Process MeNtOR 3.o

Uni-SEP

Easy Estates

Initial Design Requirements

Version:	1.5
Print Date:	
Release Date:	
Release State:	
Approval State:	
Approved by:	
Prepared by:	Hasti Ghaneshirazi, Wenzhuo Li, Jayant Varma and James Prime
Reviewed by:	
Path Name:	https://github.com/hastighsh/Easy-Estates
File Name:	
Document No:	

object oriented pty. ltd.

Document Change Control

Version	Date	Authors	Summary of Changes
1.0	04/02/2023	Hasti Ghaneshirazi, Jayant Varma	Filled section 1.1 and 1.2
1.1	19/02/2023	Jayant Varma	Added the UML class diagram highlighting the structure of the whole application
1.2	20/02/2023	Jayant Varma	Added sequence diagrams for use case 3, 4
1.3	22/02/2023	James Prime	Rearranged text within the introduction Subtitle added in front page with respect to deliverable requirements Fixed typos
1.4	23/02/2023	James Prime	Added in sequence Diagrams Added in reference to NHPI
1.5	24/ 02/2023	James Prime and Jayant Varma	Added test cases, Gantt chart, component diagram, uml class diagram refinement with description.

Document Sign-Off

Name (Position)	Signature	Date
Jayant Varma	Jayant Varma	19/02/2023
Jayant Varma	Jayant Varma	20/02/2023
James Prime	James Prime	22/02/2023
James Prime	James Prime	23/02/2023

Contents

1 Introduction

1.1 Purpose

This document details the requirements of the system Easy Estates.

An application that helps the user find the most affordable (in relative terms) region in Canada to buy a house. The application allows the users to choose different geographic regions of the same level (provinces or towns) and a specific period and compare the NHPIs over time using various visualizations.

1.2 Overview

Application requirements:

1- The user should be able to select data to load to the application.

2- The user should be able to switch between raw data provided through the database and the provided data table in the application with the descriptive statistics of the data.

3- The user should be able to toggle the visualization of the data they want to view from a certain number of provided filters.

4- A user should be able to compare data from two time-series to compare in a statistical test.

5- A user should be able to predict future data patterns for an X time period based on choice.

1.3 Resources - References:

GITHUB REPOSITORY: https://github.com/hastighsh/Easy-Estates

What is the New Housing Price Index (NHPI)?

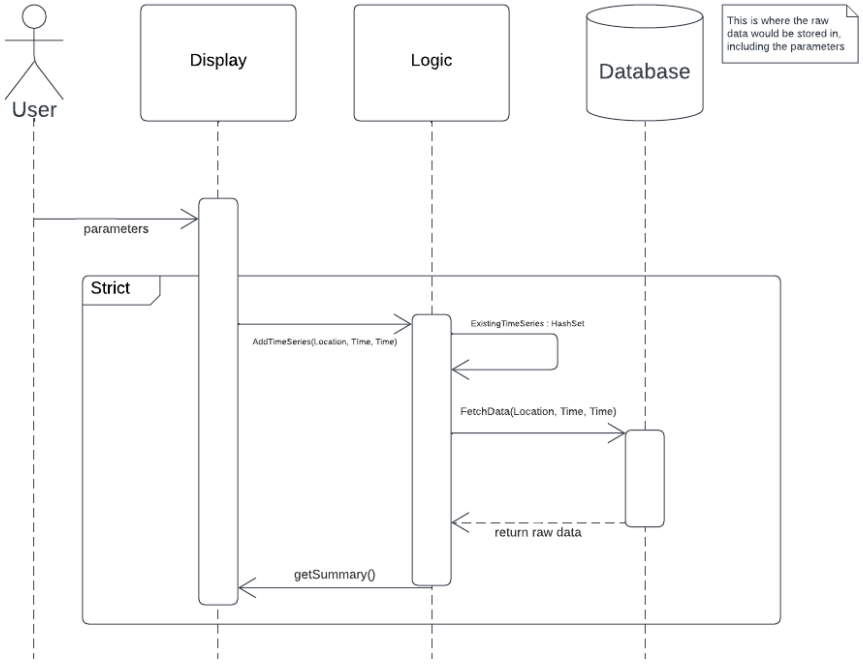
The New Housing Price Index (NHPI) is a monthly series that measures changes over time in the builders' selling prices of new residential houses.

The NHPI is used by economists, academics and the general public to monitor trends in the residential sector of the construction industry. Within Statistics Canada, components of these series are used in the calculation of some elements of the Consumer Price Index. The NHPI series are of particular interest to the real estate industry for comparison with the resale market.

