Prerequisite

Test automation is written in Java and based on following technologies and frameworks. Please get familiar with them:

- Maven
- JUnit
- Selenium
- Selenide

Local test execution is usually using a local installed Chrome browser. In CI/CD Pipelines we are using SauceLabs multi-browser support. The tests are running in CI/CD side inside docker container.
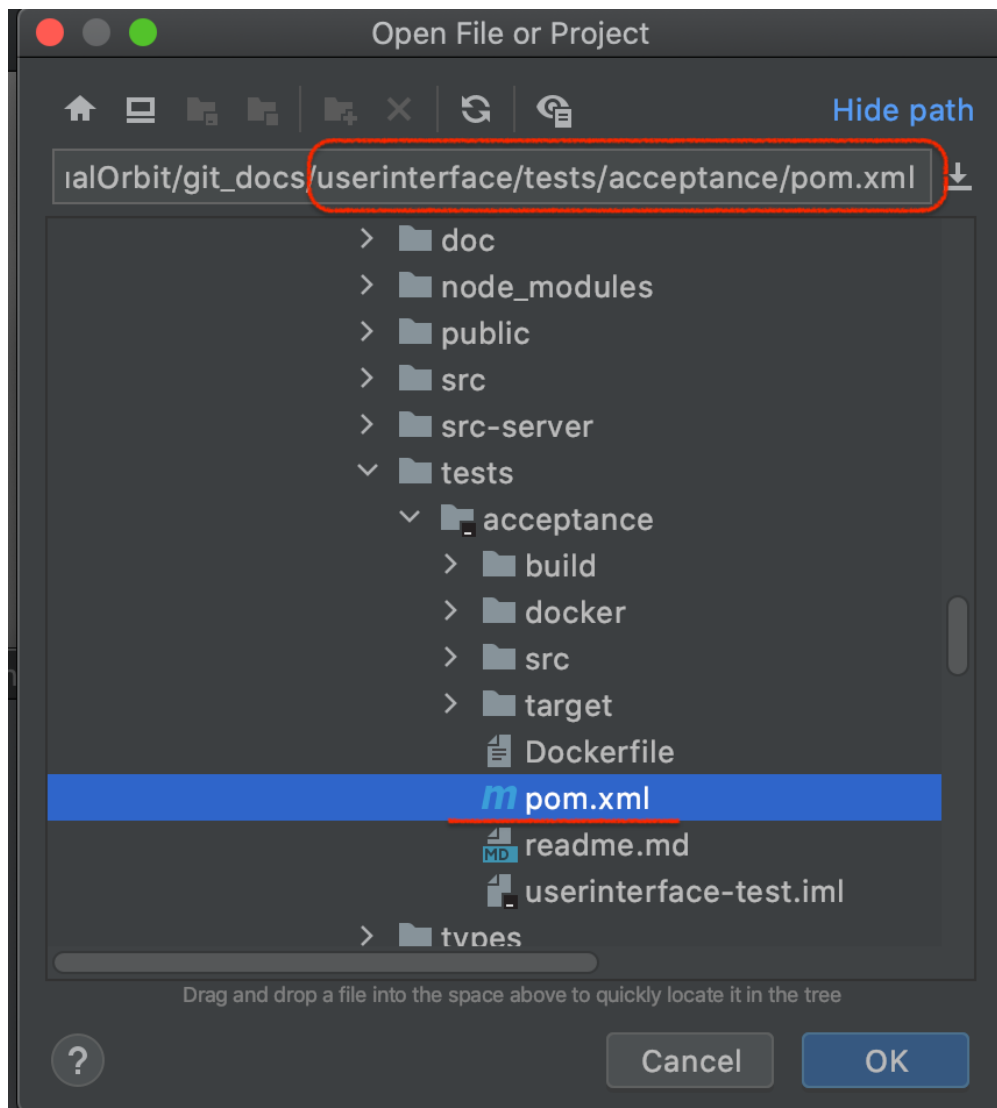
For best practices please check the Selenide project

Setup IDE

- Download and install Intellij IDEA Community Edition (recommended). You can as well use Eclipse or NetBeans or something else, but it is on your own.

