* You have **48 hours** to complete the test. DON'T push any code after delivering it in this system.
* Your solution MUST include automated tests for both the frontend and backend. Having good coverage and testing all the features is part of the test.
* You should submit your solution as a PRIVATE GitHub repository and invite  [**projects@shawandpartners.com**](mailto:projects@shawandpartners.com) as collaborator. You can also try by username sp-tests.
* DON'T create 2 repositories, make sure to include all code in the same GitHub repository. Create a "frontend" and "backend" folder inside your repository and code directly inside them. We'll run cd backend && npm install && npm run dev and cd frontend && npm install && npm run dev on the root of your repository.
* Frontend and Backend should work by just executing npm install followed by npm run dev (to run the application) or npm run test (to run all the tests). DON'T add extra instructions or docker commands on readme, if anything else needs to be executed before starting the application, make sure to include it in your dev script.
* When executing npm run test in your project, it must show the coverage report on the terminal after the tests execute.
* Javascript files are only allowed in lib configuration files, all your code MUST be in Typescript and completely typed.
* When you are finished, deploy your code in a hosting service like [https://render.com](https://render.com/) or [https://vercel.com](https://vercel.com/). You will be asked to provide the link of your repository and the link(s) of your deployed app at the final, make sure to provide the root link without any path.

How you should NOT do it: <https://server.render.com/api/users>

How you should do it: [https://server.render.com](https://server.render.com/)

* Not following any of the instructions above will cause immediate failure in the test.

The application should include the following features:

**Frontend**

* Should run on port 4000, and everything should be in the "/" route as a single-page application (SPA) using React.
* A button to select a CSV file from the local machine.
* A search bar that allows users to search for data within the loaded CSV file.
* The search bar should update the displayed cards to show only the matching results.
* The loaded CSV data should be displayed as cards on the website, with each card displaying all the data from a single row of the CSV file.
* A responsive design that works well on both desktop and mobile devices.
* Clear and user-friendly error handling.
* Check here a wireframe exemplifying how it should be: <https://timofi-assets.s3.amazonaws.com/test.png>

**Backend**

* Should run on port 3000.
* The backend should be implemented as a RESTful API using Node. (DON'T use any opinionated framework such as Adonis or Nest).
* The backend must include the following endpoints:
  + **[POST /api/files]**
    - An endpoint that accepts a CSV file upload from the frontend and stores the data in a database or a data structure. You should use the key "file" in the body request.
    - This route should return status 200 and an object with the key "message" with the value "The file was uploaded successfully."
    - Or this route should return status 500 and an object with the key "message" with an error message in the value.
  + **[GET /api/users]**
    - Should include an endpoint that allows the frontend to search through the loaded CSV data. This route should accept a **?q=** query parameter for search terms and should search through EVERY column of the CSV. The filter should search for partial matches and also be case insensitive.
    - This route should return status 200 and an object with the key "data" with an array of objects inside it.
    - Or this route should return status 500 and an object with the key "message" with an error message in the value.
* We'll run automated tests into your code and make sure the routes match 100% of what has been requested.
* The backend should include appropriate error handling for invalid requests or other errors.
* The usage of a database is not mandatory, but if you decide to use one, use SQLite and make sure everything will be set up once the dev script is executed.

**Fullstack**

* Implement both frontend and backend features as described above.
* Ensure that the frontend can communicate with the backend API to load the CSV data and search through it.

**CSV Data Example**

* That's the structure example of the CSV file we'll test on your software.
* The CSV uses commas to split the fields and not semicolons, make sure your code supports that.

name,city,country,favorite\_sport

John Doe,New York,USA,Basketball

Jane Smith,London,UK,Football

Mike Johnson,Paris,France,Tennis

Karen Lee,Tokyo,Japan,Swimming

Tom Brown,Sydney,Australia,Running

Emma Wilson,Berlin,Germany,Basketball

**Evaluation**

We will evaluate your solution based on the following criteria:

* Completion of all required features and functionality.
* Quality and organization of code.
* Quality and coverage of automated tests.
* User-friendliness and responsiveness of the frontend.
* Performance and efficiency of the backend.