Project Report

**EffiSync**

Kareena Tuli (E22CSEU0269)

Sadiya Faizan (E22CSEU0260)

A report submitted in part fulfilment of the degree of

**BTech in Computer Science**

**Supervisor:** Dr. Priyanka



Department of Computer Science

Bennett University

November 16, 2023

# Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 8462

Student Names: Kareena Tuli (E22CSEU0269), Sadiya Faizan (E22CSEU0260)

Date of Submission: November 16, 2023

Signature:

# Abstract

EffiSync is an all-inclusive productivity application that we have developed in C++ using the Qt Creator framework, integrating data structure concepts and object-oriented programming principles. This project aims to provide users with a multifaceted platform to enhance their daily productivity, task organization and time management.

#### 1. Features:

**Reminder Management:**

EffiSync enables users to add and edit reminders easily using a user-friendly interface for managing their daily tasks.

**Calendar Events:**

Users can view and manage calendar events, providing a convenient view of their scheduled activities and commitments.

**To-Do List:**

Our project helps in the creation and management of to-do lists tasks, thus enabling better task organization and completion.

**Music Player:**

EffiSync includes an in-built music player that allows users to listen to their favorite tunes without leaving the productivity environment.

**Motivational Quotes:**

Users can access a collection of motivational quotes, that helps in creating a positive mindset and encouraging productivity.

**Quick Notes:**

EffiSync provides a quick and efficient way to jot down notes, ensuring users can capture ideas and thoughts on the go.

**Date, Time, and Weather:**

The application displays real-time information, including date, time, and weather updates, creating an all-rounded experience for users.

#### 2. Technologies Used:

**Programming Language:** C++

**Development Framework:** Qt Creator

**Concepts Employed:** Object-Oriented Programming (OOP), Data Structure Concepts

#### 3. Key Aspects:

**Data Structure Implementation:**

EffiSync uses advanced data structures for efficient organization of tasks, reminders, and notes.

**Object-Oriented Design:**

The project follows object-oriented design principles, promoting maintainability and code reusability in the code.

**User-Centric Interface:**

The application prioritizes a user-friendly interface, ensuring a smooth and seamless experience for users across all the functionalities.

EffiSync emerges as a powerful tool that integrates task management, entertainment, and information retrieval into a single platform. The strategic use of data structures and object-oriented design principles contributes to the project's powerful architecture, enabling users to enhance their productivity and time management in an easy and engaging manner.

#### GitHub Project Link:

##### https://github.com/SadiyaFaizan12/CPP\_EffiSync

# Table of Contents

|  |  |
| --- | --- |
| **Topic** | Page Number |
| 1. **Introduction**    1. **Technological Framework**       1. **Programming Language and Framework**       2. **Object-Oriented Design**       3. **Data Structures**    2. **Features of Effisync**       1. **Reminders and Calendar Events**       2. **To-Do List**       3. **In-Built Music Player and Motivational Quotes generator**       4. **Quick Notes**       5. **Date, Time and Weather**    3. **Academic Significance** | 7 |
| **2. Problem Definition & Objectives** | 9 |
| 1. **Proposed Work/Methodology**     1. **User Interface**       1. **Date Time Widget**       2. **Weather Widget**       3. **Random Quote Generator**       4. **Music Player**       5. **Quick Notes**       6. **To-Do List**       7. **Calendar Widget & Reminders** | 11 |
| **4. Data Structure Used**  **4.1. Why do we need Data Structures?**  **4.2. Data structure used in Quote generator.**  **4.3. Data structure used in Music player**  **4.4. Data structure used in Quick Notes**  **4.5. Data structure used in To-Do List**  **4.6. Data structure used in Reminders** | 17 |
| **5. Language and Tools**  **5.1. Programming language used: C++**  **5.1.1. Reason**  **5.1.2. Outcome**  **5.2. Choice of Framework: Qt Creator**  **5.2.1. Reason**  **5.2.2. Outcome**  **5.3. Architecture and design principles**  **5.3.1. Object-Oriented Design**  **5.3.2. Advanced data structures**  **5.4. Conclusion** | 19 |
| **6. Source Code**  **6.1 header files**  **6.1.1 MAINWINDOW.H**  **6.2 Source Files**  **6.2.1 mainwindow.cpp**  **6.2.2 main.cpp** | 21 |
| **7. Results**  **7.1 Functioning of Date, Time and weather widget**  **7.2 Functioning of Quote Generator**  **7.3 Functioning of Music Player**  **7.4 Functioning of the Quick Notes**  **7.5 Functioning of To-Do List**  **7.6 Functioning of the calendar widget and reminders** | 37 |
| **8. Conclusion** | 46 |
| **9. Bibliography** | 47 |

