London Covid-19 Statistics

Team: Binary Baes

- ➢ Egor Kuzmichev - k23066266
- ➢ Emil Khojayev - k23092237
- ➢ Bernas Guterres **-** k

Main App Window (program entry point):
The main stage serves as the application's navigational hub, featuring controls such as 'Back', 'Forward', 'Instructions', and 'Quit' buttons. These facilitate easy navigation through the application's panels and access to usage guidelines. Additionally, date pickers are provided for users to specify the date range of interest, which applies across the various data visualizations.

Welcome Panel (Panel 1):
- ➢ Function: Greets users with a welcoming message and displays an introductory image.

Map Panel (Panel 2):
- ➢ Function: Visualizes COVID-19 death toll data across London boroughs using a hexagonal map representation.
- ➢ Features: Each borough is represented as a clickable hexagon, color-coded to reflect death tolls within the selected date range. Hover effects and a detailed statistics window for each borough enhance interactivity and information accessibility.

Statistics Panel (Panel 3):
- ➢ Function: Allows users to explore four distinct overall COVID-19 statistics for the selected date range.
- ➢ Features: Interactive buttons enable users to switch between statistics, fostering an engaging data exploration experience.

Borough Data Comparison Panel (Challenge task):
- ➢ Function: Facilitates comparative analysis of COVID-19 statistics between two boroughs.
- ➢ Features: Users can select boroughs and a specific statistic for comparison, with the application dynamically displaying appropriate cumulative line graphs or bar charts. Default selections ensure immediate data presentation upon access.

Technical Implementation:

The application leverages Scene Builder for panel layout creation, supplemented by manual FXML adjustments and a CSS stylesheet.

DataSorter class:

The DataSorter class is designed for managing and analyzing COVID-19 data specific to London's boroughs. This class serves as a intermediate between the controller classes and the covid data retrieved from

Data Filtering Methods:
- ➢ filterDataByDateRange(LocalDate fromDate, LocalDate toDate): Filters the loaded data to include only the records within the specified date range.
- ➢ getDataByBorough(LocalDate fromDate, LocalDate toDate, String boroughName): Further filters the data by borough name within the given date range.

Statistical Analysis Methods:

- getTotalDeathsByDateRange(LocalDate fromDate, LocalDate toDate): Calculates the total number of new deaths due to COVID-19 within the specified date range.
- getAverageCases(LocalDate fromDate, LocalDate toDate): Determines the average number of new cases per day within the date range.
- Methods for calculating average Google Mobility Reports (GMR) metrics (getAverageRetailGMR, getAverageGroceryGMR, etc.): These methods calculate average mobility trends for various sectors like retail, grocery, parks, etc., within the specified date range. Borough-specific versions of these methods (getBoroughGroceryGMR, getBoroughRetailGMR, etc.) are also provided.

Interval-Based Cumulative Data Methods:

- getCumulativeDeathByIntervals(LocalDate fromDate, LocalDate toDate, String boroughName) and getCumulativeCasesByIntervals(LocalDate fromDate, LocalDate toDate, String boroughName): These methods generate maps linking dates to cumulative death and case counts, respectively, divided into intervals for a more granular analysis over time.
- It splits the date range into intervals so that graphs don't become cluttered with too many date values.

Map Panel:

Utilises MapPanel.fxml and MapPanelController class

This controller manages a series of buttons representing different boroughs and a table view for presenting detailed statistics of a selected borough. Below is a detailed breakdown of its functionalities and structure:

Methods:

- Shape and Color Handling:
  - applyHexagonShape(Button button): Applies a hexagonal shape to the specified button.
  - hexagonShape(double radius): Generates a hexagonal Shape object given a radius.
  - updateButtonColors(): Dynamically updates the color of each borough button based on COVID-19 death data, creating a visual representation of the impact intensity.
- Utility Methods:
  - interpolateColor(Color start, Color end, double fraction): Calculates an interpolated color between two specified colors based on a given fraction, used for creating a gradient effect.
  - toHexString(Color color): Converts a Color object to a hexadecimal color string representation.
- Data Handling and Display:

  setButtonClickHandlers(): Assigns action handlers to borough buttons, triggering data display in a table upon selection.
  - setDataSorter(DataSorter dataSorter): Allows injection of a DataSorter instance for data processing.
  - setDateRange(LocalDate fromDate, LocalDate toDate): Sets the date range for data retrieval.
  - displayDataInTable(ArrayList<CovidData> filteredBoroughData): Populates the table view with COVID-19 statistics for a selected borough. This involves setting up cell value factories for each column to map data from CovidData objects to table cells.