https://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&SDDS=2310

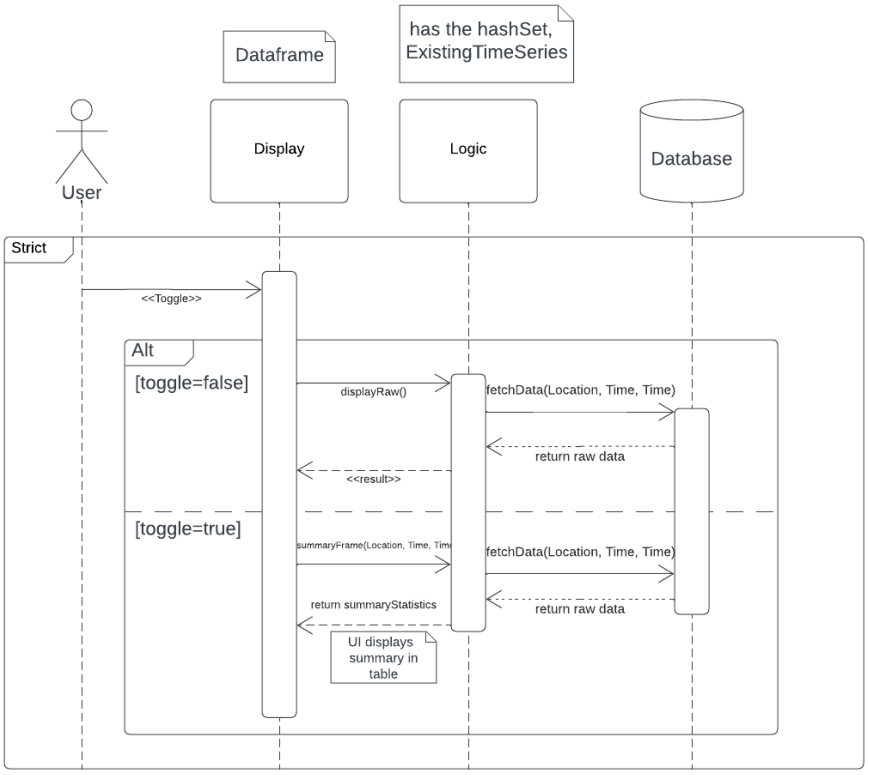
2 Sequence Diagrams

Use Case 1



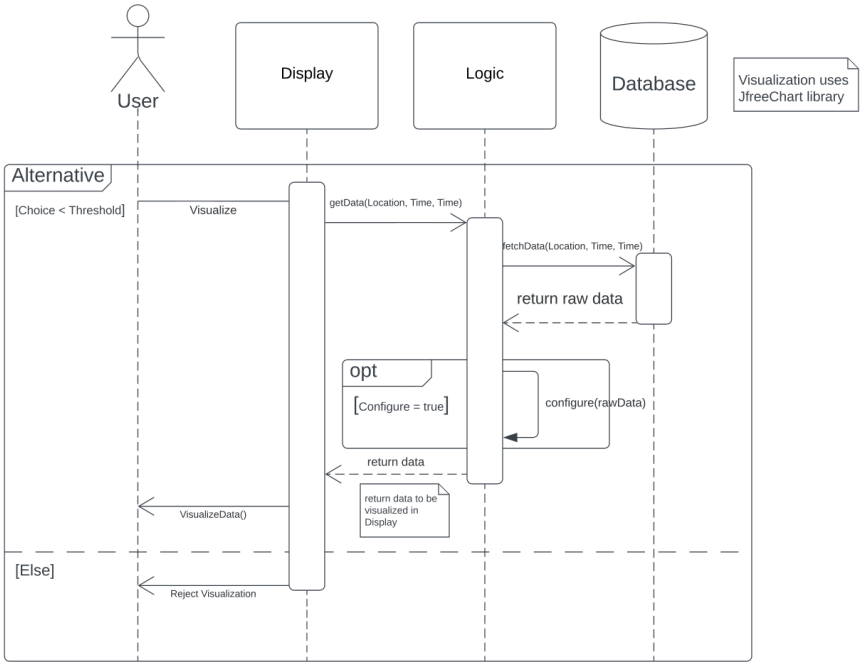
The user passes parameters through the display. Once the Load Data button has been pressed, the display passes the parameters to the Logic to be stored as a time series. The Logic fetches the data from the DATABASE. The data queried will be returned to the Logic, and finally calling the display to present the table of raw data.

Use Case 2



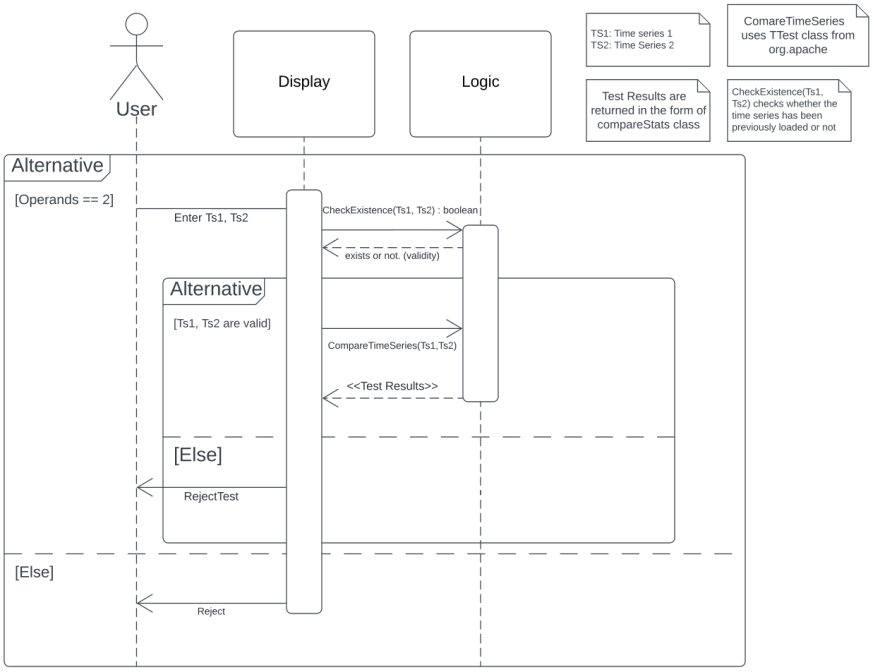
The user toggles the dataframe display. Depending on the value, the call to the Logic would differ. False, the Display would call for raw data from the logic to be displayed in a scrollable frame. True, the Display would call for the Logic, in turn would pass the data from the Logic to Table to create stats to be eventually displayed.