# Chapter 1: Introduction

As a student in this rapidly evolving world of digital technologies, the need for an all-inclusive productivity application has become more pronounced. As individuals handle complex schedules and several diverse tasks, the need for an integrated and efficient tool for task management, entertainment, and personal enrichment has become the need of the hour. Recognizing this need, the project "EffiSync" has been created as a response to the challenges faced by users in their pursuit for excellence. EffiSync is visualized and developed to be a holistic productivity application that rises above prevalent task management tools by simplifying task management greatly for the benefit of the user. The project aims to provide users with a unified platform that combines a range of features, including reminders, calendar, to-do lists, music playback, motivational quotes, quick notes, and real-time weather updates all in one desktop application. This all-rounded approach is designed to address the varied needs of users, especially students, providing a comprehensive environment for both personal and professional task management. The decision to work on the development of a productivity application “EffiSync” as part of our project, is a result of a thorough evaluation of existing productivity applications and their compatibility, or lack thereof, with the daily needs of students. Commonly observed shortcomings in terms of student experience and lack of integration across different features of pre-existing productivity applications serve as the driving force behind the creation of our project EffiSync. By employing advanced programming concepts, such as data structures, and object-oriented principles, the project tries to bridge existing gaps and offer a state-of-the-art solution to the ever-changing demands of users, especially students.

## Technological framework

### Programming Language and Framework

EffiSync has been created using the C++ programming language as per the requirements of our project development. The Qt Creator framework is employed as the primary development tool, providing an easy-to-use environment for the creation of a graphical user interface (GUI) that is both intuitive and visually appealing.

### Object-Oriented Design

EffiSync follows the principles of object-oriented design. The integration of multiple features is made possible by this design paradigm, which promotes modularity, reusability, and scalability. Each component of EffiSync is treated as an object, encapsulating both data and the methods that operate on that data. This modular approach not only enhances maintainability but also makes it easier to add new features in the future.

### Data Structures

Our project allows the easy management of diverse types of data through the integration of advanced data structures. Maps, dynamic and static arrays are strategically employed to improve the organization and retrieval of information. These data structures provide to the efficiency and the flexibility of the app by handling tasks such as to-do lists, calendar events, and reminders.

## Features of EffiSync

EffiSync consists of a wide range of features to enhance the productivity and time management of users. These features have been developed keeping in mind primarily the needs of students.

### Reminders and Calendar Events

Users can add reminders and events on the calendar as well as delete or edit them whenever necessary. The calendar provides an easy view of the daily reminders and one can even search for an event to find, simply using its name.

### To-Do List

EffiSync comes with a dynamic To-Do List that allows users to easily add and remove tasks as and when needed, as well as cross off the ones that have already been completed.

### In-Built Music Player and Motivational Quotes generator

Entertainment and task management go hand in hand to maximize user productivity and achievement. Our application provides an In-Built music player to allow users to listen to tunes that they prefer whether they want to increase their concentration while working or take a relaxing break from their tasks. For the times when users need a little bit of encouragement to keep going, EffiSync delivers a set of dynamic quotes for that extra boost and motivation.

### Quick Notes

EffiSync allows quick notetaking by providing users with a digital notepad for quickly jotting down ideas, thoughts, and important information.

### Date, Time and Weather

Real time information about date, time and weather implemented using an API adds a practical dimension to EffiSync, enhancing the users’ awareness of their environment and allowing them to plan out their day accordingly.

## Academic Significance

Our project EffiSync holds great academic significance by demonstrating the practical application of advanced programming concepts of C++, object-oriented design principles, and data structure implementation in the development of a real-world productivity application. This project helped in combining theoretical knowledge with real-world problem-solving techniques, resulting in a thorough comprehension of our course.

EffiSync, as a productivity application, not only addresses the present-day challenges faced by users, especially students, in the field of time management but also serves as a testament to the capabilities and practical application of object-oriented programming with C++ and the usage of data structures. This project introduction lays the base for an in-depth analysis of EffiSync's features, programming algorithms and potential future enhancements.

# Chapter 2: Problem Definition & Objectives

EffiSync - Where productivity meets inspiration.

