Login credentials for the website qa avenue

Username: [shriram0205@gmail.com](mailto:shriram0205@gmail.com)

Password: test1234

Avenue Code To Do App is a web Application to create and manage day to day tasks for users. It has the following facilities.

1. To Create tasks for users.
2. Create and manage Sub tasks for a particular task.

The application was through test for these two use cases and the manual test case document was prepared stating the positive and the negative test cases. Bugs were also captured in the manual document as well as the bug tracker.

The following tests can be performed on the Application:

1. Functional Tests for the use cases mentioned in the use cases document.
2. Nonfunctional tests can be performed like the performance test, stress test by running the build on tools like JMeter to check the maximum concurrency allowed by the application.

We can improve the value of testing in this application by following a neat concise framework from the beginning

1. **What to automate**

Automating every test case is neither possible nor practical. So, test managers need to first identify cases that best play to automation’s strengths. Ideal test cases:

* Run frequently on multiple builds
* Involve copious amounts of data performing the same action
* Are prone to human error
* Feature high risk conditions for commonly used functionality
* Are difficult to perform manually
* Run on numerous hardware and software platforms & configurations
* Applications without a graphical user interface
* Composite and distributed applications
* Test cycles executed outside normal business hours

 2. **Create a plan**

Once the test cases are identified, they need to be incorporated into a comprehensive test strategy and plan. This plan should spell out the specific testing objectives as well as quality assurance goals.

Effective testing plans should break test cases down into logical and executable components focused on a single task, as opposed to application functionalities, which create large and complex test cases that are difficult to edit and debug. Focusing on small, bite-size activities enables you to maintain test scripts, test data and processes thereby improving productivity and collaboration. As well, it is easier to combine these simple automated tests into a larger automated test by application functional areas, common areas or a base set of data.

 3. **Choose the optimal tool**

The selection of a particular automation testing tool can be the difference between success and failure. The choice of the tool will be dependent on many business and technical factors. In general, the testing team should look for a tool that:

Supports your hardware, operating systems and development environment

Enables testing by all skill and experience levels

Is easy to use but does not lack key functionality

Provides for the creation of reusable, maintainable and robust automated test assets.

4. **Test early and often**

To maximize bug identification, automation testing should be undertaken as early as possible and ran as often as needed. Fixing bugs earlier in the SDLC is always less expensive than addressing them later in production or deployment. The optimum method would be to utilize a risk based assessment to prioritize the automation effort early towards the highest risk test cases.

 5. **Organize the testing**

Success with automated testing depends on getting the people equation right. For example, putting the right people in the right roles will help maximize testing productivity. The average quality assurance engineer will not be good at everything, such as writing both test scripts and test cases. Furthermore, getting the testers to collaborate with each other will improve testing productivity and problem-solving.

 6. **Develop scalable automated tests**

It is common for the UI to change between builds, especially early in the process. These changes may affect test results. Moreover, your automated tests may no longer work with future versions of the application. One of the UI challenges centers around how the testing tool identifies and locates an object on the screen (e.g., buttons, text fields, windows … etc.). When the UI changes the automated test script may be unable to locate the original object causing the test to fail. Quality assurance engineers would then need to update test scripts, before running the test against the latest version of the application. A better approach is using unique identifiers that can make the automated tests resistant to UI changes and makes the tests themselves more reusable.

 7. **Generate quality test data**

Creating test data for your automated tests can be cumbersome, time-consuming and boring. However, it is an essential task. The payoff is a much more streamlined testing process as well as the ability to easily maintain and reuse the automated tests. Furthermore, high quality data allows you to easily extend the automated test to different testing scenarios within the application’s development.

Like many other technologies, automation will deliver compelling value when effectively specified and deployed. This requires a team that is versant with testing best practices, a systematic quality assurance process and a practical understanding of automation testing tools. For organizations inexperienced with automation, we recommend starting with a clearly defined project and a limited number of test cases.

 8. **Consider advanced automation tools**

Companies may want to leverage Business Process Testing tools, which convert business processes into modularized testing components. Testing teams may also want to explore leveraging keyword-driven scripts and automated object recognition technologies that reduce the amount of rework that needs to be done.

**The structure of the code is as follows**:

I have used Page Object Model, Cucumber and Selenium to run the automated test cases. I can divide the section into three parts.

1. Login. Feature and its corresponding step definition file would be Login.java. This file contains all the necessary page objects and the page methods related to Login. LoginPage.java and LoginPageConstants.java are related to this.
2. CreateTasks. Feature and its corresponding step definition file would be CreateTasks.java. This file contains all the necessary page objects and the page methods related to create task. CreateTasksPage.java and CreateTasksPage Constants.java are related to this.
3. CreateSubsTasks. Feature and its corresponding step definition file would be CreateSubTasks.java. This file contains all the necessary page objects and the page methods related to creating sub tasks for a particular. CreateSubTasksPage.java and CreateSubTasksPage Constants.java are related to this
4. CreateTasksBugs1, CreateTasksBugs2, CreateTasksBugs3 features are for the captured bugs using the bug tracker and their corresponding step definition classes are inside the step definition package.
5. SubTasksBugs1, SubTasksBugs2, SubTasksBugs3 features are for the captured bugs using the bug tracker and their corresponding step definition classes are inside the step definition package.

**Some of the essential features for a bug tracker and the changes I would want in the bug tracker embed in the To Do Application**

* Communication between the developer and the user.
* Ability for the user to assign certain bits of information such as severity (how much that bug relates to them).
* Ability for the developer to override that priority and, if possible, give a reason.
* Ability to assign tasks to a developer.
* Ability to sort between bug, enhancement, and feature request. The difference between an enhancement and feature request is very subtle but VERY important.
* Ability to attach files (such as screen shots)
* Ability to have custom fields (such as being able to select which OS, which service pack level, application version, etc).
* Privacy. Some items, such as security exploits or information that deals with financial information, will need to be kept secret. Even OSS does this from time to time until they can get a patch ready. Everyone has their own rules.
* Ability to show the changes between revisions so you can email out a Change Log so users know what you have and have not done.
* Reminders about which items are left undone and are assigned to you / unassigned at all.

I have worked with bug tracking tools like Bugzilla and I feel all these should be incorporated to make it a great application.