Use Case 3



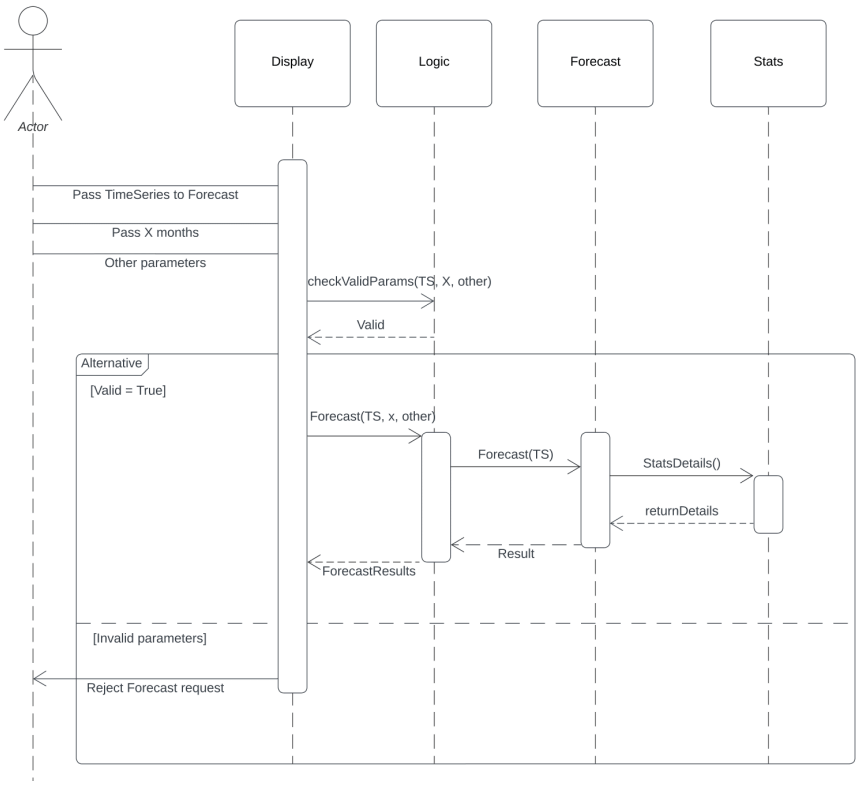
The user chooses the type of visualization in the Display, assuming. The Display gets the data from the Logic, in turn fetching the desired data from the Database. The Database returns the data to the Logic. If a certain visualization requires a configuration, the Configure Button must've been triggered, thus allowing the Logic to configure the data received previously. The Logic then returns the data to the Display to then visualize depending on the choice made.

Use case 4



To compare two time-series, the user must have already entered them through the Display and into the Logic. The Logic will check if they exist and valid, if not the Display will reject the user with an invalid test response. Otherwise, the Display will call logic to compare the two time-series, with the test results being returned afterwards to be displayed.

Use case 5



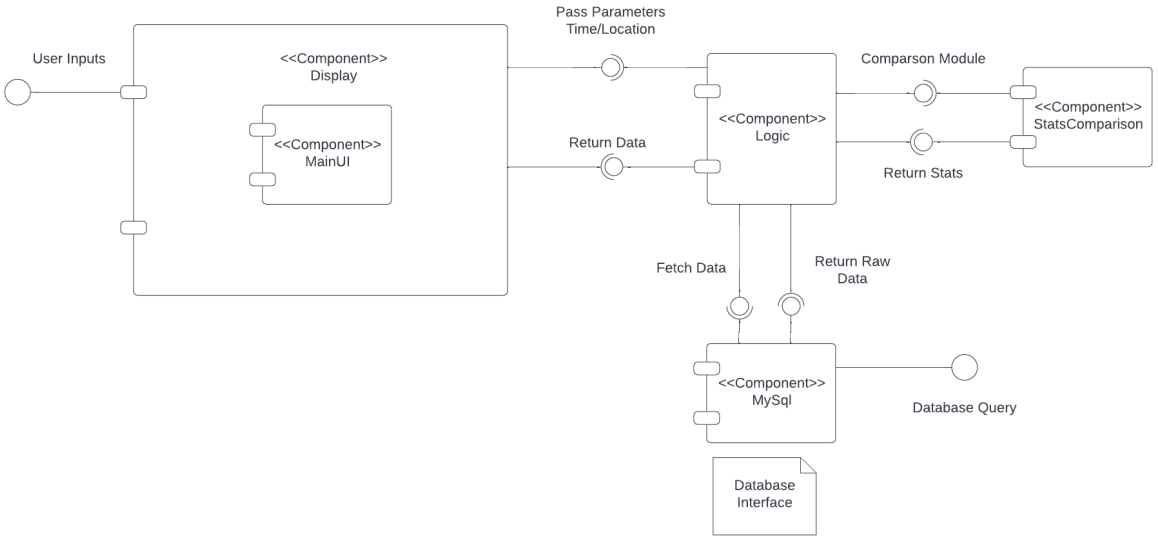
The user provides parameters to predict the next X months of a given time-series to the Display. The Logic checks if the parameters received are valid, if not reject the user and send an error message. Otherwise, The Display prompts the Logic to forecast the predictions given the parameters, Logic then calls upon Forecast to determine the results with statistical data from the Stats. Afterwards, with the forecast data, Logic returns the results to the Display for the user.