It might be difficult to remain organised and focused in the chaos of modern life. There are lots of distractions, motivation is low, and the expectations of productivity are frequently too much to handle. As a comprehensive productivity app, EffiSync charges into this turmoil with the goal of easing these difficulties by combining necessary tools into a single, fluid interface.

EffiSync is a revolutionary programme that integrates many capabilities to provide a harmonic blend of motivational resources and productivity aids. Its goal is to revolutionize the user experience. The principal aim is to augment efficiency, arrangement, and inspiration by providing an array of functionalities:

Music Player: The goal of EffiSync's music player is to provide tasks with more vigor and concentration. It has been scientifically demonstrated that music improves focus and productivity. To increase productivity, users can create custom playlists by choosing songs that correspond with their pace of work. They can create different playlists in their device and upload the folders as per their choice.

Random Motivational Quote Generator: To keep things moving forward, you need a steady stream of inspiration. The inspirational sayings provided by EffiSync's quote generator inspire optimism and tenacity while acting as subtly encouraging words that help people move closer to their objectives quotes change after every 10 seconds making the ui interesting.

Note-Taking Widget: Users may quickly and easily scribble down ideas, to-do lists, or significant passages using this user-friendly note-taking widget. This feature facilitates effective idea capture and helps with mobile thought organization. Users can quickly capture their ideas as they come, ensuring that no valuable thoughts are lost. Whether it's an innovative business concept, a creative inspiration, or simply a fleeting thought, this widget provides a convenient platform to preserve ideas for future reference.

Calendar with Reminder Function: Critical for individuals from all walks of life. From busy professionals to students juggling multiple responsibilities, effective time management is a key determinant of success. This function gives users the ability to efficiently manage their time. Reminders can be added, updated, and deleted, ensuring that no assignment or appointment is overlooked. Better time management and work prioritization are ensured by doing this.

To-Do List: With the help of EffiSync, users may now create lists of chores and mark them off as completed. This simple-to-use but effective application improves job management and gives you a clear picture of the work that still has to be done.

Date, Time, and Weather Widget: By providing real-time data, EffiSync boosts efficiency even further. Using the OpenWeatherMap API, the widget displays the current time, date, and weather. With the weather and time limits taken into account, this aggregated data helps users organize their activities more effectively.

Why EffiSync?

The complexity of productivity is acknowledged by EffiSync. People won't have to waste time switching between tabs or applications, nor will they have to switch between them frequently.

With its all-encompassing approach, EffiSync delivers a cohesive solution that addresses the core impediments to productivity. By combining motivation, time management, and organization into a streamlined process, EffiSync empowers individuals and teams to achieve their goals efficiently and effectively. Whether you are a student, professional, or business owner, EffiSync offers the tools and techniques you need to optimize productivity and unlock your full potential.

This environment is designed to maximize productivity. It includes a music player, inspirational sayings, note-taking features, a calendar with reminders, a to-do list, and real-time information on the weather and date. A concentrated work atmosphere can be experienced by users, who can also effectively manage their calendars and get inspiration from motivating cues.

EffiSync is more than simply an app; it's a transformative catalyst and a partner in the quest of effectiveness and achievement. It wants to be the go-to choice for everyone looking for a comprehensive strategy to increase productivity by skillfully combining technology, organization, and motivation. EffiSync functions as a beacon, directing people towards their objectives with clarity and purpose in a world full of dis

# Chapter 3: Proposed Work/Methodology

The development of EffiSync, a comprehensive productivity app, involves a methodical approach encompassing various features to ensure a seamless user experience and functional efficiency.

## 3.1 User Interface

A screenshot of a computer

Description automatically generated

The user interface is an amalgamation of all the features into one screen so that the user does not have to take the pains of switching tabs while organizing and planning is day.

### 3.1.1 Date Time Widget

The QDateTime class of the Qt Library has been used in creating this widget. A QTimer object is instantiated to update the displayed time every second (timer->start (1000)). The updatetime() method is a user defined method that retrieves the current date and time using the currentDateTime() method of the QDateTime class . It then displays the data returned information on the label.

### 3.1.2 Weather Widget

The OpenWeatherMap API has been used to access the current weather of a given place. The QNetworkAccessManager allows us to handle the network requests and connects its finished signal to onWeatherDataReceived () slot that is user defined.

The QJSonDocument class is used to get, parse and work with the json files received via the API and helps to display the retrieved information on the labels.

### 3.1.3 Random Quote Generator

This uses the QStringList class to store the quotes in a list which is then iterated over to display the next quote in the interval of 10 seconds which is managed by the timer object of the QTimer class. A user defined function called updateLabel () is used to change the quote on the label after every 10 seconds.