- Import/open test project from userinterface as Maven project:
  Select 'Open or Import':
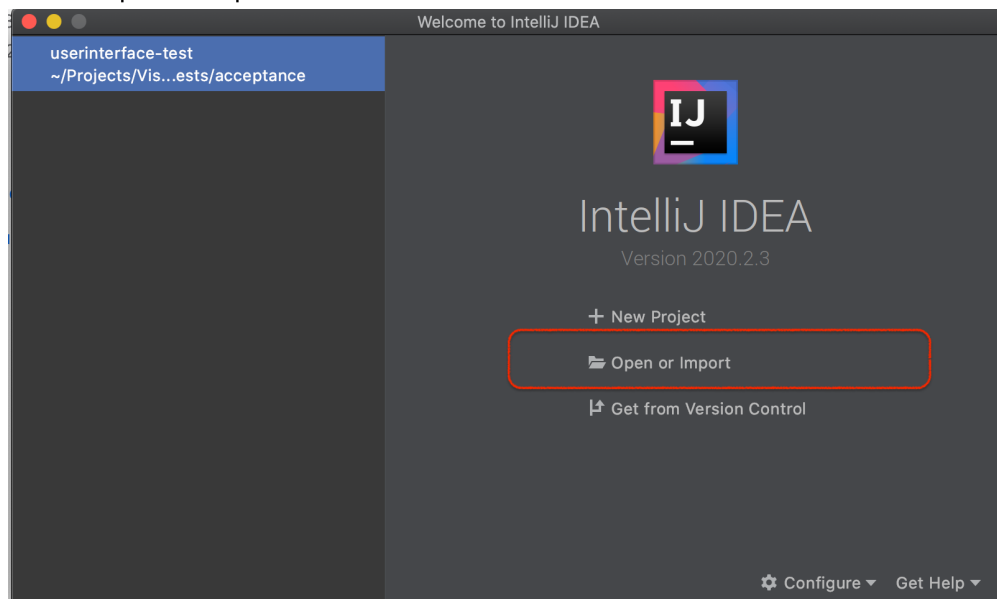
  Select pom.xml file inside tests/acceptance folder:

Open as Project:
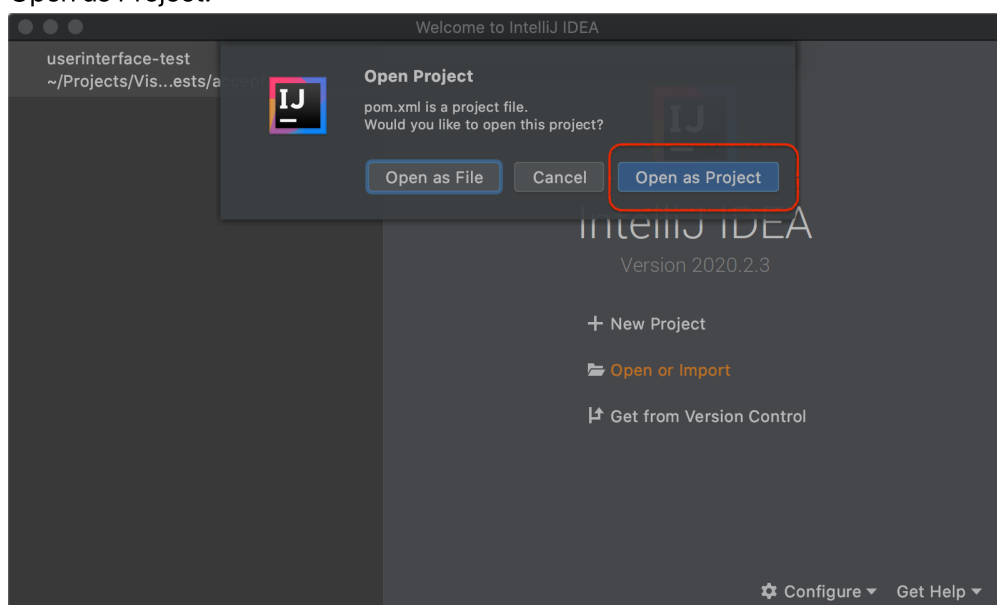
- Setup few environment variables in Run/Debug Configurations via templates. This will add defined environment variables to all tests later.
  Create new JUnit Run/Debug Template:

Fill environment variables. Instead of test user you can use your personal account. Please do not share your password and do not commit it anywhere!
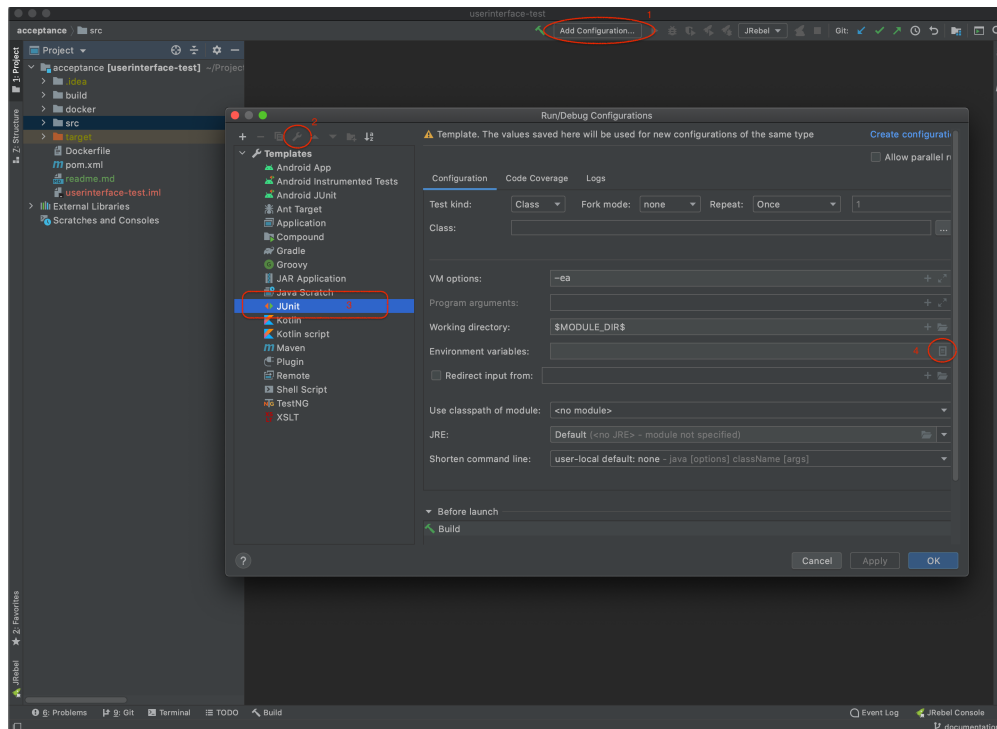