3 Major Design Decisions

Decision to refine the data in the database to a certain extent rather than to load it completely in the logic class and then compute. This makes a more efficient system, with best memory usage.

To modularize the application, and achieve high cohesion and low coupling, we decided on a MVC architecture. Separating the module responsibilities for the Model, View, and Controller.

4 Architecture



Modules			
Module Name	**Description**	**Exposed Interface Names**	**Interface Description**
Display	The UI the user interacts with, and where the requested data would be displayed	User Inputs Display: Pass Parameters	UserInputs:Display User inputs data parameters Display:Logic Display passes parameters into the Logic to create a time-series
Logic	The controller of the app that interacts with the view and model.	Logic:Comparison Module Logic:Fetch Data Logic:Return Data	Logic:Comparison Module Logic passes time-series to generate statistics Logic:Fetch Data Logic passes time-series data parameters to the database interface Logic:Return Raw Data Logic returns requested data back into the Display
MySql	Database interface that queries, receives, and returns the requested data.	MySql:Database Query MySql:Return Raw Data	MySql:Database Query MySql formats a query, and receives the requested data MySql:Return Raw Data MySql returns requested data back into the Logic
StatsComparison	Provides Statistical comparison between two time series from the Logic.	StatsComparison:Return Stats	StatsComparison:Logic Returns statistical data derived from comparing multiple time-series

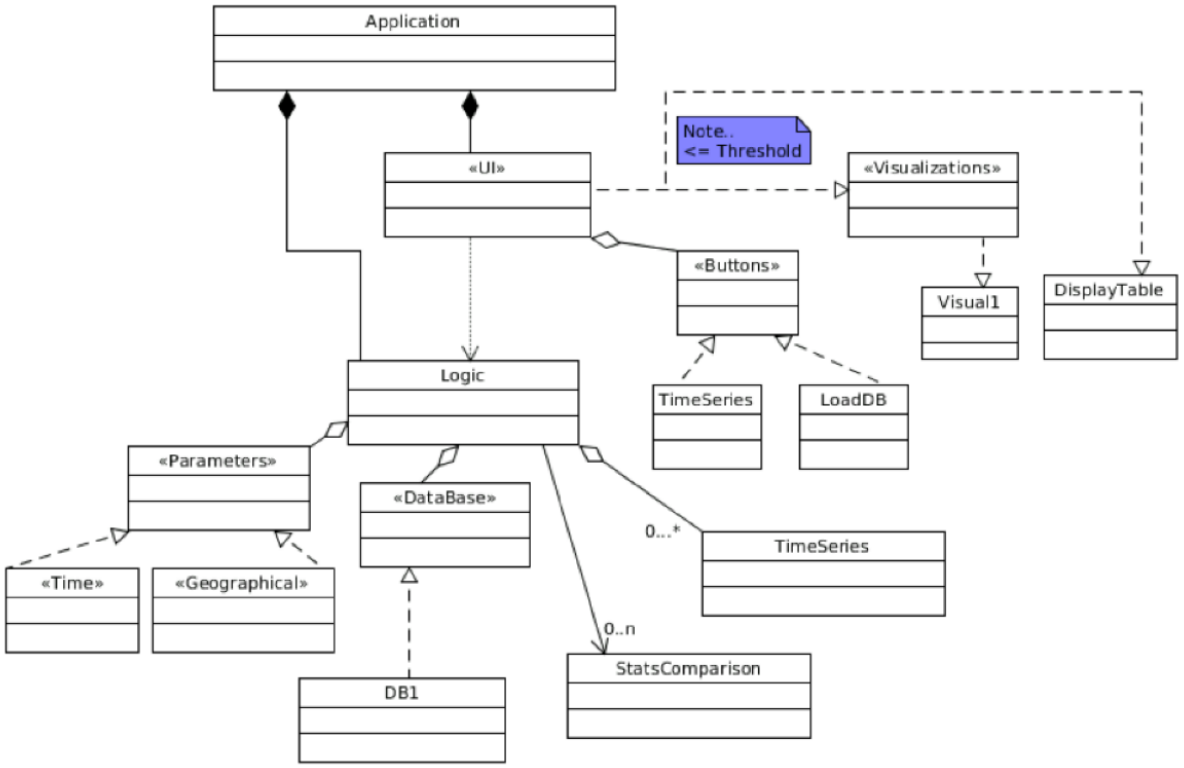
Interfaces		
Interface Name	**Operations**	**Operation Descriptions**
Display: Pass Parameters	<void>Display:Pass Parameters: loadData() used by Display	After the loadData button listens to an action, it passes the parameters to the Logic
Logic:Comparison Module	<StatsComparison> Logic:Comparison Module:compareTimeSeries(TimeSeries, TimeSeries) used by Logic	The Logic sends the two time-series to be compared. Returning a StatsComparison data structure
Logic:Fetch Data Logic:Return Data	<ArrayList<Double>>Logic:Fetch Data: fetchData(Location, Time, Time) used by Logic	Fetches data from the given parameters.
MySql:Database Query	<void>MySql:Database Query: exec() used by MySql	Executes query to the database
MySql:Return Raw Data	<ArrayList>MySql:Return Raw Data: getIndex() used by Logic	The logic, after sending the data to MySql, accesses that data from the MySql object.
StatsComparison:Return Stats	<double>StatsComparison:Return Stats: getPValue() and getConclusion() used by Logic	The stats data is accessed from the Logic when needed from the StatsComparison