#### Flow Of Execution

1. Upon starting the QTimer, it counts down 10 seconds and emits the timeout signal.
2. The updateLabel () function is triggered by the timeout signal, updating the QLabel (ui->label\_5) with the next quote from the textArray.
3. The updateLabel () function is triggered by the timeout signal, updating the QLabel (ui->label\_5) with the next quote from the textArray.
4. This process repeats, cycling through the quotes at a 10-second interval as dictated by the QTimer.

### 3.1.4 Music Player

The QMediaPlayer class has been used to manage the functioning of the music player. QAudioOutput helps to generate the sound of the music and helps it project on to the speakers of the user system. QDir and QDialog classes allow us to access the source of the music from the user system and display the relevant information on to the labels .

The progress bar and time labels get updated through the durationchanged (), positionchanged() and updateduration() functions.

The MPlayer object provides specific methods to play pause and stop the current song. The playCurrentSong () method allows to play the song present in the beginning of the list of the songs created when the directory is fetched. playPreviousSong () and playNextSong () call the playCurrentSong() method after updating the index according to the requirement.

The mute button is managed by the audioOutput object of the QAudioOutput class.

#### Flow Of Execution

1. The user selects the folder where the music is stored using the ‘Open File’ option in the tool bar above. The selected song's folder path is obtained and passed to the loadsong () method.
2. The loadsong () method creates a QDir object based on the provided path and applies filters to retrieve specific file types (e.g., \*.mp3) using setNameFilters ().The absolute file paths are constructed from the directory path and file names, then added to the playlist (a QList<QString>) using playlist.append(filePath).
3. If the playlist is not empty, then the player starts by setting the current index to 0 to play the first song. playCurrentSong () method retrieves the path of the current song from the playlist, sets it as the source for the QMediaPlayer using SetSource(QUrl::fromLocalFile(currentSong)), and displays the song details on the UI (e.g., file name using QFileInfo).
4. The user interacts with playback control buttons in the UI (Play, Pause, Stop, Next, Previous, etc.) and corresponding functions (e.g., on\_pushButton\_Play\_clicked (), on\_pushButton\_Stop\_clicked (), on\_pushButton\_nextSong\_clicked ()) are triggered.

### 3.1.5 Quick Notes

A QVector<QString> named quickNotes is a dynamic array which is used to store and manage the notes. The functionality provided by the various button clicks orchestrates note management within a QPlainTextEdit field. Clicking "Add” appends a newly entered note to the quickNotes array . "Clear" erases the content in the QPlainTextEdit. Navigational buttons "Previous" and "Next" facilitate movement through the quickNotes array, displaying the preceding or succeeding notes in the QPlainTextEdit.

#### Flow Of Execution

1. User inputs a note in the QPlainTextEdit and clicks the "Add" button (on\_pushButton\_9\_clicked ()).The entered note is appended to the quickNotes array.
2. User clicks the "Clear" button (on\_pushButton\_10\_clicked ()) to clear the text in the QPlainTextEdit.
3. Clicking the "Previous" button (on\_pushButton\_11\_clicked ()) fetches and displays the preceding note in the array relative to the current displayed note.
4. Clicking the "Next" button (on\_pushButton\_12\_clicked ()) fetches and displays the subsequent note in the array relative to the current displayed note.

### 3.1.6 To-Do List

The To-Do list is managed by a QVector<QString> called ToDo to manage tasks. The two main functions, triggered by button clicks, facilitate task management within a QVBoxLayout containing QCheckBox widgets. The "Add Task" button adds a new task to the ToDo list based on user input entered in the QLineEdit. This function creates a new QCheckBox with the task description, adds it to the vertical layout, and clears the input field for subsequent entries. The "Delete Task" button removes a task from the To-Do list. It searches for a QCheckBox matching the entered task description within the layout and removes it if found, subsequently clearing the input field.

#### Flow Of Execution

1. Users input a task in the QLineEdit and click "Add" (on\_pushButton\_clicked()).The entered task is appended to the ToDo list as a QString and a new QCheckBox with the task description is added to the QVBoxLayout.
2. Clicking "Delete" (on\_pushButton\_8\_clicked()) retrieves the task description to be deleted. The function iterates through QCheckBox widgets and checks if a checkbox matches the entered task description. If yes , then it is removed from the layout and deleted.
3. Input fields are cleared after each action, ensuring a clean space for new entries or deletions without redundant information.

