# **Technical Test for the position of Full Stack JavaScript Programmer with React**

Hi, and thanks for participating in the selection process for this position.

This is a entry-level, yet comprehensive technical test for the position of Full Stack developer in our organization. The technologies that we are going to be using on this test are our bread and butter for building web and mobile cloud-based applications.

Some things to consider:

- We consider that with the common knowledge and proactive, and experienced developers can finish this test in one single day, however, we understand that constraints can be faced due to already scheduled responsibilities, so, you will have a maximum of 3 days to deliver this test.
- Keed in mind that more candidates are doing the same exact test, and whereas there is a benefit in finishing faster, and will be relevant for the evaluation, we give more value to other aspects of the evaluation (mentioned below)
- **Elements to be evaluated:**
    - Functionality
    - Code Structure, naming and organization
    - Data validation
    - User Experience: Loading indicators, proper UX
    - Code Documentation
    - Tests
- **Strategy to the evaluation:**
    - Frontend and backend should be committed to GitHub: 2 repos or a mono repo.
    - The system should run locally on a computer, both backend and frontend. And connect to a cloud Database
- VIDEO EXPLANATION: Record a 5 minute video demonstrating the app and its features and its connection to the frontend and backend code. The video has to be in English.
- Feel free to contact us if you have any questions in how to solve this test.

**<u>Proposition:</u>**

1) This is going to be a Task List with Authentication and a Task CRUD:

- Create Task
- Edit Task
- Delete Task
- Mark a task as Done (Using a Cloud Function or API web service, or a Custom Resolver)

2) The test frontend code needs to be built using Typescript with a ReactJS Framework:

- Vite
- NextJS
- React Native

3) For the backend code, it needs to be built using a Typescript backend web framework:

- Express Web Framework (NodeJS)
- NextJS

4) The communication between backend and frontend needs be using one of these 3 strategies:

- Grapqhl
- JSON REST API
- RPC

REQUIREMENT: Communication between Frontend and backend should be protected or authenticated. Example: Authentication Token

REQUIREMENT: You can use a Cloud Database: planetscale.com (MySQL) mongodb.com (MongoDB)

5) Create / Edit tasks must consider data validation and consistency

6) Enforce additional Task Rules:

- A task can have statuses: PENDING → IN_PROGRESS → DONE → ARCHIVED (Invalid transitions must be rejected)
- Only the task owner can mark it as DONE
- A task marked as DONE cannot be edited (except title typo fix)

**7) Concurrency Requirements: (Only for candidates opting for a mid-level or higher position)**

- Prevent marking a task as DONE twice

- Handle concurrent requests: Two requests attempt to mark the same task as DONE
- The markAsDone endpoint must be idempotent
- Log all api activities with timestamp and actors using a middleware strategy: include input (parameters, query params, headers) and the output (status code, etc)

8) Authentication has to be done using an Authentication provider:

- Auth0
- AWS Cognito (Preferred)

9) For the UI of the project, you have to use a components library:

- Material UI
- Bootstrap
- Ant Design

10) For the specific task of Mark a Task as Done is required to be using a Cloud Function, Resolver (Graphql) or Web Service

11) Automatic Tests: Is required to implement at least 2 different tests

- A Unit Test of the backend code using JEST or MOCKA
- An e2e Test using CYPRESS or SELENIUM (this test could be just the login or something else simple)
- A testing tool or framework of your choice