Sample Unit Tests

Filter Freelancers by Job Count

```
javascript
const filterByJobCount = (freelancers, min, max) => {
  return freelancers.filter(
    (freelancer) => freelancer.jobCount >= min && freelancer.jobCount <=
max
  );
};

test("Filters freelancers by finished job count", () => {
  const freelancers = [
    { name: "Omer", jobCount: 5 },
    { name: "Ornek", jobCount: 10 },
    { name: "Odine", jobCount: 15 },
  ];

  const result = filterByJobCount(freelancers, 5, 10);
  expect(result).toEqual([
    { name: "Omer", jobCount: 5 },
    { name: "Ornek", jobCount: 10 },
  ]);
});
```

Fetch Comments for a Specific Job

```
javascript
const fetchCommentsForPost = (comments, postId) => {
  return comments.filter((comment) => comment.postId === postId);
};

test("Fetches comments for a specific post", () => {
  const comments = [
    { postId: 1, body: "Great work!" },
    { postId: 1, body: "Excellent!" },
    { postId: 2, body: "Well done!" },
  ];

  const result = fetchCommentsForPost(comments, 1);
  expect(result).toEqual([
    { postId: 1, body: "Great work!" },
    { postId: 1, body: "Excellent!" },
  ]);
});
```

Assumptions and Notes

- The mock data from JSONPlaceholder provides a sufficient basis for testing the primary functionalities.
- Placeholder attributes (e.g., photos) can be simulated using default URLs.
- All submission processes are simulated, as no real backend integration exists.
- Non-functional requirements like responsiveness and accessibility are considered in manual testing plans.

Ömer Örnek