5 Detailed Class Diagrams

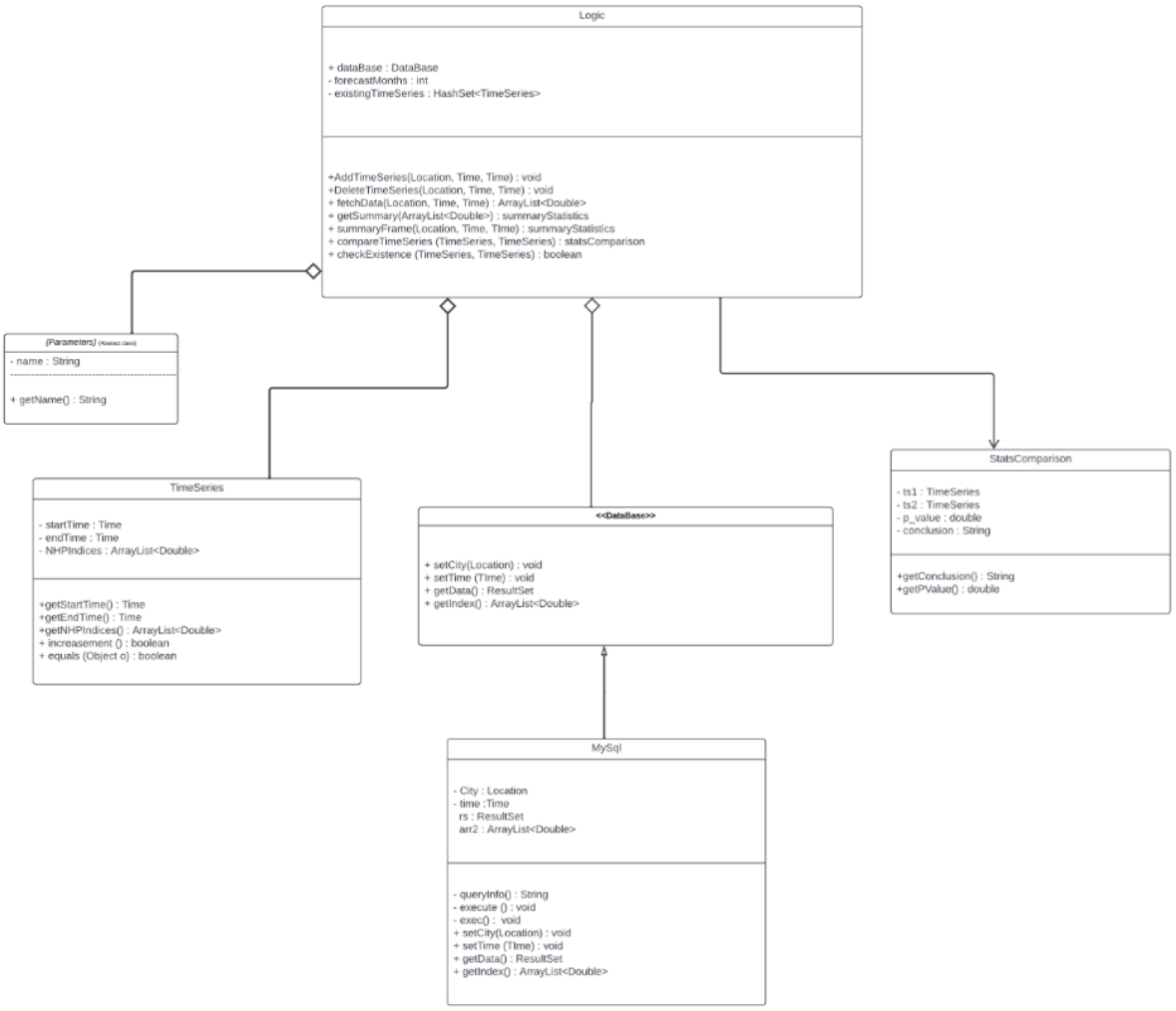
5.1 UML Class Diagrams

Detailed class diagrams for each class you modify or write for the extensions. You can separate the class diagrams per module they appear in. Tables should also be included listing the methods of each class with a short description of what each method does. Please indicate if a specific design pattern is used in your class diagrams.