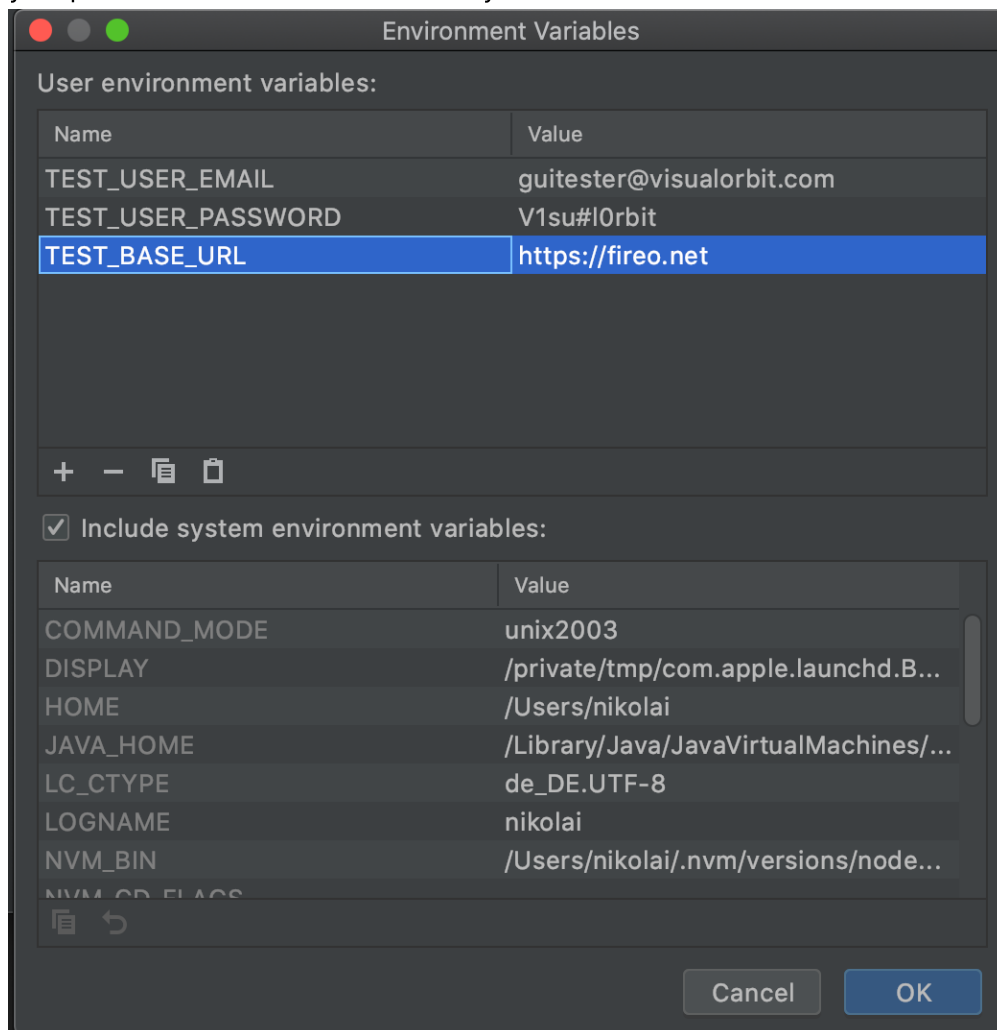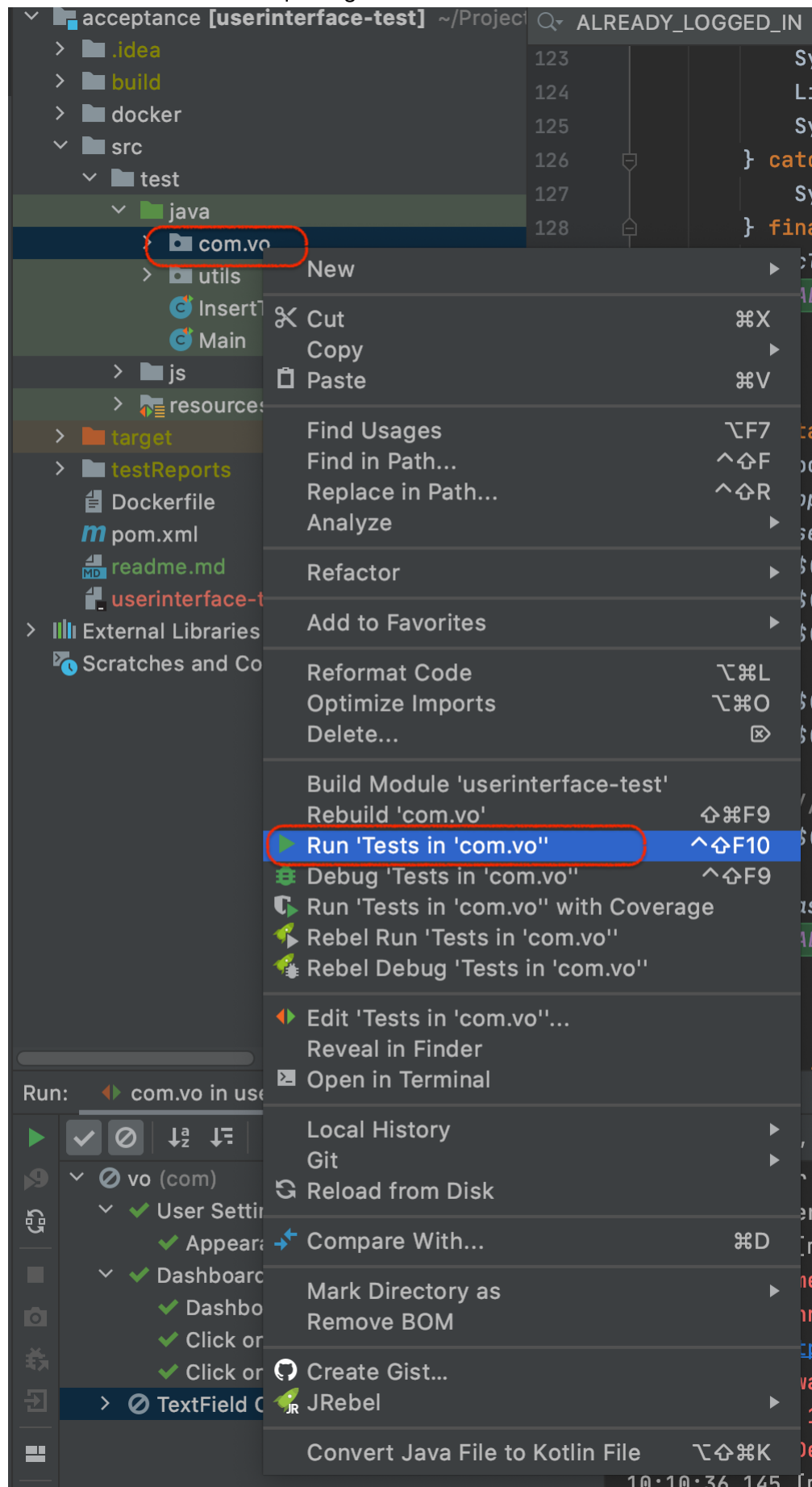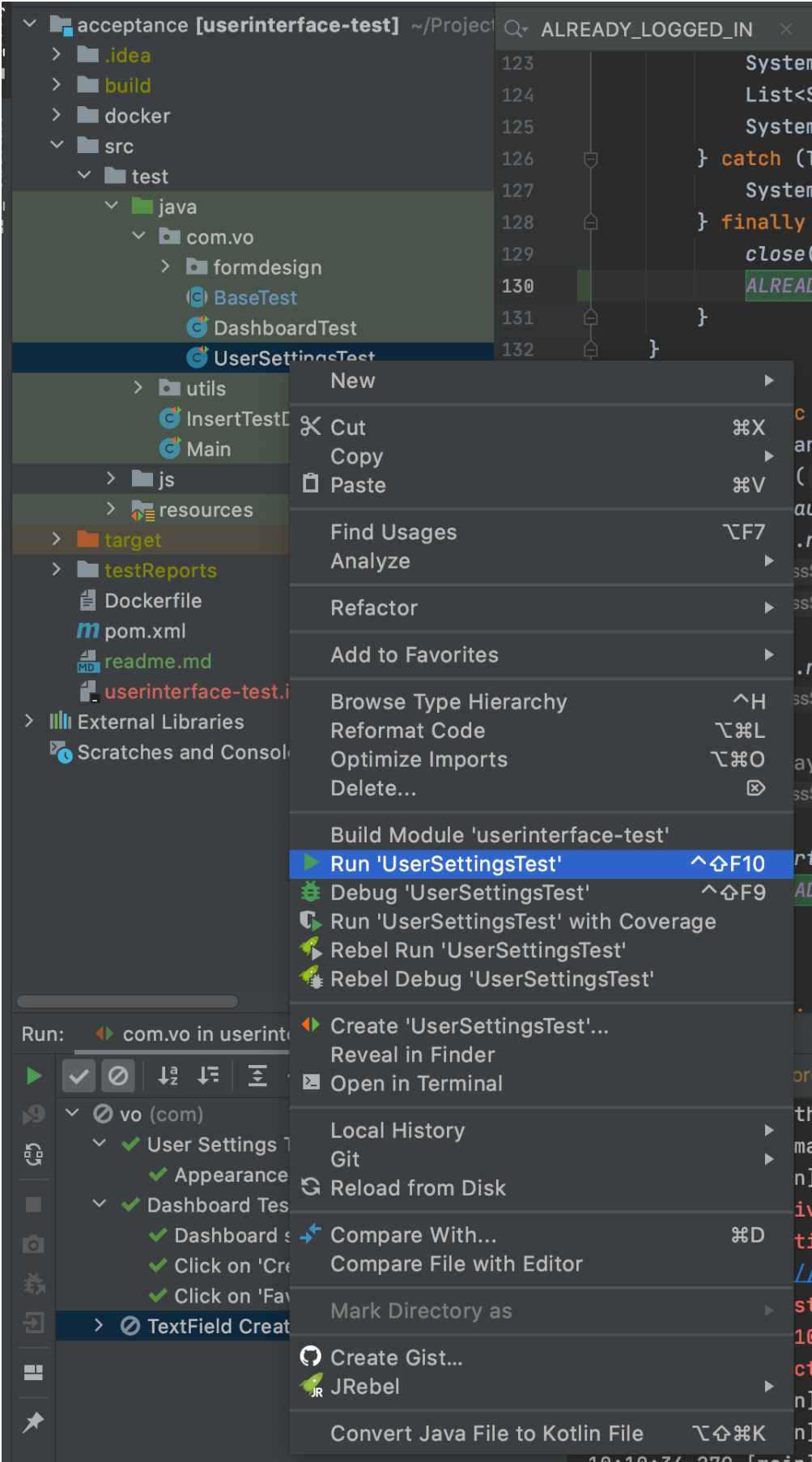

