Technical Approach – LinkedIn Data Scraper

Objective The goal of this project is to build a tool that extracts:

1. Employee information from a LinkedIn company page.
2. Public profile information from a LinkedIn profile page.

The focus is on demonstrating problem-solving, technical approach, and handling dynamic platforms, rather than bypassing LinkedIn restrictions.

---

1. Tools & Technologies

- Node.js – backend runtime
- Puppeteer – browser automation and scraping
- JavaScript – main programming language
- fs / path – for saving JSON outputs
- Cookies (session) – optional for logged-in scraping

---

2. Scraping Strategy

#Company Scraper

1. Navigate to `https://www.linkedin.com/company/{company}/people/`.
2. Wait for DOM content to load and React to render.
3. Scroll the page to trigger lazy loading of employee cards.
4. Extract visible profile links and names:
    - Filter out UI text like `Following` or `· 1st`.
    - Deduplicate repeated profiles.
5. Fallback: if no employees are found:
    - Open a verified profile.
    - Check company association via the "Experience" section.
    - Return at least one verified employee.
6. Output JSON contains:
    - `name`, `title`, `location`, `profileUrl`, and a `verified` flag.

#Profile Scraper

1. Navigate to a public profile URL.
2. Wait for page load and React hydration.
3. Extract:
    - Name, headline, location, connections
    - Experience, education, skills
4. Use safe selectors with defensive coding to handle missing fields.
5. Output JSON matches the assignment structure.

---

3. Handling LinkedIn Restrictions

- Dynamic content: LinkedIn uses React; content may not be immediately in the DOM.

- Lazy loading: Scroll simulation ensures additional profiles load.
- Access limits: Some fields require login.
- Rate limits: Added small delays (`delay(ms)`) to reduce temporary blocks.
- Ethical fallback: Verify at least one employee via public profile if `/people` page is restricted.

---

4. Pagination & Infinite Scroll

- Employee lists are loaded via infinite scroll.
- Scroll is simulated in steps with delays to load more profiles.

---

5. Error Handling

- Defensive DOM queries (`querySelector`, `querySelectorAll`) with optional chaining.
- Empty/restricted fields are set to `null`.
- Warnings logged if employees cannot be extracted.

---

6. Scalability Considerations

- Modular functions separate navigation, extraction, and parsing.
- Can scale with queues or concurrent browser contexts for multiple companies/profiles.
- Session management allows reuse of logged-in cookies.

---

7. Limitations

- Scraping may return partial data due to LinkedIn restrictions.
- Some fields may not be visible in public view.
- JSON output can vary depending on account type and session state.

---

8. Optional Enhancements

- CSV export of employee data.
- Seniority or department inference from job titles.
- Queue or concurrency system for large-scale scraping.

---

9. Summary The scraper is a best-effort, ethical implementation:

- Produces structured JSON outputs.
- Includes fallback mechanisms.
- Demonstrates knowledge of web automation, dynamic DOM handling, and backend design.