An overall view of the whole application, focusing on the relationships between classes and interfaces, inheritance and aggregation/composition:



A more detailed uml class diagram describing the attributes, and methods of the Logic class, <<DataBase>>, parameters {Abstract class}, statsComparison, and TimeSeries class:



Class name	Attribute/Method name	Description
Logic	+dataBase : <<DataBase>>	Logic has an instance of DataBase interface that is then implemented by MySql
Logic	-foreCastMonths : int	The number of months the user wants to forecast for
Logic	-existingTimeSeries : HashSet	Keeps track of the TimeSeries that exist in the system already (i.e, the saved TimeSeries that have been added using the AddTimeSeries button)
Logic	+<Void>AddTimeSeries(Location place, Time startTime, Time endTime))	When the user wants to Add a TimeSeries, this method is called and it loads the TimeSeries in the system
Logic	+<Void>DeleteTimeSeries(Location place, Time startTime, Time endTime))	When the user wants to delete a TimeSerie, this method is called and it loads the TimeSeries in the system

Logic	+<ArrayList <Double >>FetchData(Location place, Time startTime, Time endTime))	To get the raw data from the data base, the logic class calls this method and gets an ArrayList of NHPI's for the selected location and between the start and end time.
Logic	+<summaryStatistics>getSummary(ArrayList<Double> RawData)	Uses the summaryStatistics class to give the summary of the NHPI indices logic got from fetchData().
Logic	+<summaryStatistics>summaryFrame(Location place, Time startTime, Time endTime)	Uses getSummary() to give a presentable summary in case the user clicks the toggle button.
Logic	+<StatsComparison>CompareTimeSeries(TimeSeries ts1, TimeSeries ts2)	Uses org.apache.commons.math3.stat.inference, specifically TTest() for now and stores all the results of the comparison in a class called StatsComparison. The results can be accessed using getters.
Logic	+<Boolean>CheckExistence(TimeSeries ts1, TimeSeries ts2)	Used as a helper method to check validity of ts1, ts2 for compareTimeSeries method.
Parameter {Abstract class}	-name : String	The name of the location or time
Parameter {Abstract class}	+<String>getName()	A getter for name of the parameter
TimeSeries	-startTime : Time	Signifies the start time chosen by the user
TimeSeries	-endTime : Time	Signifies the end time chosen by the user
TimeSeries	-NHPIndices : ArrayList<Double>	The raw data from the csv file containing the NHPI values from the startTime until the endTime
TimeSeries	+<Time>getStartTime()	Getter for startTime
TimeSeries	+<Time>getEndTime()	Getter for endTime
TimeSeries	+<ArrayList<Double>>getNHPIndices()	Getter for NHPIndices
TimeSeries	+<boolean>increasement()	Checking whether the start and end time are equal
TimeSeries	+<boolean>Equals(Object o)	Equals method for equality checking
MySql	-City : Location	A location
MySql	-time : Time	A time
MySql	rs : ResultSet	A return type data of sql (in the form of a table)
MySql	+<void> : exec	execute query
MySql	+<void> : execute	execute query
StatsComparison	-ts1 : TimeSeries	One of the operands for the comparison

StatsComparison	-ts2 : TimeSeries	One of the operands for the comparison
StatsComparison	-p_value : double	Resultant p-value of the T-Test
StatsComparison	-conclusion : String	Resulting conclusion of the comparison
StatsComparison	+<double>getPValue()	Getter for p-value
StatsComparison	+<String>getConclusion()	Getter for conclusion

6 Use of Design Patterns

Possible use of:

Factory design pattern to make instances of Time and Geographical parameters. Since only 1 each can be made for a time series, a factory design pattern with a restriction on the number of creations is natural to use.

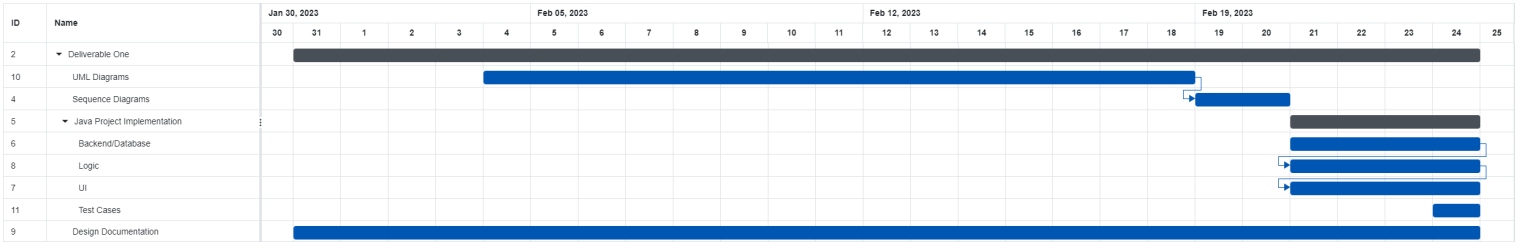
Chain of responsibility in use case 4, when the logic decides what comparison test to use based on the parameters entered.

7 Activities Plan

1.1 Project Backlog and Sprint Backlog

In this Section, and assuming you follow a Scrum process model, provide a list of product backlog items so that you can select items for your Sprint backlog. Make sure the product backlog list and the tasks in each product backlog item are consistent with the Gantt Chart in Section 6.1. above.

