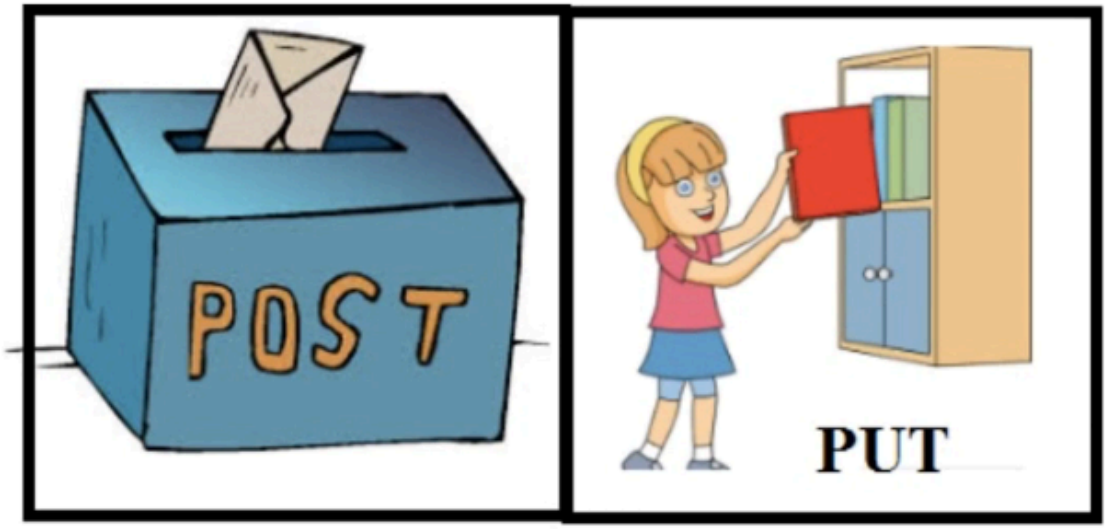
Analogy:

- PUT i.e. take and **put** where it was.
- POST as send mail in **post** office.



Social Media/Network Analogy:

- **Post** on social media: when we post message, it creates new post.
- **Put**(i.e. edit) for the message we already Posted.

