**Student C#/.NET Backend Developer test 4.22**

Answer one of the following questions, please submit your code (either as a zip file or as a link to a GitHub repository) to: [jobs@ravendb.net](mailto:jobs@ravendb.net)

All code must be written in C#.

# **Query Engine**

Given a data source such as:

public class Data

{

    public List<User> Users;

    public List<Order> Orders;

    // etc

}

public class User

{

    public string Email;

    public string FullName;

    public int Age;

    // etc

}

We want to write an engine that would accept a SQL string and output the results of the query.

Queries are in the following format: from <Source> where <Expression> select <Field>

Where Source is one of the properties on the data source, the where expression is a potentially compound predicate that can include equality, range comparisons, etc.

You do not need to support queries that are more complex than the format above.

Sample queries:

* from Users   
  where FullName = "John Doe" AND Age > 30   
  select Email
* from Users   
  where Email = "selected.databases@ravendb.net"   
  select FullName, Email
* from Users   
  where Email = 'jobs@ravendb.net' or Email = 'jobs@hibernatingrhinos.com'   
  select FullName, Email
* from Users

where Email = 'jobs@ravendb.net' or Age >= 18 and Age <= 99

select FullName, Email

* from Users

where (FullName = 'foo' or FullName = 'bar') and Age < 99

select FullName, Email

You can output the results of the query to the console or return them as a list from the query engine.

# **Implement a caching server**

Build an application that would expose a network service to provide a caching system. The application would listen on port 10011 and will accept TCP connections. The application should be able to handle concurrent connections and perform work for multiple clients at the same time.

The application will accept the following text-based command from clients. You can assume UTF8 encoding and line ending using “\r\n”.

Available commands:

* set cache\_key size\_in\_bytes\r\n  
  [size\_in\_bytes value]
* get cache\_key\r\n

All commands are single line, with arguments separated by spaces. Line end is marked using “\r\n”.

The *set* command will accept two arguments, the cache key and the size in bytes of the value to put in the cache. It will read the value from the network and store the value in the cache to be retrieved by a later *get* command. There is not “\r\n” following the value, the next command starts immediately after the end of the value. The server will respond with: “OK\r\n” after a set command has been processed.

The *get* command has a single argument, the cache key and will return the following output:

OK size\_in\_bytes\r\n

[size\_in\_bytes value]

O if the key doesn’t exist, it will output: MISSING\r\n

An example interaction with the server, where green represent the client and blue represent the server. Line ending are using the \r\n separator.

$ telnet 127.0.0.1 10011

Trying 127.0.0.1...

Connected to 127.0.0.1.

Escape character is '^]'.

set email\_addr 16

jobs@ravendb.net

OK

get email\_addr

OK 16

jobs@ravendb.net

get home

MISSING

set home 18

where the heart is

OK

get home

OK 18

where the heart is

The server should hold a maximum of 128 MB of values and when a new value is set that will exceed the maximum memory used, it should evict entries from the cache. The key or keys to be removed is up to you.

# **Synchronize folder to RavenDB**

Write an application that will synchronize a folder of JSON documents to a RavenDB database. The application is invoked every 5 minutes using the following command:

sync\_folder\_to\_db /path/to/folder http://live-test.ravendb.net db\_name

The application accepts three arguments:

* The path to the folder to sync
* The URL of the RavenDB server
* The database name to use

You can assume that the command is always invoked with the same parameters. You need to check only the files in the provided folder, you can assume that there are no subfolders in the specified folder.

On each invocation of the application, you need to store to RavenDB all *new* files that does not exists already in the database, delete documents that represent files that were removed on the folder and update any document that was changed since it the last update.

The amount of work required per run should be proportional to the number of *changes* that happened to the files on the folder and not to the total number of files.

In other words, if the folder contains 10,000 files and only 5 of them changed since the last invocation, you cannot read 10,000 documents from RavenDB to compare them.

The files in the folder are modified by a 3rd party process over which you have no control. You can assume that files aren’t being modified while you are running the application, but you can make no assumptions about the state of the files between invocation of the application.