### 3.1.7 Calendar Widget & Reminders

**Add Reminder**

A QHash<QDate, QVector<QVector<QString>>> named myHash is used to store reminders associated with specific dates. Two key functions handle reminder addition and display based on user actions. The "Add Reminders" button processes user inputs, such as title, description, time, and email, and creates a formatted reminder string. This string is stored within nested vectors based on the selected date in myHash. Subsequently, it retrieves and displays reminders for a selected date on the calendar via the on\_calendarWidget\_clicked() method.

#### Flow Of Execution

1. Users input reminder details (title, description, time, email) in designated UI elements and click "Add Reminders" (on\_push\_Button\_clicked()).The entered details are formatted into a string and stored within a nested vector associated with the selected date in myHash.
2. Clicking a date on the calendar triggers on\_calendarWidget\_clicked() to retrieve reminders for the selected date from myHash. If reminders exist for the selected date, they are formatted into a string and displayed in the UI (ui->remin->setText(dataString)).
3. Reminders are stored within nested vectors in myHash, allowing for the organization of multiple reminders per date.

**Search Reminder**

This method, activated by the "Search Reminder" button conducts a search operation within myHash to find and display reminders matching a specified title. The method iterates through all dates in the hash using an iterator and examines each reminder associated with those dates. For every match found , it constructs a formatted string (reminder) with details like date, title, description, time, and email. If any matching reminders are discovered, they are displayed in the QTextEdit widget (ui->remin). Otherwise, if no matches are found, the interface shows a message indicating "No reminders found".

#### Flow Of Execution

1. Users enter a title to search for and click "Search Reminder" on\_pushButton\_6\_clicked()).The function iterates through all dates in myHash using an iterator (it), extracting the associated reminders for each date.
2. For each reminder associated with a date, it compares the entered title with the title of the reminder (title\_1=inner[0]).If a match is found, it constructs a formatted string (reminder) containing details (date, title, description, time, email) of the matching reminder.
3. If any matching reminders are found (count > 0), the formatted reminder details are displayed in the QTextEdit widget (ui->remin).
4. If no matches are found (count == 0), the interface shows a message indicating "No reminders found", providing feedback to the user regarding the search result.

**Delete Reminder**

Using a user-entered title, the "Delete Reminder" method removes the reminders. It searches through all dates and the reminders that go along with them to modify myHash. It navigates the hash structure and inner reminder vectors using iterators. A reminder is eliminated from the nested vector of reminders connected to a given date if the title entered matches the title of that reminder. Furthermore, the entire date entry for a particular date is eliminated from the hash if all reminders for that date are deleted. Following the deletion process, the interface notifies the user if no matching reminders were identified, or it gives feedback on how many reminders were erased.

#### Flow Of Execution

1. To delete a reminder, users must type its title and click the "Delete Reminder" button (on\_pushButton\_2\_clicked()).The function uses an iterator (it\_del) to loop through each date in myHash, enabling hash alteration.
2. It accesses and verifies the titles of all related reminders for every date.
3. The reminder is extracted from the nested vector in the event that a match is made.
4. The complete date entry is eliminated from myHash if all reminders for that date are removed.
5. After the deletion procedure, the function provides feedback on the number of erased reminders (count).The interface changes to indicate how many reminders have been erased or alerts the user if no matching reminders could be located.

**Edit Reminder**

A screenshot of a computer

Description automatically generated

The "Edit Reminder" method uses the title entered to determine which selected reminder should open in the edit window. To display an editing window ,this function alters the interface. It takes the title that was entered and looks through myHash to locate the reminder linked to that title. After finding the corresponding reminder, it sets editing available by loading the title, description, time, and email of the detected reminder in the edit window . The method will stop looking and display the data for modification if it is able to find a single matching reminder.

#### Flow Of Execution

1. To open the edit window, users click the "Edit Reminder" button (on\_pushButton\_3\_clicked()).The weather widget is moved, and the edit window (ui->groupBox\_2) is displayed as part of the interface modification. Using this->resize(), it modifies the window's overall size to make room for the edit window.
2. After retrieving the entered title, it starts iteratively (iterating over myHash) to locate the reminder linked to that title. It retrieves the related reminders for every date and verifies that the entered title matches each reminder's title.
3. It finds a matched reminder and adds the reminder's details for editing to the UI elements of the edit window.
4. The search is halted, and the reminder details are displayed in the edit box, enabling users to change the reminder if only one matching reminder is found (count == 1).