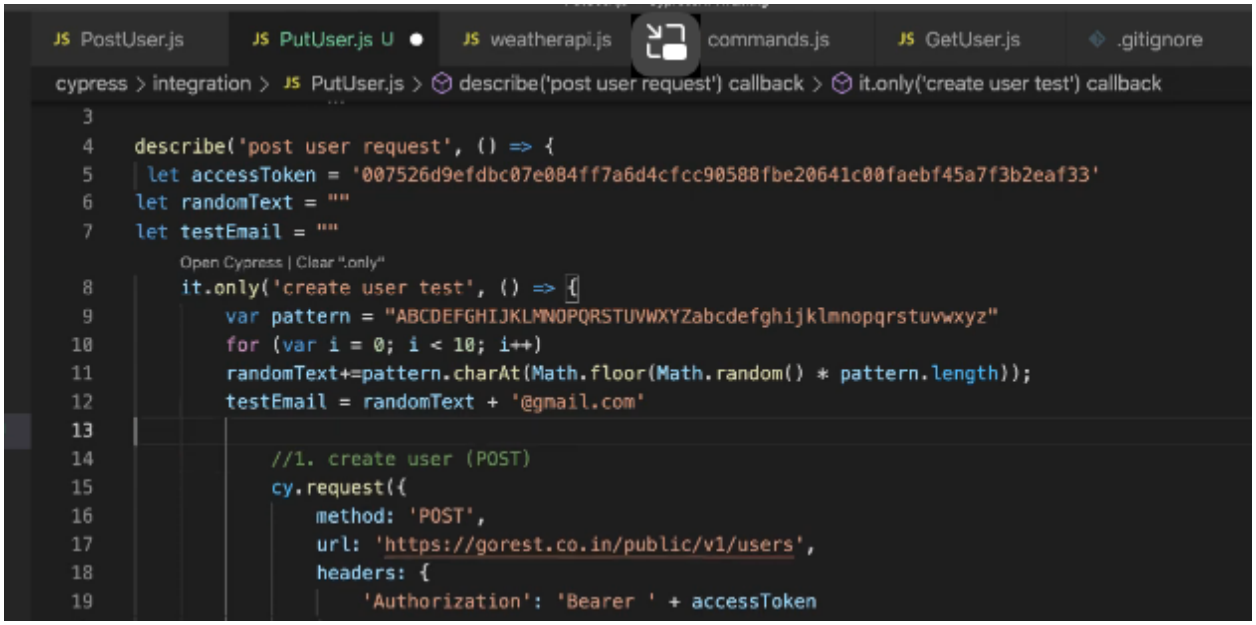
HTTP Methods and Their Meaning

Method	Meaning
GET	Read data
POST	Insert data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

/user/		
GET	/	All users in the same organisation as the requestor
GET	/{ExistingEmailId}/	Returns single user with the same Id
POST	/{ExistingEmailId}/	Updates a user with the emailId, returns the updated User
POST	/{newEmailId}/	Returns an Error
PUT	/{newEmailId}/	Creates a new user and returns the user
PUT	/{ExistingEmailId}/	Overwrites a user with the emailId, returns the updated User
DELETE	/{ExistingEmailId}/	Deletes the user

PUT	POST
Replacing existing resource or Creating if resource is not exist *http://www.example.com/customer/{id}* *http://www.example.com/customer/123/orders/456* Identifier is chosen by the client	Creating new resources(and subordinate resources) *http://www.example.com/ customer/* *http://www.example.com/customer/123/orders* Identifier is returned by server
Idempotent i.e. if you PUT a resource twice, it has no effect. Ex: Do it as many times as you want, the result will be same. x=1;	POST is neither safe nor idempotent. It is therefore recommended for non-idempotent resource requests. Ex: x++;
Works as specific	Works as abstractive
If you create or update a resource using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call.	Making two identical POST requests will most-likely result in two resources containing the same information.

Create user then update the same user.



```
            'Authorization': 'Bearer ' + accessToken
        },
        body: {
            "name":"Test Automation Cypress",
            "gender":"male",
            "email": "naveencypress@gmail.com",
            "status":"active"
          }

    }).then((res)=>{
        cy.log(JSON.stringify(res))
        expect(res.status).to.eq(201)
        expect(res.body.data).has.property('email', 'naveencypress@gmail.com')
        expect(res.body.data).has.property('name','Test Automation Cypress')
        expect(res.body.data).has.property('status','active')
        expect(res.body.data).has.property('gender','male')
```

```
    }).then((res)=>{
        cy.log(JSON.stringify(res))
        expect(res.status).to.eq(201)
        expect(res.body.data).has.property('email', 'naveencypress@gmail.com')
        expect(res.body.data).has.property('name','Test Automation Cypress')
        expect(res.body.data).has.property('status','active')
        expect(res.body.data).has.property('gender','male')
    }).then((res) =>{
            const userId = res.body.data.id
            cy.log("user id is: " + userId)
            //2. update user (PUT)
            cy.request({
                method: 'PUT',
                url: 'https://gorest.co.in/public/v1/users/'+userId,
                headers: {
                    'Authorization': 'Bearer ' + accessToken
```

```
                headers: {
                    'Authorization': 'Bearer ' + accessToken
                },
                body: {
                    "name":"Test Automation Cypress Updated",
                    "gender":"male",
                    "email": "naveencypressupdated@gmail.com",
                    "status":"inactive"
                  }
            }).then((res)=>{
                expect(res.status).to.eq(200)
                expect(res.body.data).has.property('email', 'naveencypressupdated@gmail.com')
                expect(res.body.data).has.property('name','Test Automation Cypress Updated')
                expect(res.body.data).has.property('status','inactive')
                expect(res.body.data).has.property('gender','male')

})
```