**CS673 Software Engineering** 

**Team 4 - ResumAI**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Hemant Krishnakumar | Team Lead |  | 11-07-2024 |
| Faizan Ahmad | Design and Implementation Lead |  | 11-07-2024 |
| Tushar | Requirement Lead |  | 11-07-2024 |
| Shubh Gupta | Q/A Lead |  | 11-07-2024 |
| Amruth Reddy | Security Lead |  | 11-07-2024 |
| Jaindra Parvathaneni | Configuration Lead |  | 11-07-2024 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1** | **Shubh Gupta** | **11-07-2024** | **New** |
|  |  |  |  |

[Testing Summary](#_sm5odwyvuk3j)

[Manuel Tests Reports](#_pqso2mbjyzx4)

[Automated Testing Reports](#_mtfbusfb0eq3)

[Testing Metrics](#_rijyjeu2ojqa)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Testing Summary

* **Tested Components**: Key frontend and backend units like login, registration, and recruiter shortlist.
* **Involved**: Development team, with QA input on test cases.
* **Timing**: Unit tests during development; automated tests on every commit via Husky.
* **Techniques**: Unit testing with Jest, Husky pre-commit hooks, mocking, and snapshot testing.
* **Results**: All unit tests passed; minor issues fixed; Husky successfully blocks commits with errors.

# Manual Testing Report

**Test Case 1: Login Page**

* **New or Old**: New
* **Test Items**: Verifying login functionality with correct and incorrect credentials
* **Test Priority**: High
* **Dependencies**: Registration Page
* **Preconditions**: User must be registered in the system
* **Input Data**: Valid and invalid email/password combinations
* **Test Steps**:
  1. Navigate to the login page.
  2. Enter valid credentials and submit.
  3. Enter invalid credentials and submit.
* **Postconditions**: User is redirected to the homepage on successful login, receives an error on failure.
* **Expected Output**: Successful login redirects to the homepage; unsuccessful login displays an error message.
* **Actual Output**: Redirects to the homepage on success; error message on failure.
* **Pass or Fail**: Pass
* **Bug ID/Link**: N/A
* **Additional Notes**: N/A

### **Test Case 2: Registration Page**

* **New or Old**: Old
* **Test Items**: Verifying registration with valid and invalid data
* **Test Priority**: High
* **Dependencies**: Login Page
* **Preconditions**: None
* **Input Data**: Various email/password combinations, including valid, invalid, and duplicate data
* **Test Steps**:
  1. Navigate to the registration page.
  2. Enter valid user details and submit.
  3. Attempt registration with duplicate email and invalid data.
* **Postconditions**: Registration completes successfully, or the user is informed of errors.
* **Expected Output**: Successful registration redirects to the login page; errors are shown for invalid data.
* **Actual Output**: Works as expected.
* **Pass or Fail**: Pass
* **Bug ID/Link**: N/A
* **Additional Notes**: Check for edge cases with duplicate registrations.

### **Test Case 3: Analytics Page**

* **New or Old**: New
* **Test Items**: Verify analytics page data loads and displays correctly
* **Test Priority**: Medium
* **Dependencies**: None
* **Preconditions**: User must be logged in
* **Input Data**: N/A
* **Test Steps**:
  1. Log in and navigate to the analytics page.
  2. Verify that data loads correctly for each category.
* **Postconditions**: Data appears as expected.
* **Expected Output**: All graphs and data should load and be interactive.
* **Actual Output**: Data loads with slight delays in some sections.
* **Pass or Fail**: Pass
* **Bug ID/Link**: N/A
* **Additional Notes**: Consider optimizing load times.

### **Test Case 4: Shortlist Page**

* **New or Old**: New
* **Test Items**: Verify adding and removing candidates from the shortlist
* **Test Priority**: Medium
* **Dependencies**: Job listing page
* **Preconditions**: User must have viewed job listings
* **Input Data**: Candidate profiles
* **Test Steps**:
  1. Navigate to the shortlist page.
  2. Attempt to add candidates to the shortlist.
  3. Remove candidates from the shortlist.
* **Postconditions**: Candidate is added or removed from the shortlist.
* **Expected Output**: Candidates appear in the shortlist; removal works as expected.
* **Actual Output**: Works as expected.
* **Pass or Fail**: Pass
* **Bug ID/Link**: N/A
* **Additional Notes**: Ensure shortlist updates in real time.

### **Test Case 5: Job Listing Page**

* **New or Old**: Old
* **Test Items**: Verify job listings load and filters function correctly
* **Test Priority**: High
* **Dependencies**: Login Page
* **Preconditions**: User must be logged in
* **Input Data**: Job titles, company names, and filters
* **Test Steps**:
  1. Navigate to the job listing page.
  2. Apply various filters and sort options.
  3. Open specific job listings.
* **Postconditions**: Filtered jobs load and details display correctly.
* **Expected Output**: Job listings display as per applied filters.
* **Actual Output**: Works as expected.
* **Pass or Fail**: Pass
* **Bug ID/Link**: N/A

### **Test Case 6: Landing Page**

* **New or Old**: Old
* **Test Items**: Verify landing page content and responsiveness
* **Test Priority**: Low
* **Dependencies**: None
* **Preconditions**: None
* **Input Data**: N/A
* **Test Steps**:
  1. Navigate to the landing page.
  2. Check content and responsiveness across devices.
* **Postconditions**: Page should load quickly and be responsive.
* **Expected Output**: Content displays correctly on all screen sizes.
* **Actual Output**: Works as expected.
* **Pass or Fail**: Pass
* **Bug ID/Link**: N/A
* **Additional Notes**: Test on different browsers and devices.

# Automated Testing Report

### **Husky in Automated Testing**

* **Purpose**: Husky is used to enforce pre-commit hooks, automatically running tests and checks to ensure code quality before each commit.
* **Pre-commit Hooks**: When a developer commits code, Husky triggers:
* **Jest Tests**: Runs unit tests to verify core functionality and catch issues in new changes.
* **Nodetap Tests**: Executes server-side tests, particularly for Node.js components, ensuring backend stability.
* **Linting**: ESLint checks for code style and formatting errors.
* **Commit Blocking**: If any test fails, Husky blocks the commit, ensuring issues are fixed before pushing code.
* **Configuration**: Configured in the .husky directory at the project root, specifying commands for linting, Jest, and Nodetap tests in the pre-commit hook.

Husky’s integration ensures consistent code quality by catching potential issues early in the development workflow.

# References

1. Jest Documentation - <https://jestjs.io/docs/getting-started>
2. Husky Documentation - <https://typicode.github.io/husky/#/>
3. Nodetap Testing Framework - <https://www.npmjs.com/package/tap>
4. ESLint Documentation - <https://eslint.org/docs/user-guide/getting-started>

# 