Gantt chart:



a. Deliverable One
    i. Design Patterns
    ii. Architecture
    iii. Major Use Cases
        1. Database access
        2. Data visualization
        3. User Interface
    iv. Test Cases

1.2 Group Meeting Logs

In this Section you write minutes of each meeting, listing the attendance, what the topics of discussion in the meeting were, any decisions that were made, and which team members were assigned which tasks. These minutes must be submitted with the project report in each deliverable and will provide input to be used for the overall assessment of the project.

Present Group Members	Meeting Date	Issues Discussed / Resolved
Hasti Ghaneshirazi, Wenzhuo Li, Jayant Varma, James Prime	04/02/2023	1) Reading of description, documentation (~35 minutes) 2) Setting up communication platforms and live development environments (~10 minutes) 3) Summarizing the use cases (~30 minutes) 4) Deciding a meeting plan (~10 minutes)
Hasti Ghaneshirazi, Wenzhuo Li, Jayant Varma, James Prime	08/02/2023	1) Set goal: Try to finish (do at least 80%) all diagrams by Sunday (12th Feb) night. 2)Work on use case 1 tomorrow
Hasti Ghaneshirazi, Wenzhuo Li, Jayant Varma, James Prime	09/02/2023	1) Appoint different principles for each use case. 2) Brainstorming on different architectures.
Hasti Ghaneshirazi, Wenzhuo Li, Jayant Varma, James Prime	10/02/2023	1) worked on the UML diagram of the main classes.
James Prime Jayant Varma	16/02/2023	1) went over UML diagram for case one 2) some precursor of the UI structure
Jayant Varma, James Prime, Hasti Ghaneshirazi	18/02/2023	1) Decide the final uml class diagram(~100 mins). 2) The components for the component diagram (~10 mins) 3) Architecture style as MVC and graphed a model (~40 mins)
Jayant Varma, James Prime, Hasti Ghaneshirazi,	19/ 02/ 2023	1) Distribution of teams and their tasks: (~0.5 hours) 1.1) GUI handling by Hasti and Wenzhuo 1.2) Database handling by Jayant and James

Wenzhuo Li		
Jayant Varma, James Prime, Hasti Ghaneshirazi, Wenzhuo Li	20/02/2023	1) Make a sequence diagram for use cases 1 to 5. (~6 hours)
Jayant Varma, James Prime, Hasti Ghaneshirazi, Wenzhuo Li	22/02/2023	1)Went over the current state of frontend and backend code
Jayant Varma, James Prime, Wenzhuo Li, Hasti Ghaneshirazi	23/02/2023	1) Current state of the development was discussed 2) Merging of code between Jayant and Li, mixing the Logic and Database 3) Documentation was minorly discussed

2 Test Driven Development

Test ID	Test# 01: FetchData
Category	Logic class and Database
Requirements Coverage	Fundamental test, no prerequisites.
Initial Condition	A database implementing <<database>>. The database must have a fully loaded Statistics Canada csv file containing the data for NHPI values from 1981 to 2022 across Canada.
Procedure	The list of steps required for this test case: *1. Use any start and end time between January 1981 and December 2022* *2. Use any city/town/province/Canada as location* *3. Pass these values into FetchData(Location, Time, Time) as parameters.*
Expected Outcome	The expected outcome of the test case: An ArrayList is returned containing the NHPI values between the start and end time passed for the selected location.
Notes	This fetchData() method is used in almost all tests and use cases.

Test ID	Test#02 : LoadData

Category	User-Interface, Logic class. Evaluation of selecting and loading data, taking on use case 1
Requirements Coverage	Fundamental test, depends only on Test# 01: FetchData is working
Initial Condition	The user is able to see the application running on their screen. The user is able to select a time and location from the drop down menu.
Procedure	The list of steps required for this test case: *1. The user selects the desired location from the drop down menu* *2. The user selects the desired start and end time from the drop down menu* *3. The user clicks on load data button*
Expected Outcome	The user is presented a frame containing the data formatted in a scrollable table showing the NHPI values for the selected location between the selected start and end time.
Notes	If output matches expectations, this proves the correct functioning of the load data function for use case 1.

Test ID	Test#03 : AddTimeSeries
Category	User-Interface and Logic class
Requirements Coverage	Test#01: FetchData, Test#02 : LoadData are working
Initial Condition	The user is able to see the application running on their screen. The user is able to select a time and location from the drop down menu.
Procedure	The list of steps required for this test case: *1. The user selects a desired location from the drop down menu* *2. The user selects the desired start and end time from the drop down menu* *3. The user clicks on load data button*
Expected Outcome	The logic class stores the added time series in a Hashset. When the data is loaded, it is automatically added to the Hashset, keeping track of all the created time series. Although nothing is outputted, one may verify the addition by trying Hashset.contains() method.
Notes	Used for use case 1

Test ID	Test#04 : ToggleButton

Category	User-Interface, Logic class, Database
Requirements Coverage	Test#01 : FetchData, Test#02 : LoadData are working
Initial Condition	The user is able to see the application running on their screen. The user is able to select a time, and location from the drop down menu and has clicked loadData button
Procedure	The list of steps required for this test case: *1. The user selects desired location from the drop down menu* *2. The user selects the desired start and end time from the drop down menu* *3. The user clicks on the load button* *4. The use clicks on the toggle button*
Expected Outcome	A new window opens showing a table containing the descriptive statistical summary (mean, variance, std. deviation, max, min) of the raw data that was present in the display frame before clicking on the toggle button.
Notes	Used for use case 2

