Documentation

Grid Beyond Assignment

Sinead NAUGHTON

Contents

[Program Start Up 2](#_Toc34985335)

[Connect to database 2](#_Toc34985336)

[Testing 2](#_Toc34985337)

[Run application 2](#_Toc34985338)

[Program Requirements 2](#_Toc34985339)

[Program Architecture 3](#_Toc34985340)

[API 3](#_Toc34985341)

[Controller 3](#_Toc34985342)

[Service and data layer 4](#_Toc34985343)

[Model 4](#_Toc34985344)

[Client 4](#_Toc34985345)

[Components 4](#_Toc34985346)

[Services: 4](#_Toc34985347)

[Models 4](#_Toc34985348)

[Database 4](#_Toc34985349)

[Recommended Improvements 4](#_Toc34985350)

# Program Start Up

Connect to database

The folder that contains information relating to a database connection is GridBeyondAssignmentSineadNaughton\Data

To point the application to your database, update the DatabaseConnectionString.txt file with your connection string.

A copy of the database is also included: GridBeyondAssignmentSineadNaughton.bak.

Testing

Unit test were developed to test the logic methods of the API. You can find these is the GridBeyondAssignmentSineadNaughton.Test project.

Run application

Debug the project

# Program Requirements

|  |  |
| --- | --- |
| Create price entry | A user should be able to create a PriceItem. A PriceItem has:   * price (decimal with precision two) * timestamp(date and time).   To do this the user enters three fields a price, a date, and a time. For user entry:   * Valid price input is a number * Valid date input is any date. * Valid time input is an hour and either zero or thirty minutes.   As per sample data given, there can only be one price entry per timestamp (as such a timestamp should be unique) |
| Display table of all price entries | All PriceItems items should be displayed to the user in a table with columns for price and timestamp. |
| Display graph of all price entries | All PriceItems should be charted on a graph with the timestamp as the X axis and the price as the Y axis. |
| Display minimum price | Display the price value for the PriceItem that has the minimum price. |
| Display maximum price | Display the price value for the PriceItem that has the maximum price. |
| Display average price | Display the average value of price for all PriceItems. |
| Display most expensive 60 minute period | Display the start timestamp and the end timestamp for the sixty minute period that collectively has the highest value for PriceItems entered than any other sixty minute period. For clarification if this period started at 12:00 it would include entries for 12:00, 12:30, and 13:00. The rile was made by following sample data given. |

# Program Architecture

The application is composed of a REST API built using .NET Core that connects to an on premise SQL Server Express database and a web client built with Angular. It uses MVC architecture. There is breakdown below of the API and Client, although it is currently a simple application with not much functionality, it has been built with this structure so it could be extended more easily.

## API

The API is composed of:

Controller

PriceController has three endpoints. Create a PriceItem, Get all PriceItems, and Get PriceItem calculations. It uses Web MVC to define endpoints. Below each of the endpoints are defined.

|  |  |
| --- | --- |
| URL | /priceitems |
| HTTP method | GET |
| Request body content | None |
| Response body content | List<PriceItem> |
| Function | Returns all PriceItems in the system |

|  |  |
| --- | --- |
| URL | /priceitems |
| HTTP method | POST |
| Request body content | PriceItem |
| Response body content | None |
| Function | Creates a new PriceItem in the system |

|  |  |
| --- | --- |
| URL | /priceitems/calculations |
| HTTP method | GET |
| Request body content | None |
| Response body content | PriceItemCalcuation |
| Function | Returns a PriceCalculation that contains the minimum, maximum, average price for all PriceItems and the most expensive hour. |

### Service and data layer

The data layer connects to a SQL Server DB, and the service layer provides create and read functionality for the database.

### Model

This contains the object models.

PriceItem

This item is instantiated to create PriceItems and is mapped to the database using Entity Framework. It has the fields Price, Timestamp, and Id. Id has been included so that if in the future it was not required that timestamp be unique there is a clear unique identifier

PriceCalculation

This item is instantiated to create PriceCalculations. It is used to send API response information about the min, max, average, and most expensive hour.

## Client

The Client in built with Angular. Its basic architecture is:

Components: home.component is the main view, it is responsible for displaying the PriceItem information to the user and accepting input for a new PriceItem, it uses angular data binding to do this. I It is styled with bootstrap 4. It also uses CanvasJs package for the graph

Services: PriceItemsService is responsible for making requests and receiving responses to the API.

Models:

## Database

There is a SQL Server database. This contains one table – PriceItems. This table has three columns price, timestamp, and id. As per database requirements Id is the key and the only unique filed. It is ensured that Timestamp is unique through application logic.

# Recommended Improvements

* Date Storage: dates are no converted to UTC before storage, this could create inconsistencies and issues if viewing in different time zones.
* Date Display: formatting on date display is at times inconsistent, this should be improved.
* Testing: Tests for expected behaviors in case of null data etc. should be added. It has been manually tested for thing like null data but these should be automated too.
* Error Message: there are currently no error message returned to the user. If a user tries to enter another price for a timestamp that is already used it will not add it but no message is returned to let them know why it appears the system isn’t doing anything. Another example is if the user has not selected a date.
* Http requests/response: should specify things like content type in header