**RED Technologies Coding Challenge**

Thanks for your interest in working for RED Technologies! As a rapidly growing and changing workplace we are excited to speak with and learn from developers who are as passionate about programming as we are. As you traverse the challenge, put your best foot forward and feel free to comment or ask questions where clarifications are needed. If an objective is within your reach yet can’t be implemented in the time given, feel free to walk us through how you would approach it conceptually.

The challenge is broken into 3 scoring tiers. The first of which are our MANDATORY objectives. These are the items we consider day-to-day development tasks which may present some challenge but ideally are achievable in the time given. The second is what we consider ABOVE AVERAGE, implying there will be no penalty for failure to achieve, but would help a candidate stand out against competition. The final tier EXCEPTIONAL is for candidates with a wide variety of experience who want to present a fully featured, hosted application.

The application is stubbed out using .NET Core 3.1 (C#) and React (Typescript). VSCode or Visual Studio 2019 are required to complete the challenge. Feel free to use any packages or utilities necessary to create the deliverables. We love Stack Overflow so make heavy use of research where needed, pasting links as comments to show context and design strategy is heavily encouraged.Good luck!

**BACKEND:**

PROJECT OVERVIEW:

* Controllers: Currently contains a single controller for the Order API
* DataModels: This is where any data layer models live, currently just Order
* Enums: A location for any application enums
* Interfaces: Stubbed out schemas for a DI implementation of a data reader
* Projections: These are POCO/DTO classes for the API to return data
* Program: This is where the in-memory data is initialized as a dictionary

DELIVERABLES:

* Upgrade the in-memory (hardcoded) data dictionary to any kind of external store. Files, normalized/non-normalized databases, caching etc are acceptable. Preference will be given to normalized database implementations (SQL scripts are an ok deliverable).
* Implement 3 new APIs:
  + A search API that searches on Customer and OrderType
  + A create API that creates a new Order given all the fields an Order object has
  + A delete API that removes a list of orders by id
* Create and implement at *least* a repository layer using Dependency Injection (DI). Feel free to use .NET Core’s native features or a 3rd party package if preferred.
* Implement unit tests using the platform of your choice for the CRUD operations that validate that the data (or service layer) is called with the expected values from the API

MANDATORY FEATURES:

* Fully functional set of endpoints to match the expected deliverables. Endpoints should return ‘Bad Request’ statuses if given incomplete/faulty data.
* Any kind of external data store for source data
* Basic DI Implementation for a data ingress class
* Basic unit tests as application guardrails (controller or service layer)

ABOVE AVERAGE FEATURES:

* A normalized data store for application entities. A SQL script is an acceptable deliverable. Special consideration is given for knowledge/implementation of Entity Framework. Code-first or Database-first is acceptable.
* Install Swagger on top of the project
* Demonstrated knowledge of Dependency Injection lifetime management. A transient or scoped service layer to handle business logic or repository passthrough is acceptable.
* Unit tests at all layers including Controllers and DI classes

EXCEPTIONAL FEATURES:

* All the above-average deliverables
* Host your Database and API in a location accessible remotely i.e. publicly
* Implement authentication (username/password, token, etc) for the API
* Utilize containerization and/or a deployment pipeline to host the app programmatically

**FRONTEND:**

RESOURCES:

* A web API is available for your use to list, create, search, and delete example orders for your project app
* The url is <https://red-candidate-web.azurewebsites.net/index.html> which will take you to a Swagger page describing the API
* The Swagger page is read-only (no authorization for API calls)
* To make requests use an HTTP header with the key ‘ApiKey’ and a value of ‘b7b77702-b4ec-4960-b3f7-7d40e44cf5f4’
* **NOTE:** If applying/attempting as a fullstack candidate, your frontend challenge is expected to use a working implementation of the backend challenge

DELIVERABLES:

* + Create a view of order entities in-memory on the page
  + Implement a feature where users can create new order entities
  + Implement simple dropdown filters to limit the order view to certain customers and order types
  + Implement a simple textbox search to lookup an order by id
  + Implement a delete feature by which a user can remove one to many orders at once

MANDATORY FEATURES:

* Fully functional web page/application which behaves as described above
* Usage of a component library (Material UI recommended)
* Reasonably responsive behavior for different screen widths

ABOVE AVERAGE FEATURES:

* Implement frontend testing using a framework of your choice
* Implement a command to build/run the application with a console command
* Implement Redux (or other state management) and use it to create a ‘save draft’ feature for order creation. Ok if drafts don’t survive page refreshes.

EXCEPTIONAL FEATURES:

* All the above-average deliverables
* Host the app in a publicly available location
* Implement containerization and/or pipelines to automate the app deployment
* Design a custom view/theme for the app including animations or other user-friendly features

For questions or clarifications feel free to email [jmounts@spotinc.com](mailto:jmounts@spotinc.com) or [ajohnson@spotinc.com](mailto:ajohnson@spotinc.com)