**Update Reminder**

#### Reminders are updated by the update button in response to modifications made in the edit box. First, it purges the reminders that are currently visible and deletes the reminder that is linked to the given title from the myHash hash table. It goes through every date and the reminders that go with it, eliminating the reminder if the title matches the input. Additionally, it makes sure that a date entry in the hash table is eliminated if all reminders for that date are destroyed. After that, it generates a new entry for the update and obtains the data from the edit window.

#### Flow Of Execution

1. Eliminates the current reminder linked to the given title from the hash table myHash. Deletes the corresponding reminder after iterating over all dates and associated reminders. If every reminder for that date was removed, then the date entry is removed.
2. It clears the reminders that are currently displayed once again and gets the most recent information about the reminder from the UI components (ui->lineEdit, ui->lineEdit\_3, ui->timeEdit\_3, ui->lineEdit\_4) of the edit window.
3. It creates a new entry for this revised reminder, containing the information in a nested vector called dateArray. The chosen date in the myHash hash table is connected to this nested vector.
4. After that, it formats and retrieves the reminders for the chosen date to update the reminders that are visible on the interface (ui->label\_11->setText(dataString)).

**Cancel Reminder / Close Reminder**

The Cancel and Close button work similarly . However, if the user clicks on close tab the edit text

box is also cleared.

# Chapter 4: Data Structure Used

### 4.1 Why do we need Data Structures?

Data Structures in C++ play a crucial role in developing applications to solve real world problems. They provide a way to efficiently organize and store data. Choosing the right data structure is crucial for the functioning of a program as it has a drastic impact on operations such as insertion, deletion, traversal, and retrieval of data. Data structures not only influence the way data is accessed in the program, but they also have an impact on effective memory management.

Keeping these points in mind, EffiSync uses appropriate data structures to efficiently store different types of data to make the processing and retrieval of data an effective and convenient process. This section deal with the different components of our project in terms of the data structures used by them and the reason why they are used.

### 4.2 Data structure used in Quote generator.

The motivational quote generator displays quotes that are dynamically changing after a fixed interval of time. In order to store the quotes in the memory and retrieve the quotes individually after the fixed clock interval, we have stored the quotes in a String List. The Syntax for the same in the Qt Creator IDE is const QStringList textArray = {}. The const keyword states that the list is static and once the quotes have been stored by the developer, they cannot be modified during runtime. This allows for easy continuous storage of strings as well as successive and iterative retrieval of the strings during runtime.

### 4.3 Data structure used in Music player.

The In-Built music player accesses the music files from the local repositories on the system where the application is run. The files of the correct format (i.e., *.mp3)* are extracted from the folder and are stored in a String List. This allows for the serial storage of the music files in the list as well as the chronological access/retrieval of the files during runtime.

### 4.4 Data structure used in Quick Notes.

The quick notes serves as a notepad that allows users to jot down their thoughts as and when they come up with one, In order to allow the proper functioning of the “undo” and “redo” buttons of the quick notes component, these notes are stored in a dynamic string array, i.e., a string vector. The syntax for the same is QVector<QString> quickNotes. This allows us to chronologically store the notes as well as append and delete notes from the array easily.

### 4.5 Data structure used in To-Do List.

The To-Do List component follows the same backend code as the quick notes component, i.e., the To-Do List also uses a dynamic string array, i.e., a string vector. The syntax for the same is QVector<QString> ToDo. This allows us to easily store the tasks as well as append and delete tasks from the array easily and in chronological order.

### 4.6 Data structure used in Reminders.

The reminders that are added in the calendar are stores in a hash map. The hash map is a dynamic data structure that allows easy storage and retrieval of the events/reminders. The syntax for the same is QHash<QDate, QVector<QVector<QString>>> myHash. In this Hash map, the key is the date selected on the calendar on the interface and the events and reminders corresponding to that date are stored in the form of a nested dynamic array i.e., a nested vector. The inner vector stores strings as it’s elements (these strings are the details of the reminder or event) while the outer vector (each outer vector contains all the reminders for a particular date) stores the inner vector (each inner vector corresponds to a single reminder) as its element. Since hash map is a dynamic data structure, the reminders can be easily stored, edited and deleted from the map. A hash map is also unsorted thus allowing easy date-wise access to the daily reminders.

It may be observed that in the syntax of declaring a data structure in the Qt Creator IDE, the “Q” prefix has been used everywhere. This is because Qt follows a consistent naming convention, and classes related to the different data structures usually start with a "Q" prefix. This convention makes it clear to the compiler that these classes are part of the Qt framework and distinguishes them from standard C++ libraries or other external libraries.

