**What are story points?**

Story points are a way to estimate the amount of effort required to complete a user story in your product backlog. You'll usually estimate story points before a sprint planning meeting, since that's when your team determines how much work they can carry out in an upcoming sprint.

**How do you estimate the Story Points.**

As a QA Lead, my role in Sprint Planning involves ensuring that the team has a clear understanding of the scope of work and can realistically estimate the effort required to complete each user story. Story points are a tool we use to estimate this effort, focusing on factors like complexity, uncertainty, effort, and dependencies.

Here's how I approach defining story points in Sprint Planning:

* **Understanding User Stories**: I start by making sure everyone on the team understands each user story's requirements and acceptance criteria. If there .are any ambiguities, we address them during Sprint Planning to avoid misinterpretation later.
* **Relative Estimation with a Baseline**: I choose a consistent scale for story points, often the Fibonacci sequence (1, 2, 3, 5, 8, etc.), which represents increasing levels of uncertainty. To ensure consistency, we select a baseline story with a known level of effort. This serves as our reference point for estimating other stories.
* **Using Planning Poker**: To estimate story points, we use Planning Poker, a collaborative technique where team members privately select a card representing their estimate for the story's effort. When everyone reveals their cards, we discuss any discrepancies and aim to reach a consensus. This method promotes open discussion about complexity and effort.
* **Considering All Relevant Factors**: In assigning story points, we consider factors like:
  + **Complexity**: Is the task straightforward or does it involve complex logic or multiple components?
  + **Uncertainty**: Are there unknowns or new technologies involved?
  + **Effort**: How much work is required? Does it involve extensive coding, testing, or both?
  + **Dependencies**: Does the story depend on other work or external resources?
* **Aligning with Team Velocity**: Once we've estimated all user stories for the Sprint, we compare the total story points with the team's average velocity from previous Sprints. This helps us ensure we don't overcommit and set realistic expectations for the Sprint.
* **Adjusting the Sprint Scope**: If the total story points exceed the team's capacity, I work with the Product Owner to adjust the Sprint backlog, ensuring a manageable scope. We prioritize stories to focus on what's most important and achievable within the Sprint timeframe.
* **Continuous Improvement**: During Sprint Retrospectives, I encourage the team to reflect on the accuracy of our story point estimates. We discuss what worked well and identify areas for improvement. This feedback loop helps us refine our estimation process for future Sprints.

By following this approach, I ensure that Sprint Planning is collaborative, transparent, and results in a clear, achievable plan that aligns with the team's capacity and goals.

**How do you assign the tasks in your team?**

As a QA Lead, I assign tasks to my team by considering each member's skills, expertise, and current workload. Here's my approach:

1. **Understand Project Scope and Priorities:**
   * I start by understanding the project goals, priorities, and key deliverables. This helps identify which tasks need immediate attention and which can be deferred.
2. **Evaluate Team Skills:**
   * I assess each team member's strengths and areas of specialization. This allows me to assign tasks to those best equipped to handle them while ensuring a balance of workload.
3. **Use Collaboration Tools:**
   * I use tools like Jira or Trello to assign tasks and track progress. This keeps the process transparent and allows everyone to see what needs to be done.
4. **Balance the Workload**:
   * I aim to distribute tasks evenly to avoid overburdening any team member. If the workload becomes uneven, I adjust assignments to maintain balance.
5. **Encourage Collaboration and Cross-Training**:
   * I promote teamwork and cross-training to ensure that knowledge is shared and there are no single points of failure. This also allows for flexibility if someone is unavailable.
6. **Regular Check-ins and Feedback:**
   * I hold regular team meetings to discuss progress, address any issues, and reassign tasks if needed. This ensures that everyone is on track and has the support they need.

By following these steps, I ensure that tasks are assigned effectively, the team remains productive, and we can deliver high-quality work on schedule.

**What is the size of Regression test cases?**

In a recent project, our **regression test suite** contained a total of about **200 test cases**. Out of these, **150** were **automated** and **50** were **manual**. The automated cases **focused on stable, repeatable scenarios**, while the manual cases covered more complex, nuanced, or **new functionality** that required human judgment. This balance allowed us to achieve efficient regression testing while maintaining flexibility for exploratory testing and edge cases.

**Why should you go with Microservices? What is the difference between micro services and regular API?**

**Real-Time Airline Project Example**

In a real-time airline project, consider the following scenario:

* The airline uses microservices to manage different parts of its operations. There's a **flight booking** microservice that handles **customer bookings**, a **seat selection** microservice for choosing seats, a **loyalty program** microservice for managing customer rewards, and a baggage tracking microservice to monitor luggage.
* If the airline experiences high booking demand during peak travel seasons, it can scale the flight booking microservice without impacting other services. Similarly, if the airline wants to update the loyalty program with new features, it can do so without affecting flight booking or baggage tracking.
* This modularity also allows the airline to recover quickly from failures. If the seat selection service goes down, it doesn't affect flight bookings or baggage tracking, ensuring continuity of operations.

In contrast, with a regular API-based monolithic approach, a failure in one component could cascade, affecting other parts of the system. Additionally, deploying updates or scaling resources would require dealing with the entire application, increasing risk and complexity.

**Conclusion**

Microservices offer flexibility, scalability, resilience, and easier deployment compared to regular API-based monolithic architectures. In an airline project, these benefits translate into a system that's better equipped to handle high demand, rapid changes, and isolated failures, leading to improved operational efficiency and customer experience.

**Can you explain how you implemented your web browser for parallel execution?**

To implement parallel execution for web browser-based testing, I used Selenium Grid to set up a distributed testing environment, allowing me to run multiple tests simultaneously across different browsers and platforms. Here's the approach I took:

**Set Up Selenium Grid:**

I established a "hub" on a central server to manage the distribution of test cases.

Configured multiple "nodes" on remote machines to run the tests. These nodes were set up with various browsers (like Chrome, Firefox, Safari, etc.) and operating systems to ensure comprehensive coverage.

**Configure Desired Capabilities:**

I specified the desired capabilities for each test, including browser type, version, and operating system. This ensured that the tests were executed in the correct environment.

**Write Test Scripts for Parallel Execution:**

The test scripts were designed to run in parallel, ensuring they were independent and didn't share states or data to avoid conflicts.

I used test frameworks with built-in support for parallel execution, like TestNG or JUnit, to control the level of concurrency.

**Distribute Test Load:**

The test suite was sent to the Selenium Grid hub, which distributed the tests among the available nodes. This allowed us to run multiple tests at once, reducing overall execution time.

**Monitor and Aggregate Results:**

I implemented logging and monitoring to track the progress of each test running in parallel.

After execution, I aggregated the test results to generate comprehensive reports, allowing for efficient analysis and identification of any issues.

This approach enabled us to execute web browser tests in parallel, significantly reducing testing time and increasing test coverage across different environments. It also provided scalability, allowing us to add more nodes as needed to handle larger test suite.