Task 1

Imagine the following situation: you are joining a cross-functional team, which builds, a front-end application using REST APIs. You are a first QA engineer and need to establish a QA process in the team.

Question 1: What would you do in your first few days of work? Where would you start?

Solution:

In first few days we will be developing test strategy and test plan to implement test strategy. We will achieve this by documenting the following

1. Before writing any test strategy, first we will do the requirement analysis to develop a strong business understanding. We will achieve this through discussion with business people, client and core development teams.
2. We will also do our exploratory testing on similar type of product/requirements available in market and will analyze the shortcomings. By using comparative analysis, we will define our test strategy. In addition, it is not mandatory that we should have similar product in market, so in this case may be we can utilize our own experience or team experience or we can foresee/imagine by using our technical and business knowledge to identify bottlenecks and their solutions.
3. We will generate the uses-cases from requirements not only from client perspective but also explore the hidden uses cases. Will also note down all the flows and paths that a user can follow on front-end. We will use these use cases to write test case scenarios.
4. Other than test plan and test strategy document, we will be utilizing BDD (behavior driven development) to write up test cases in feature file using Gherkin. When developers starts coding mean time testers should start writing test cases.
5. While writing test cases, we will also define automation strategy or road map. We can have discussions among team on prioritizing the automation of test cases.
6. We will also do the dependency analysis from testing point of view, like if the new functionality is ready but is dependent to other functionality for testing, in this case we will prepare bot/mock service to remove the dependency
7. After compiling possible test cases, we will add tags on priority bases like high, low or medium. We can also use other tags like positive, negative, smoke, sanity, regression, unit, end to end, functional, non-functional

Question 2: Which process would you establish around testing new functionality?

Solution: I will be doing following manual/automated testing activities on UI layer and I will doing assertions of UI layer with database and service layer where required.

1. Firstly, of all we will run smoke test cases to decide application under test is ready for detail testing. For this, we will run automated test suite.
2. Secondly we will run functionality test cases to test new module, here I will perform manual testing first then we will go for automation
3. After that, we will be doing integrated testing by running end-to-end test cases. I will leverage automation here.
4. Then we will perform automated regression testing

Question 3: Which techniques or best practices in terms of code architecture and test design would you use in your automated tests?

Solution:

Strategy:

We will design a centralized automation framework for performing automated testing different application instances (test, stage, prod) on different platforms (web, android, IOS) at different testing layers (UI, API, Databases).

Framework Architecture Detail:

1. We will automate our application on the three layers approach. **UI** layer, **Database** layer and **API** service layer. We will design and automate test cases to test these three layers independently. Most of the time issues lie underneath API or database layer rather than UI layer. In addition, automation cost (time/efficiency) of UI layer is more than API.
2. We will follow the **page object mo**del by dividing our application on the bases of modules and value delivered to end user.
3. We will convert these modules in to test suites and test suites in to test cases. **Classes** will represent test suite and **functions** will represent test cases.
4. We will strictly follow object orientation approach to avoid redundancy and maintenance cost. We will use modular approach and segregate the components, which needs segregation, in order to increase enhancements and integration for future. Components will be like feature files, step files, logic files, xpath, test data, test case output, error logs files etc. will be stored in different directories

5. We will utilize exception handling and will store all of the exception in the centralized repo like a files or database. On same rule, our framework will maintain logs for backtracking and debugging of failed test cases because suppose when you have ran thousands of test cases simultaneously and a test case gets failed then we have to back track either its due functionality crash or automation failure. This approach will make backtracking easier.
6. We will be utilizing Behavior Driven Development using **Gherkin** and write the test cases in **GIVEN**, **WHEN THEN** form
7. We will be storing output of our test cases in database or CSV files. Based on test result we will be developing centralized UI dashboard (node.js), which will show output in different visualization. We will provide filters to create custom reports based on number of passed test cases, failed one, high priority, low priority, passed regression test cases, failed lower priority test cases etc.
8. We will integrate our test case with **CI/CD** pipeline. The Jenkins will trigger a testing job while pushing the development code from development environment to test environment. Testing job will get the latest automated code from **git** repo and it will deploy it to the testing server and after job completion, it will return pass or fail flag. If the flag is true then developer code would be pushed to staging environment or otherwise revert.

Implementation:

For implementation, we can choose any language like python, java or C#. I have experience in all of three languages but python and java have more support regarding to automation libraries. Considering python as implementation language then we can utilize following supporting libraries for creating our own automation framework.

- **selenium** library for web automation
- **appium** library for android and IOS app automation
- **request** and **pact-test** library for API testing
- **pandas** and **sql** libraries for data testing
- **py-test** library as test runner and **pytest-bdd** library for following gherkin approach
- bash scripting for creation of testing job
- **linux** server (Cent-OS/Ubuntu) with **docker** for deployment of automation script
- **git** repository for holding automation code
- **Jenkis** for scheduling and triggering automation job
- **Node.js** for creating dashboard page for displaying and filtering test results
- Python **smptp** module for sending emails to stake holders
- **Pycharm** as IDE (Integrated Development Environment)

Architecture Diagram

Below is my proposed architecture. In this architecture, we will have three core components **inputs**, **automation engine** and **outputs**. Automation engine will take inputs and process them to produce outputs.