## Chapter 5: Language and Tools

Our project, “EffiSync”, a sophisticated productivity application, has been developed using the C++ programming language and the Qt Creator framework. This section provides insights into the language and tools chosen for the development of EffiSync, highlighting the reason behind these choices and their impact on the project's development and functionality.

### 5.1 Programming Language used: C++

#### 5.1.1 Reason

Our project for the subject CSET202 has been developed entirely using C++. C++ was selected as the programming language for EffiSync due to its versatility, performance, and efficiency. C++ allows for both high-level data handling and low-level memory manipulation thus making it suitable for building complex applications while ensuring optimal memory utilization and organization. Additionally, the object-oriented nature of C++ aligns with the design principles used in EffiSync, allowing easy code organization and maintainability.

#### 5.1.2. Outcome

The use of C++ in EffiSync facilitated the implementation of advanced data structures along with the smooth integration with the Qt framework.

### 5.2 Choice of Framework: Qt Creator

#### 5.2.1. Reason

Qt Creator was chosen as the primary integrated development environment (IDE) for EffiSync. This was done due to its extensive set of tools, cross-platform compatibility, and strong support for C++. Qt Creator’s set of integrated tools included the Qt Designer which helped in the development of the graphical user interface (GUI) design and allowed a sleek development process. The Qt framework's rich and easy to use library of components facilitated the creation of a feature-rich and visually appealing user interface for our project.

#### 5.2.2. Outcome

The usage of Qt Creator significantly accelerated the development process of EffiSync. The easy to use IDE, combined with Qt's Features, enabled seamless communication between different components of the application. The Qt framework's signal and slot mechanism simplified event handling drastically and contributed to the responsiveness and smooth functioning of EffiSync's user interface.

### 5.3 Architecture and Design Principles

#### 5.3.1 Object-Oriented Design

EffiSync's program architecture adheres to object-oriented design principles, using C++'s support for classes and encapsulation. Object-oriented design promotes modularity, and code reusability. EffiSync's components, such as task management, calendar events, and note-taking functionalities, are encapsulated within classes, creating a well designed and organized version of an otherwise complex system.

#### 5.3.2 Advanced Data Structures

EffiSync utilizes advanced data structures such as static arrays, dynamic arrays and maps to efficiently manage and organize diverse types of data. These data structures optimize the performance and responsiveness of EffiSync, especially in scenarios which involve dynamic task management, scheduled events, and efficient storage of reminders and notes.

### 5.4 Conclusion

The choice of C++ as the programming language and Qt Creator as the development environment for EffiSync has proven crucial in the creation of this feature-rich productivity application which caters to the needs of all their users, especially students. The combination of C++'s flexibility and efficiency with Qt Creator's integrated development tools has facilitated the smooth integration of complex functionalities and the creation of an intuitive and easy-to-use user interface. The usage of object-oriented design principles and the strategic use of advanced data structures have further contributed to the success of EffiSync as a state-of-the-art productivity application and time management solution.

## Chapter 6: Source Code

### 6.1 Header Files

#### 6.1.1 MAINWINDOW.H

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include<QtMultimedia/QtMultimedia>

#include <QtCore>

#include <QtWidgets>

#include <QMediaPlayer>

#include<QAudioOutput>

#include<QDialog>

#include <QNetworkAccessManager>

#include <QNetworkReply>

#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonValue>

#include<QJsonArray>

#include<QTimer>

#include <QLabel>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_push\_Button\_clicked();

void on\_calendarWidget\_clicked(const QDate &date);

void on\_pushButton\_6\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_5\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_7\_clicked();

void on\_pushButton\_8\_clicked();

void updateLabel();

void durationChanged(qint64 duration);

void positionChanged(qint64 progress);

void loadsong(const QString& folderPath);

void playCurrentSong();

void playPreviousSong();

void on\_pushButton\_Volume\_clicked();

void on\_pushButton\_Stop\_clicked();

void on\_pushButton\_Pause\_clicked();

void on\_pushButton\_Play\_clicked();

void on\_pushButton\_nextSong\_clicked();

void on\_pushButton\_previousSong\_clicked();

void on\_actionOpen\_Audio\_File\_triggered();

void on\_pushButton\_9\_clicked();

void on\_pushButton\_10\_clicked();

void on\_pushButton\_11\_clicked();

void on\_pushButton\_12\_clicked();

private:

Ui::MainWindow \*ui;

void updateduration(qint64 duration);

bool IS\_Muted = false;