Design and Implementation:

- The use of TableView and TableColumn components for data display emphasizes the application's data-driven nature, providing users with detailed and organized information.
- This controller integrates various aspects of UI management, event handling, and data visualization to provide an aesthetic design.

Statistics Panel:

Utilises Stat Panel.fxml, StatPanelController class, and DataSorter

This controller allows users to view different statistics, such as total deaths, average cases, and Google Mobility Report (GMR) metrics, by interacting with navigation buttons.

Methods:

- Event Handlers (handleNextAction, handlePreviousAction): These methods adjust currentStatisticIndex to cycle through the available statistics in response to user interactions, invoking updateStatisticsDisplay to refresh the displayed data accordingly.
- updateStatisticsDisplay: this method updates the statName and statValue labels based on the currently selected statistic and the date range. It switches based on currentStatisticIndex to determine which statistic to query via dataSorter and displays its value.
- Setters (setDataSorter, setDateRange): These methods allow external entities (such as the main application controller) to provide the StatPanelController with necessary context (i.e., the DataSorter instance and the selected date range).

Design and Implementation:

- It showcases dynamic UI updates through the use of event listeners attached to buttons, allowing for interactive exploration of different statistics without needing to reload the view or data explicitly.
- By delegating data querying and calculation tasks to the DataSorter, the controller maintains a focus on UI logic, keeping data handling and business logic separately encapsulated.

Borough Data comparison Panel (Challenge):

Utilises CompPanel.fxml, CompPanelController class, and DataSorter

This panel is specifically focused on comparing COVID-19 statistics between two selected London boroughs over a specified date range.

Methods:

- setupComboBoxes(): Adds listeners to the ComboBoxes to trigger data updates when selections change.
- updateChartData(): Main logic for updating chart data based on the current selections. Clears old data, determines which chart to display based on the selected statistic, and invokes specific methods to display the new data.
- Display Methods (displayCumulativeDeaths, displayGroceryGMR, etc.): Each of these methods is responsible for querying the DataSorter for the relevant data and updating the chart series accordingly. The choice of line chart or bar chart and the data format depend on the selected statistic.

Design and Implementation:

- The class is heavily reliant on JavaFX's FXML for UI component declaration and event handling, demonstrating a typical Model-View-Controller (MVC) pattern where CompPanelController acts as the controller managing interactions between the UI (view) and the data model (CovidDataLoader and DataSorter).
- Dynamic UI updates are handled through property listeners attached to the ComboBoxes, ensuring the charts reflect the current user selections.
- There is differentiation between line and bar charts based on the type of statistic selected.

Test class:

Utilises DataSorterClass

The Test class is designed to test the DataSorter class, as this is the method with the most complex data filtering methods. The test class was made to test all methods with a specific date range in mind, to ensure it works correctly The time range to be tested is from the 4th of March 2021 to 16th of March 2022 .

Methods:

- setUp():Needed method to start a test
- tearDown(): Needed method to end a test after
- testFilterDataByDateRange(): Tests the more basic method that removes any data outside the date range. THis method is extremely important, as it is used in essentially all controller classes, to display most data To test, three methods are called, assertNotNull and assertFalse, to ensure the array is created correctly, and assertEquals, getting a random value inside the filtered data to ensure the data is stored correctly as well.
- testFilterDataByDateRange():Tests the more advanced method that not only removes any data outside the date range, but also removes data taht isnt from the specific borough. Three methods were called again, assertNotNull and assertFalse, to ensure array is created correctly, and assertEquals, which tests if it returns a random gmr data value for the specific borough “Croydon”.
- testGetTotalDeathsByDateRange():Tests if sum method of adding all total deaths withing a date range works. Uses only assertEquals()
- testGetAverageCases():Same as method above only for average cases instead
- testGetAverageRetailGMR():Same as method above, only for average retail GMR instead
- testGetAverageGroceryGMR():Same as method above, only for average grocery GMR instead
- testGetCumulativeDeathByIntervals():Tests the method used in the challenge panel to get the death values, uses assertNotNull and assertFalse methods to ensure its created correctly, and an assertEquals method looking at the final value store in map, to ensure it returns the correct number of deaths for Sutton within a specific time range.
- testGetCumulativeCasesByIntervals():Same implementation as the previous method, checking if it is returning the correct number of cases for Sutton within a specific time range.

Task Allocation:

There was a general task allocation in our group, but everyone contributed bits and pieces to each task
Egor Kuzmichev: Challenge Tasks, Panel 3, Report, FXML Files
Emil Khojayev: Application Window, Panel 1, FXML files
Bernas Guterres: Panel 2, Unit Test, Report, CSS and Aesthetics