**Take-Home Technical Assessment**

Please see below for the details of the assessment. You have one week to return it to me, and it does not need to be completed in a single sitting. Feel free to let me know if you have any questions. Looking forward to seeing the results!

1. Navigate [here](https://app.triallibrary.com/trials/opportunities?zip5_code=89145&radius_in_miles=250) to our signed out sponsored trial opportunities page.
2. Poke around and familiarize yourself with the page.
   1. Focus on the content below the header (i.e. don't bother with the logo, navigation items, sign in button).
   2. Feel free to attempt to break it in an ad hoc manner via the browser UI (i.e. don't do something like write a script to load the page 100x/second ). Report any issues.
3. Write a test plan for the underlying API call, trial-search GET.
   1. In the Network tab of any browsers' developer tools, you can view the actual API requests you implicitly make when changing zip or radius via the UI.
   2. Obviously, you don't have visibility to the underlying BE code or what the FE code expects from the BE. The purpose here is to see how you'd approach testing it.
   3. Provide a copy of the test plan.
4. Write (working) python code to execute a few backend tests (presumably which are part of the test plan you created in the previous step).
   1. You should be able to access the API directly-- it's not locked down to the domain (yet).
   2. Please don't execute any performance tests, but you're free to run tests that execute the endpoint one at a time.
   3. The code can be as short or as long as you want it to be.
   4. Provide a copy of the code
5. Write front-end tests for the sponsored trial opportunities page
   1. Using the same page you explored in steps 1-2, write automated tests that verify the front-end functionality from a user's perspective. You should treat this as "black box" testing - use only what you can observe through the browser interface and developer tools.
   2. Requirements
      1. Write tests using a modern front-end testing framework (Jest + React Testing Library, Playwright, Cypress, or similar)
      2. Focus on testing user interactions and the integration between the UI and the API you analyzed in step 3
      3. Include tests for both successful scenarios and error handling
      4. Your tests should be runnable with standard commands (e.g., npm test, npx playwright test)
   3. What to provide
      1. Working test code with setup instructions
      2. Brief explanation of your testing approach and framework choice
      3. Any assumptions you made about expected behavior
      4. Instructions for running the tests
   4. Notes
      1. Don't worry about achieving 100% coverage - focus on the most important user scenarios
      2. Feel free to mock API responses for consistent test results
      3. If you encounter any bugs during your exploration (step 2), write tests that demonstrate the expected vs. actual behavior