Full Stack Interview Questions:

Introduction: Create a web scraper that overcomes anti-scraping measures, handles dynamic content, implements automated testing, and displays data through a web app.
Requirements: TypeScript, Node.js, Puppeteer, Redwood.js

Task 1: IP Rotation and Anti-Scraping Integration: Create a web scraper to fetch content from **2 popular blog sites** with anti-scraping measures (e.g., Coinbase Blog).
Requirements:

- Perform scheduled high-volume scraping.
- Use proxy services to rotate IP addresses (any costs will be covered by the team).
- Mimic human behavior with random delays and interactions.
- **Note:** Refer to the example code provided below for guidance.

Task 2: Handle Pagination, Infinite Scrolling, and Errors

- **Pagination Handling:** Navigate through paginated content effectively.
- **Infinite Scrolling:** Implement scrolling to load dynamic content.
- **Content Loading and Error Handling:**
    - Wait for elements to load before extracting data.
    - Implement retries for transient failures.

Task 3: Automated Testing and Continuous Integration

- **Objective:** Write tests to ensure the stability of the scraper and handle website changes.
- **Requirements:**
    - Create automated tests that validate scraper functionality.
    - Configure tests to run as a GitHub Action.

Task 4: Front-End Application using Redwood.js

- **Objective:** Develop a simple application to display the scraped content.
- **Requirements:**
    - Use Redwood.js to build the front end.
    - Ensure the application cleanly presents the data you've collected.

EXAMPLE CODE BELOW

```
export default async function scrapeBlog(
  config: IBlog,
  existingPage: Page | null = null,
  limit: number | null = null
) {

  puppeteer.use(StealthPlugin());
  let browser = await puppeteer.launch();
  let page = await browser.newPage();

  await page.goto(`${config.blogUrl}${config.indexPage}`, {
    waitUntil: "networkidle2",
  });

  let blogLinks = await page.evaluate((config) => {
    return eval(config.articleLinkSelector);
  }, config);

  if (browser) {
    await page.close();
    await browser.close();
  }
  // Cut array
  if (limit) blogLinks = blogLinks.slice(0, limit);
  const existingArticleLinks = await Article.find(
    { dataSourceId: config._id },
    'articleUrl'
  );
  const existingArticleLinksSet = new Set(
    existingArticleLinks.map((a) => a.articleUrl)
  );
  const newLinks = blogLinks.filter(
    (url: string) => !existingArticleLinksSet.has(url)
  );

  for (const url of newLinks) {
    console.log("Adding article to queue", url);
    await articleQueue.add(url, { url, config });
  }
  return blogLinks;
}
```