Design Issues with existing codebase (find it in mater branch of the repo)

1. **Use of src/main/java** :- Maven is a build tool that allows any kind of code to be compiled and executed - both the application code as well as test code. Hence, the default artefact of maven provided 2 source packages, one for application code (src/main/java) and other for test code (src/test/java). The existing codebase has all frontend tests and hence should have been the part of src/test/java

2. **Use of default package** :- Packaging codebase is significant in terms of maintenance and debugging. It is always a good practice to use dedicated package nomenclature and structuring which signifies purpose and ownership of the codebase. Like com.testVagrant.tesCode.testClasses, com.testVagrant.tesCode.PageObjects etc.

3. **Code repetition/no reusability** :- All test classes have a lot of code that is common but repeatedly written in all of them, for e.g. setDriverPath() - definition of this method could be taken out and written in a single wrapper class which could be extended by all test classes and hence can be directly used.

4. **Use of hard waits (Thread.sleep)** :- Hard waits are unreliable and a overhead on automation scripts - the very basic fundamental of creating automation fails with introduction of Thread.sleep - we create automation scripts to do testing faster, i.e., as an when the resource is available, take action and move to the next step. But with hard waits, a static time is added for script to wait before engaging into any action. Application may behave differently with fluctuating internet connections - a hard wait of 10 seconds could be huge on one day and less on other, while making it mandatory for scripts to wait for that much time. Instead, dynamic waits can be used (implicit and explicit) to allow automation tool (selenium in this case) to wait upto a maximum time for carrying out an action while moving on to the next if current action can be taken earlier (as and when the resource/element is available)

5. **1-tier architecture of codebase** :- Tests, Page Objects and initializations - all are done in the same class which is a bad practice (plus a huge overhead with larger codebases). Rather, a layered architecture should be embedded with codebase to allow easy debugging and maintainence. All Page Objects should be added separately and test classes should just call action methods and do assert on them

6. **No explicit Logging** :- TestNg provides a logger that allows adding information to test reports - that is missing in this codebase. Logging helps a lot when tests fail, also they give information about tests getting passed with different data sets.

7. **Resources in root folder** :- Resource files like ChromeDriver executables must be kept in a resource folder and not directly in root of project - that is not a good organization of utilities

8. **Lack of uniformity** :- While One test class uses the Page Object pattern (@FindBy), the other test classes do not use it - there should be a uniform pattern across the scripts for carrying out all test steps.