Redlen Technologies | Full Stack Developer

SUBMISSION GUIDELINES

1. Please complete within 6 days of reciept
2. Questions are welcome—please ask if something is not clear!

THE BRIEF

Members of Redlen are faced with too many restaurant choices. Make a web service that helps us decide where to dine. Please make sure you provide us the necessary README file showcasing the steps to run the application.

Acceptance Criteria

The web service must support.

- A list of recommendations filterable by
  - Geographic location (ex city + country, postal code, etc.)
  - Distance
  - Currently open
- Un/Favoriting restaurants
- Un/Blacklisting restaurants
  - Once blacklisted, a restaurant is listed ONLY in the user's blacklist and won't show up anywhere else to the user
- A RESTful API with endpoints to achieve the above
- You are free to write in any language you are comfortable with (recommended C#/Django for backend, any JS for frontend, any database).

Note: We acknowledge your ideas. You are allowed to follow any methodology (or) architectural pattern. But explain to us in the documentation briefly regarding your approach. Any further enhancements/ changes made with your interpretation without completely deteriorating the above criteria are acceptable.

Deliverables

A hosted, git-based repository (Bitbucket, GitHub, etc.) containing the following:

- API documentation (choose whatever industry standard you prefer)
- A README with
  - Instructions for installing dependencies and running the web service
  - Specific instructions for each operation (you may choose whatever tool the web

service will be accessed). Ex. If you choose cURL, provide the *exact* copy+paste commands to run; if you choose Postman, provide the Postman Collection file.

To make the scope/time investment of this test reasonable:

- Assume members of Redlen are all honest, capable users: Ignore login, authentication, authorization. Each person can remember their own userId and assume they wouldn't be using another person's userId.
- Data persistence
  - DO NOT build a robust datastore—use any pattern that you are comfortable with and mock data of at least
    - 3 different "users"
    - 5-10 different restaurants
  - Do this as cheaply/simply as possible (not intending to test database expertise)

THINGS TO CONSIDER

It is not mandatory to implement everything mentioned below, but it will be considered while evaluating.

- Security
  - Input validation
    - Endpoints should handle invalid input and respond appropriately
  - Sanitization
    - Vulnerability to basic attack vectors, such as SQL-injection, result in an automatic test failure
- Semantics and standards
  - Naming, conventions, etc. should be sensible and align to some common Industry-standard

EVALUATION CRITERIA

This test will be evaluated on the following:

- Architectural design
- Code quality
- Documentation

- Adherence to industry standards
- Ease of use
- Performance/resource utilization (less is more)

BONUS POINTS

- Proper API responses for any wrong user input (Ex If I request for wrong/unavailable info from the database, the API should respond with proper error responses)

BEST OF LUCK!