# System Design Document (SDD)

**Project Name:** JobAIAssistant  
**Prepared By:** Mohammad Afzal  
**Date:** May 17, 2025

## 1. System Overview

JobAIAssistant is an AI-powered application that automates job applications across multiple job portals (e.g., LinkedIn, Naukri). It enables users to configure their resume and job preferences through a web dashboard while visually watching the automation run in a real browser. Human supervision is enabled for manual intervention (e.g., login, CAPTCHA, submission confirmation).

## 2. Key Components

* Web Dashboard
  + User profile input (resume, skills, CTC, locations)
  + Configuration of preferences (job roles, auto-apply, etc.)
  + Job application history and monitoring
* Automation Engine (Local Agent)
  + Uses Playwright to launch browser in visible mode
  + Applies for jobs based on search results
  + Interacts with web elements, forms, and redirects
* Communication Layer
  + WebSocket or polling mechanism
  + Syncs commands from dashboard to local bot
  + Sends status updates and wait states (e.g., "awaiting login")
* Data Layer
  + Stores user configuration, resumes, application logs
  + Can use local JSON/YAML or a database (e.g., SQLite, PostgreSQL)

## 3. User Flow Scenarios

### Scenario 1: Apply to Internal Job Portal

* Automation navigates job portal (e.g., LinkedIn Easy Apply)
* Auto-fills form fields and uploads resume
* Waits for user confirmation before final submission

### Scenario 2: Redirect to External Website

* Clicking "Apply" opens new tab or navigates to external URL
* Automation detects new tab and parses the page
* If login/signup is required:
  + User manually logs in
  + System resumes after detecting login completion
* Form is parsed and pre-filled
* Waits for user to review before submission

## 4. Automation Flow

* Fetch job post from job portal
* Determine whether it's internal or redirects externally
* Launch visible browser window
* Navigate to job post or external apply link
* Detect new tab (if opened)
* Parse and fill form fields
* Prompt user to:
  + Confirm details
  + Edit fields
  + Solve CAPTCHA or log in if needed
* Resume and submit form
* Log application and continue to next

## 5. Human Intervention Points

* Login / Signup Required
* CAPTCHA detection
* Submission confirmation
* Form field mapping uncertainty
* File upload or resume change suggestion

## 6. Redirect and Tab Handling

* Playwright context listens for new page event (new tab)
* Switches focus to new tab
* Analyzes loaded content:
  + URL pattern
  + Form presence
  + Login or other obstacles
* Uses dynamic selectors or text-based detection for forms

## 7. Form Mapping and Autofill Logic

* Input fields are identified by:
  + label elements
  + placeholder text
  + Nearby descriptive text
* Uses key-value mapping from user profile
* If field is unknown, prompt user and save field mapping for future reuse

## 8. Sync Between Dashboard and Bot

* WebSocket (preferred) or polling from local bot to server
* Commands pushed from dashboard (start, pause, resume)
* Bot sends status back (job found, form filled, awaiting input)
* Shared state (e.g., status.json) can also be used for minimal setups

## 9. Error Handling and Logging

* Unexpected page structures: fallback to manual
* Form not detected: notify user
* Timeout on login: abort and log
* CAPTCHA: pause and prompt user
* Logs stored per application session with screenshots

## 10. Future Enhancements

* Resume session after crash
* Smart form classifier using LLM or ML model
* Screenshot-based diffing for form changes
* Job scoring feedback loop
* Mobile alerts for manual steps (via push or Telegram)

### End of System Design Document (SDD)