Running Tests in IDE

- Run all tests inside vo.com package:

- Run specific test by selecting the test class:

```
acceptance [userinterface-test] ~/Project          ALREADY_LOGGED_IN
  > .idea                                    123              System
  > build                                    124              List<S
  > docker                                   125              System
  v src                                      126            } catch (
    v test                                   127              System
      v java                                 128            } finally
        v com.vo                             129              close(
          > formdesign                       130              ALREA
            BaseTest                         131            }
            DashboardTest                    132          }
            UserSettingsTest
          > utils                New                            ▶
            InsertTestL    Cut                          ⌘X
            Main           Copy                             ▶
      > js                 Paste                         ⌘V
      > resources
  > target                 Find Usages                   ⌥F7
  > testReports            Analyze                          ▶
    Dockerfile
    pom.xml                Refactor                         ▶
    readme.md
    userinterface-test.i   Add to Favorites                 ▶
> External Libraries
  Scratches and Consol     Browse Type Hierarchy         ^H
                           Reformat Code                 ⌥⌘L
                           Optimize Imports              ⌥⌘O
                           Delete...                     ⊗

                           Build Module 'userinterface-test'
                           ▶ Run 'UserSettingsTest'           ^⇧F10
                           Debug 'UserSettingsTest'           ^⇧F9
                           Run 'UserSettingsTest' with Coverage
                           Rebel Run 'UserSettingsTest'
                           Rebel Debug 'UserSettingsTest'

                           Create 'UserSettingsTest'...
                           Reveal in Finder
                           Open in Terminal
Run:    com.vo in userint
  ▶ ✓ ⊘ ↓ᵃ↓ᶻ ⤒            Local History                   ▶
  v ⊘ vo (com)             Git                              ▶
    v ✓ User Settings      Reload from Disk
        ✓ Appearance
    v ✓ Dashboard Tes      Compare With...               ⌘D
        ✓ Dashboard s      Compare File with Editor
        ✓ Click on 'Cre
        ✓ Click on 'Fav    Mark Directory as                ▶
    > ⊘ TextField Creat
                           Create Gist...
                           JRebel                           ▶

                           Convert Java File to Kotlin File  ⌥⇧⌘K
```

