|  |
| --- |
| Kevin Jeremy |
| Xtramile .NET Practical Test |
| Task 1 |
| Assuming we have a large set of patients (500,000+) in a relational database, we want the user  to type part of a patient name and the system returns a list of matched patients.  • What data structure and search/matching algorithm to use and why?  • How fast the search is using this algorithm and data structure? |
| Answer:  **TL;DR:**  My suggestion is to Implement index with query like parameter on where, and paging.  Therefore the data can be retrieved faster and with paging we’re limiting the data until we got the smaller dataset.  **Detail:**  One of the solution/algorithms to search patient name is we can use trie (prefix tree) to find match from the prefix. However, I don’t suggest we doing the filtering at code level  Because whether we have 500k data of patient or only 50, if we filter it on the code level, it will be a bad implementation for this case. I suggest to use filtering on the database level by using where parameter on the query.  This is to prevent retrieving all data first and filter it on code level, that’s not a good practice because it will harmful to our server CPU & Memory utilization.  So, my proposed solution is to use index, using like query on database level, and implement paging.  **Walkthrough**   * Dataset: 500k rows * This dataset is used for * Table Data:  1. Implement Index With implementing index, there’s no different because the dataset is too small (Not enough column) 2. However if the query is only this.   It will be executed in 2.5sc 3. Next thing we can do is. By implementing limit.   It will be much faster, down into 0.003sc 4. Then we can implement paging with limit |
| Task 2 |
| Spec:   * Github Link: * API (Backend):   + Written on VS 2019.   + Target ASP.Net Core 3.1   + Using ASP.Net core Framework * Frontend   + Written in React App * Database/Access   + For Country and City data, I took the data from <https://simplemaps.com/data/world-cities> and compiled into local database (SQLite)   + For weather I’m using free <https://openweathermap.org/api> as instructed   + API Key: 1d109ba1f84f6ab8ddfc04452419a319 |
| **Project Structure** |
| 1. API API Project itself have multiple Structure and workflow    1. Workflow       1. In this solution, we’re using cache to simulate real case scenario so it will be more scalable and resilient.       2. Controller Class is responsible on what implementation is used       3. Startup class responsible for setting up routing, cors, port, cache, and config.    2. Folders       1. Controller the code purpose is set the implementation of the services       2. Model->Contract A class that define API Contract Response       3. Model->Implementation A Subfolders and classes that implement the interface based on the differences of the implementation.       4. Model->Interface A folder to hold the interfaces. 2. Frontend React App for frontend. The important folder is the src folder, and in this solution, the frontend only using single page file. 3. Publish Everything you need if you want to run this project’s binary (API & Frontend). However, you might need to install some dependencies if you don’t have it before. 4. Task Main solution of the C# Project. To edit the projects you have to open from this folder, then open Task.sln 5. UnitTest Project that hold unit tests for API with Mock, Contains LocationService and WeatherService Test.  You can run all test with typing Ctrl + R, A in visual studio.  or type Ctrl + E, T to open the test explorer. The test can be run without internet connection. 6. worldcities(.sdf/.csv/db) local database to serve cities and countries data |
| **How to run published binaries to your local machine.** |
| In this section, we’re going to run the published binaries on your local machine. I suggest using Windows to minimize OS behavior discrepancies.   1. Clone the repo. 2. Go to the publish Folder 3. The requirement will be different for API & Frontend   API/Backend   1. Open Terminal. Type dotnet --info 2. Make sure .net core runtime 3.1 is installed on your machine 3. If not. You can install here <https://dotnet.microsoft.com/en-us/download/dotnet/3.1> 4. Navigate to API Folder 5. Open appsettings.json 6. You can modify your openweather API Key if you wish 7. Change LOCATION\_DB\_SQLITE and LOCATION\_DB\_SQLCOMPACT to the correct place in your machine. 8. Open terminal on the API folder 9. Write dotnet API.dll 10. You will get Now listening on: <http://[::]:9000> 11. This 9000 Port is hardcoded on the project API->Program->Line 24 12. Done, to test it you can use postman to make a Get request at localhost:9000/api/location/countries   Frontend   1. First you have to install the NVM here <https://nodejs.org/en/> 2. Then open your terminal 3. Type npm install axios react-select 4. Type npm install -g create-react-app 5. If you changed the API Port from 9000. You must change it too at src/config.js 6. Open terminal at Frontend folder and type npm start 7. The default browser will open the frontend app. The default port is 3000 |
|  |