Test ID	Test#05 : Visualization #01
Category	User-Interface, Logic class, DataBase
Requirements Coverage	Test#01: FetchData, Test#02 : LoadData are working
Initial Condition	The user is able to see the application running on their screen. The user is able to select a time, and location from the drop down menu and has clicked loadData button
Procedure	The list of steps required for this test case: *1. The user selects desired location from the drop down menu* *2. The user selects the desired start and end time from the drop down menu* *3. The user clicks on the load button* *4. The use clicks on the visualize button and chooses a visualization from the drop down menu*
Expected Outcome	The selected visualization is outputted in a new window.
Notes	Takes on use case 3

Test ID	Test#06 : Visualization #02
Category	User-Interface, Logic class, DataBase
Requirements Coverage	Test#01: FetchData, Test#02 : LoadData, Test#05 : Visualization #01 are working

Initial Condition	The user is able to see the application running on their screen. The user is able to select a time, and location from the drop down menu and has clicked loadData button
Procedure	The list of steps required for this test case: *1. The user selects desired location from the drop down menu* *2. The user selects the desired start and end time from the drop down menu* *3. The user clicks on the load button* *4. The use clicks on the visualize button and chooses a visualization from the drop down menu* *5. Repeat step 4 continuously*
Expected Outcome	Beyond a certain number of selections of visualization generations, an exception is thrown, because of which a message is generated notifying the user of the threshold for the number of allowed visualizations.
Notes	Takes on use case 3

Test ID	Test # 07: CompareTimeSeries#01
Category	User-Interface, Logic class and Database. Taking on use case 4.
Requirements Coverage	Test#01: FetchData, Test#02 : LoadData are working
Initial Condition	The user is able to load data into the system, thereby adding a time series to it.
Procedure	The list of steps required for this test case: *1. The user selects the location, start time, end time and loads the data, thereby adding the time series into the system.* *2. The user clicks on compare time series and selects the two operand time series.*
Expected Outcome	The CompareTimeSeries method returns the result of the comparison as an instance of CompareStats class and the p-value, conclusion of the test are outputted on the display.
Notes	Takes on use case 4

Test ID	Test#08 : CompareTimeSeries#02
Category	User-Interface, Logic class and Database. Taking on use case 4.
Requirements Coverage	Test#01: FetchData, Test#02 : LoadData, Test # 07: CompareTimeSeries#01 are working

Initial Condition	The database has been loaded with the Stats-Canada csv file data. The user is able to load data into the system.
Procedure	The list of steps required for this test case: *1. The user selects the location, start time, end time and loads the data.* *2. The user adds this time series into the system.* *3. The user selects the two operand time series and then clicks on compare time series button* *4. The user selects at least one time series that has **not** been loaded into the system*
Expected Outcome	The CompareTimeSeries method throws the exception, 'MissingTimeSeries Exception' and a corresponding message is generated for display.
Notes	Takes on use case 4

Test ID	Test#09 : Forecast#01
Category	User-Interface, Logic class and Database. Taking on use case 5.
Requirements Coverage	Test#01: FetchData, Test#02 : LoadData, Test#04 : ToggleButton,Test#05 : Visualization #01 are working
Initial Condition	The user is able to see the application running on their screen. The user is able to select time, location, and the app is able to load data, get a summary of data, fetch raw data and visualize.
Procedure	The list of steps required for this test case: *1. The user selects their desired location and time series that has been loaded after loading the location, start, and end time.* *2. The user then clicks on the forecast button.* *3. In the new pop-up window that appears, the user then selects the algorithm for forecasting from the drop down menu and the number of months to forecast to.* *4. The user then chooses a means of visualization of the data being forecasted.*
Expected Outcome	A visualization of the expected time series for the entered number of months of the selected location is shown along with statistical summary in a frame (similar to the one seen in Test#04 : ToggleButton)
Notes	Takes on use case 5

Test ID	Test#10 : Forecast#02

Category	User-Interface, Logic class and Database. Taking on use case 5.
Requirements Coverage	Test#01: FetchData, Test#02 : LoadData, Test#04 : ToggleButton,Test#05 : Visualization #01, Test#06 : Visualization #02, Test#09 : Forecast#01 are working
Initial Condition	The user is able to see the application running on their screen. The user is able to select time, location, and the app is able to load data, get a summary of data, fetch raw data and visualize.
Procedure	The list of steps required for this test case: *1. The user selects their desired location and time series that has been loaded after loading the location, start, and end time.* *2. The user then clicks on the forecast button.* *3. In the new pop-up window that appears, the user then selects the algorithm for forecasting from the drop down menu and the number of months to forecast to.* *4. The user then chooses a means of visualization of the data being forecasted.* *5. The user selects all visualizations to represent the forecasting*
Expected Outcome	An exception is thrown notifying the user that the chosen visualization cannot do the required forecast. (Since not all visualizations can forecast a certain data). Another possible exception is the threshold on the number of visualizations. The upper limit of the number of visualizations will at least display the exception of choosing all the possible visualizations.
Notes	Takes on use case 5