- Each test produces a report in a simple table ascii format and generates screenshots
  Table report:
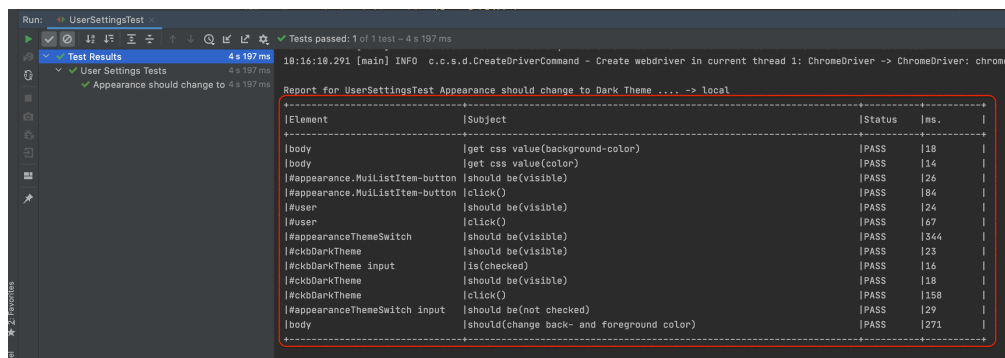
Screenshots in `buld/reports/tests` folder
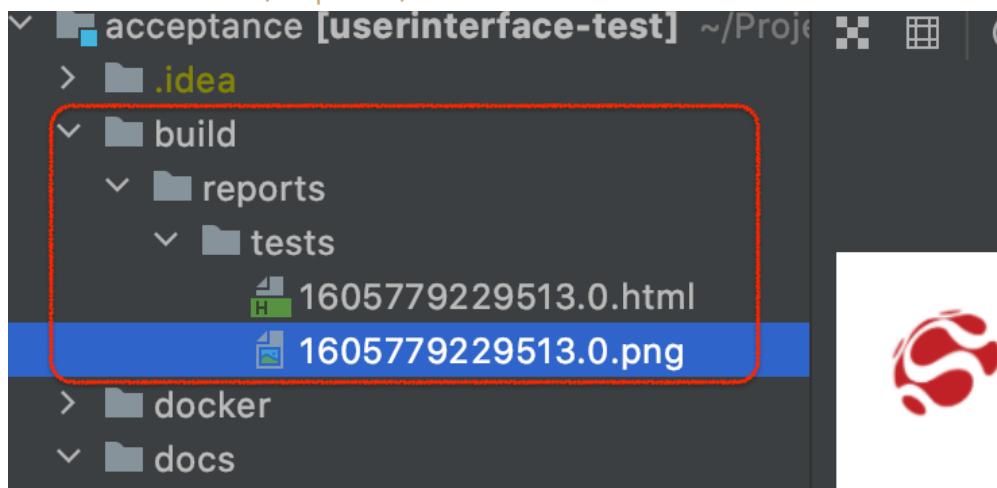
Best practices to write concise tests

1. Few lines as possible
2. Always chain element selectors with expectations and after that with actions. e.G. `$("#someId").shouldBe(visible).click()`
3. Always use following technics to select elements
   1. search for application related element ids. If not exists, ask developers to add it
   2. search for application related element css classes. If not exists, ask developers to add it
   3. if previous two points not possible (always check and follow up with developers first) use generated locators by chrome plugins (see later)
   4. as last fallback use xpath selectors. But they should be written in an relative manner and be safe against possible movements on the page.

Locating elements on the web page

Install following browser extensions to your Chrome browser:

- ChroPath - to easier lookup for CSS Selectors.
- XPath Helper - to test XPath expressions on te page.
- Selenium IDE - to make recordings and produce first approaches to skript the test.