2022

Senior Backend Developer Assessment

SENIOR BACKEND END DEVELOPER ASSESSMENT
JOAQUIM RODRIGUES

KIRON INTERACTIVE | 26 Baker Street, Rosebank

Contents

Version Control

Version	Date	Author	Description
1.00	2023/05/10	Joaquim Rodrigues	Initial draft
1.10	2024/06/20	Joaquim Rodrigues	Update to coin stats API access
1.20	2025/06/27	Joaquim Rodrigues	Deprecated Coin stats and replaced with Dragon Ball API

Database Layer

The business requires the development of a reusable, generic database layer that can be leveraged across multiple projects.

This data layer should serve as a wrapper around the Dapper ORM. If you are not familiar with Dapper, you may use an alternative ORM of your choice, provided it supports the required functionality.

The implementation should include the following:

- A **DB connection manager** responsible for managing a connection pool. It must enforce a strict limit of **10 open connections** at any given time. If the number of active connections exceeds this limit, the system should throw an exception.
- The data layer must be limited to executing **stored procedures only**. Inline SQL queries are not permitted.
- It should support **full transaction management**, including the ability to create transactions, roll them back, and commit them. All data modification operations in subsequent tasks must make use of transactions.
- Additionally, the layer should be capable of **retrieving a single database record** and deserializing it into a corresponding object or model.

Caching Layer

A central caching layer is needed to store both database entities and API response data. This layer should be implemented in a **generic and reusable** manner, allowing it to be easily integrated across multiple projects.

You may use **MemoryCache** for storing the data, or any alternative caching mechanism you are comfortable with, provided it meets the performance and flexibility requirements. The caching layer should be designed with scalability and maintainability in mind.

.NET WEB API Project

This project will make use of the database provided in the assessment pack, named **"KironTest.bak."** After applying your changes to the database, please ensure that you **create a backup** and include it in your submission.

All SQL scripts used for object creation (such as tables, stored procedures, and other database objects) must be saved as **separate Transact-SQL files** and included in your submission package. The .bak file should be restored using **SQL Server version 14.0.10000.169 or a compatible version**.

In addition, the API you develop must be **thread-safe** when performing database queries or calling external APIs. As this API may be subject to high traffic volumes, it is important to **minimize unnecessary calls** to both the database and third-party services.

Make full use of the caching layer you've implemented to reduce redundant traffic. Demonstrating thread safety in both **synchronous and asynchronous** scenarios will be viewed favorably.

User Access

Within the **KironTest** database, create a table to store site users along with their passwords. All passwords must be **securely hashed** using a one-way hashing algorithm, ensuring that they cannot be reversed to their original form.

Develop an API endpoint that allows for **user registration**, accepting a username and password in the request. This endpoint should store the user's credentials in the newly created table, with the password stored in its hashed form.

Additionally, implement a **login endpoint** that accepts a username and password, verifies the credentials, and returns a **JWT token** upon successful authentication. This token should have a **one-hour expiration period**.

All subsequent API endpoints described in this project must be secured and should **only be accessible when a valid Bearer token** is included in the request headers.

Recursive data interpretation

Within the **KironTest** database, there is a table named **"Navigation"**, which stores the configuration used to render the front-end site's navigation structure in a hierarchical format.

Your task is to expose an API endpoint that **queries the data in this table** and returns it in a structured JSON format suitable for front-end consumption. The expected response format is as follows:

```
{
        “text”: “NavText”,
        “children”: [
                {
                        “text”: “Child1”
                }
                …
        ]
}
```

The **ParentID** field in the table indicates the parent of each navigation item. A ParentID value of -1 signifies that the item is a **top-level navigation item**. Your implementation should organize the data into the correct parent-child hierarchy based on this field.

UK Bank Holidays

Create an API endpoint that retrieves UK Bank Holiday data from the following external source: **https://www.gov.uk/bank-holidays.json**

Assume that the data provider requires **minimal traffic** to their service and that any requests made must be **thread-safe**.

Upon the first request to this endpoint, the retrieved data should be **stored in the database**. You are required to design appropriate tables to store the **regions** and their associated **bank holidays**. Since many holidays are shared across multiple regions, your database schema should support storing each holiday **once**, and associating it with all applicable regions via a relational structure.

Additionally, this endpoint should **trigger an automated process** that periodically re-queries the UK Bank Holidays API to **update or insert** data as needed. Once this automated process has been successfully initiated, **any further requests to this endpoint** should return an error message indicating that the task has already been completed and the data is now managed by the background process.

You are also required to:

- Expose an endpoint that returns a list of all available **regions**.
- Expose another endpoint that retrieves all **bank holidays** for a given region from the database.
- Ensure that all data retrieved from the database is **cached for 30 minutes** to reduce load and improve performance.
- Implement **thread safety** in all operations related to database access and external API calls.

Dragon Ball Characters

Please refer to the following website: https://web.dragonball-api.com/documentation.

Your task is to expose an endpoint that retrieves all Dragon Ball characters from this API. Assume that the provider of this API requires minimal traffic to their endpoint and that any interaction must be thread safe.

You should define the necessary data models to deserialize the response from the API correctly.

Implement a caching mechanism to store the retrieved data for a duration of one hour. If the data is accessed within that one-hour window, the cache expiration time should be extended by another hour. If the cache is not accessed within the extended period, it must be cleared automatically.