General Rules:

1. Usually we use camelCase (locators, property, methods) or PascalCase (For Class)
2. Naming should be meaningful and readable

General Exception: In case of abbreviations like CSR, FPL, OTP etc, we will deviate to make sense of naming

Fixture files

1. All the fixture files are to be segregated under the 'fixtures' folder.
   Naming convention: Camel case
   camelCase.json, camelCase-camelCase.json (in case of long fixture name(s))

   Eg1: goldHomePage-banners.json => This is a fixture used to mock a banners that gets displayed in GoldHomePage

   Eg2: goldPurchasePage-paymentOptions.json => This is a fixture used to mock a paymentOptions page in GoldPurchasePage

2. All related fixture files have to be segregated in sub-folders(and if required nested sub folders).
   Eg: all the gold related fixtures, will be under 'fixtures > gold' , bank related fixtures will be under 'fixtures > bank'

3. Put the meta-data in fixture file, that will detail about url, method, queryparameters etc

```
"meta-data": {
      "url": "http://someurl.com",
      "method": "GET",
      "queryparameters": "type=thumbnail"
  },
```

4. Every now and then compare the latest actual BE response and verify with the existing mock fixture. Only the data/time, personal details, values should be different.Ideally we should have similar model structure. Keep the mockfixture updated frequently
5. Mock the BE responses at all places, however try not to mock the last page where we do the actual test case validation step (if possible and if we are getting some response from KB server), however as our application is complex and we mock the user states, more often than not, we will be mocking all the BE response and validating the FE application behavior. Most third party interactions, SMS etc need to be mocked in our FE Test Automation.

Page Objects

1. Each page of KB (or part of a page/reusable page) of application should be modeled as an independent page object.
2. Page objects are classes that are to be modeled under ‘pages > feature-sub-folder’
   Eg: GoldBuyPage.js is under ‘pages > gold’
3. Naming convention for the file name as well as class name is PascalCase
4. Each page object need to have `/// <reference types = "cypress" />` at the top followed by single line space
5. Spacing has to be strictly followed as per the internal framework rules & guidelines
6. Coding/Naming/Spacing styles
    a. Each class name should have ‘Page’ at the end
    b. Each page object should list all elements locators from top to bottom, left to right in correct chronological order in which they appear in the screen
    c. In case a new element locator is added in the page, then it has to be inserted in appropriate place
    d. Each element locator should have corresponding methods that can be performed on it, eg: inpRupees => we can click, clear and type. So for element locator inpRupees, we need to design three reusable methods that can perform an action on it
    e. If there are any synchronization issues with an element, use timeout in locator strategy, rather than hard coded waits in methods or test calls(from spec file)
    f. Do not mix verification and action methods. I.e click(), type() should NOT have validate/assert eg: it should not call should(‘have.text’) kind of assertions, while verification/assert methods should not perform action methods
    g. Element locator naming should follow camelCase
    h. Method naming should follow camelCase
    i. Do not look to build “Page Object” from Test Case point of view, rather build page object and then design the test case using the designed methods
        i. First identify the feature set/module on which you will work
        ii. Identify all the pages (most of the pages are already named)
        iii. If page does not exists - Consult/Finalize/Publish to FE Automation Team on the new file (i.e Page Object)
        iv. Put it in Tracker => FE_PageObject_Fixture.xls
        v. For the respective page, write all the locators
        vi. Write all methods (action & verification/assert methods fro each locator)
        vii. Once you’re done, then start writing the spec (test scenario)

        Note: Initially till the pages with the locators & methods are being developed, number of test cases/day productivity will be less,

j. Element Locators : Conventions

Type	Convention	Example	Description
Text Field	inp	inpRupees	
Label	lbl	lblGrams	
Button	btn	btnContinue	
Icon	icon	iconAccounts	
Heading	pageTitle		Each page to have this
Link	lnk	lnkRefresh	
Text	txt	txtBuyAmount	For smaller label validation other than Heading/Main labels
Value	val	valGoldBalance	Dynamic value/server/mocked
CheckBox	chkBox	chkBoxConsent	
RadioButton	rb	rbGender	
Info	info	infoDisplayIFSC	Used to validate info message
Message	msg	msgGoldSell	Used to check error message
DropDown	dd	ddLoanPurpose	
DropDownListItem	ddl	ddlEducationLoan	
Chevron Arrow	chevron<Direction>	chevronLeft or chevronRight	
data slide dots	slideDot	slideDotBanner	

k. methodName⇔elementLocaxtor naming style and use of intellisense

So if you read the method, it is clickHomeIcon,

i.e click→ Home → Icon

While the implementation inside will call iconHome().click()

i.e icon→ Home → click

Most of the functions should be written in this fashion to achieve consistency and to make the spec files self-documenting, readable and maintainable

As method names are test facing (spec file), we do avoid writing shortforms like lbl, inp, lnk, txt, val etc

Type	Element	Sample Method Name
Text Field	inpRupees	enterRupees(rupees)
Label	lblGrams	verifyGramsLabel(label)
Button	btnContinue	clickContinue**Btn**()
Icon	iconAccounts	clickAccountsIcon()
Heading	pageTitle	verifyPageTitle(title)
Link	lnkRefresh	clickRefreshLink()
Text	txtBuyAmount	verifyBuyAmountText(text)
Value	valGoldBalance	verifyGoldBalanceValue(value)
CheckBox	chkBoxConsent	uncheckConsent**ChkBox**()

Type	Element	Sample Method Name
Text Field	inpRupees	enterRupees(rupees)
Label	lblGrams	verifyGramsLabel(label)
Button	btnContinue	clickContinue**Btn**()
RadioButton	rbGender	selectGender(gender)
Info	infoIFSCDisplay	verifyIFSCDisplayInfo(info)
Message	msgGoldSell	verifyGoldSellMessage(message)

Note:

1. For Input Field/Text Input Box, we don't use Input at the end
   Eg: enterRupees(rupees), we don't use input as 'enter' is sufficient

2. clickContinue**Btn**() & uncheckConsent**ChkBox**() , are the only two method name where we use short form
3. Enter and Verify/Assert method needs arguments to be passed from spec files

Tests

All the test are segregated under, system and sanity folders

1. Naming Convention: camelCase.spec.js eg addBankAccount.spec.js
2. /// <reference types = "cypress" /> should be the first line
3. Import the pages in the order of consumption/usage
4. Use consistent coding style as per agreed standards
5. Describe the suite properly
6. Detail the it block as per test case id
7. Have all the intercepts in place and have it in the order of consumption/usage in the script
8. Build the script
9. Read the script and ensure that the page & method calls are self documenting and self explanatory
10. Provide verification/assertion points
11. Test the case - Execute for Pass & Fail (intentional) case
12. Document the Pass & Fail case
13. Check-in the code back to remote repo

14. Raise PR get it review and merged
15. Create new branch
16. Ensure that old local and remote branch is deleted after PR is merged [after 1 week]
17. Take the latest code before you start the next test case

Review Guidelines:

1. All the elements which are captured in the page should be reviewed to find the duplicate methods.
2. SonarLint should be used to review the page and its methods.
3. There should not be any unused imports.
4. There should not be any commented lines until and unless those are required.
5. Duplication of the test cases is not allowed.
6. Once the test cases are scripted, they should undergo a thorough review by sonarLint.
7. PR's(pull requests) to be raised once the test cases are reviewed.