eComEngine

www.ecomengine.com

.NET Coding Challenge

Thank you for participating in our candidate screening process, and for agreeing to complete the assigned technical exercise.

You are being asked to create a simple contact entry system. You should create a REST API that will enable a client to perform CRUD operations on the contact collection. The exercise is expected to take 2-4 hours to complete, but there are no constraints on how much time you can devote to it. Be prepared to answer questions about your solution in your technical interview.

We understand that you may have a limited amount of time to work on the exercise, so allow us to offer some guidance:

- To the best of your ability, deliver a fully functional solution. It should compile and run without issue when cloned to the reviewer's machine, and it should meet all the functional requirements that are specified in these instructions.
- Use the exercise as a chance to showcase your skills. There are many opportunities to layer in some additional (and unspecified) good software development practices.
- Provide a short write-up about the context of the solution in the README file in the solution.
- If you must make concessions in your implementation due to time constraints, please document them as thoroughly as possible in the README.
- Please use code comments or the README to explain any unconventional decisions or shortcuts that are apparent in your implementation.

We encourage you to provide a technical diagram of your approach (via draw.io). If you choose to do so, please put the URL in the README.

System Requirements

1. Create a new REST API using .NET (standard framework or .NET Core) technologies with the following endpoints:

HTTP Method	Route	Description
GET	/contacts	List all contacts
POST	/contacts	Create a new contact
PUT	/contacts/{id}	Update a contact
GET	/contacts/{id}	Get a specific contact
DELETE	/contacts/{id}	Delete a contact
GET	/contacts/call-list	Get a call list. (See detailed requirements in item #4 below).

2. The new contact entry request when creating or updating a contact will be JSON, and have the following format:

Format	Example

{
"name": {
"first": string,
"middle": string,
"last": string
},
"address": {
"street": string,
"city": string,
"state": string,
"zip": string
},
"phone": [
{
"number": string,
"type": string ["home" | "work" | "mobile"]
}
],
"email": string
}
	{
	"name": {
	"first": "Joseph",
	"middle": "Harold",
	"last": "Lansky"
	},
	"address": {
	"street": "8351 Ransom Street",
	"city": "Richmond",
	"state": "Virginia",
	"zip": 23238
	},
	"phone": [
	{
	"number": "540-654-1753",
	"type": "home"
	}
	],
	"email": "joe.lansky@home.com"
	}

3. The call list is generated from all contacts that include a home phone. It is sorted first by the contact's last name, then by first name, and returned as an array of objects that each have the following JSON format:

Format	Example

{
"name": {
"first": string,
"middle": string,
"last": string
},
"phone": string
}
	{
	"name": {
	"first": "Joseph",
	"middle": "Harold",
	"last": "Lansky"
	},
	"phone": "540-654-1753"
	}

4. Provide a storage mechanism for storing the contact entries. If you use a database, you do not need to include the database, but you should include instructions to set up and create the database. In order to simplify the overall solution and minimize runtime dependencies, you are strongly encouraged to use an embedded database such as LiteDB or something similar. This makes it much quicker and easier for us to run your application locally.

5. Write unit tests to verify functionality where you deem it appropriate.

6. Upload all files and instructions for building the project (if necessary) to GitHub and provide a link to us for review.