QMediaPlayer \*MPlayer;

QAudioOutput \*audioOutput;

qint64 Mduration;

void fetchWeatherData();

void onWeatherDataReceived(QNetworkReply \*reply);

void updateTime();

QTimer \*timer;

QLabel \*dateTimeLabel;

};

#endif // MAINWINDOW\_H

### 6.2 Source Files

#### 6.2.1 mainwindow.cpp

#include "mainwindow.h"

#include "./ui\_mainwindow.h"

#include <QCheckBox>

#include <QTimer>

#include<QDateTime>

#include <QDebug>

using namespace std;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

MPlayer = new QMediaPlayer();

audioOutput =new QAudioOutput();

MPlayer->setAudioOutput(audioOutput);

ui->pushButton\_Play->setIcon(style()->standardIcon(QStyle::SP\_MediaPlay));

ui->pushButton\_Pause->setIcon(style()->standardIcon(QStyle::SP\_MediaPause));

ui->pushButton\_Stop->setIcon(style()->standardIcon(QStyle::SP\_MediaStop));

ui->pushButton\_Volume->setIcon(style()->standardIcon(QStyle::SP\_MediaVolume));

ui->pushButton\_nextSong->setIcon(style()->standardIcon(QStyle::SP\_MediaSkipForward));

ui->pushButton\_previousSong->setIcon(style()->standardIcon(QStyle::SP\_MediaSkipBackward));

connect(MPlayer,&QMediaPlayer::durationChanged,this, &MainWindow::durationChanged);

connect(MPlayer,&QMediaPlayer::positionChanged,this, &MainWindow::positionChanged);

ui->horizontalSlider->setRange(0,MPlayer->duration()/1000);

ui->groupBox\_2->hide();

QTimer\* timer = new QTimer(this); //to create a timer

connect(timer, &QTimer::timeout, this, &MainWindow::updateLabel); //to connect the timer to update label slot

timer->start(10000); // Update every 10 seconds

ui->label\_15->setText("Weather: ");

fetchWeatherData();

dateTimeLabel = ui->label1;

if(dateTimeLabel) {

// Create a timer to update the time every second

timer = new QTimer(this);

connect(timer, &QTimer::timeout, this, &MainWindow::updateTime);

timer->start(1000); // Update every 1000 milliseconds (1 second)

// Update the time immediately

updateTime();

} else {

qDebug() << "Label not found!";

}

}

MainWindow::~MainWindow()

{

delete ui;

}

QHash<QDate, QVector<QVector<QString>>> myHash; // hash with date as keys and nested vec as value for reminders

QVector<QString> ToDo; //dynamic array to store tasks

void MainWindow::on\_push\_Button\_clicked() // to add reminders

{

ui->remin->clear();

QString title = ui->title->text();

QString description = ui->desc->text();

QTime time = ui->t1->time();

QString strTime=time.toString("hh:mm:ss");

QString email = ui->email->text();

QDate date = ui->calendarWidget->selectedDate();

QString formattedDate = date.toString("yyyy-MM-dd");

QString dataString;

dataString = QString();

QVector<QString> dataArray; //vector with string of reminder details

dataArray.append(title);

dataArray.append(description);

dataArray.append(strTime);

dataArray.append(email);

QVector<QVector<QString>> dateArray; // nested vector with string inside inner vector

dateArray.append(dataArray);

myHash[date]=dateArray;

QVector<QVector<QString>> value = myHash.value(date); //value stores vector with reminders of a date

for (const QVector<QString> &inner:value) //

{

dataString+="Date : "+formattedDate+"\n";

dataString+="Title : "+inner[0]+"\n";

dataString+="Description : "+inner[1]+"\n";

dataString+="Time : "+inner[2]+"\n";

dataString+="Email : "+inner[3]+"\n\n";

}

ui->remin->setText(dataString);

ui->title->clear();

ui->desc->clear();

ui->email->clear();

ui->lineEdit\_6->clear();

}

void MainWindow::on\_calendarWidget\_clicked(const QDate &date) //to show reminders of date selected on calender

{

ui->remin->clear();

QString dataString;

QVector<QVector<QString>> value = myHash.value(date);

QString formattedDate = date.toString("yyyy-MM-dd");

if (!value.isEmpty())

{

for (const QVector<QString> &inner:value)

{

dataString+="Date : "+formattedDate+"\n";

dataString+="Title : "+inner[0]+"\n";

dataString+="Description : "+inner[1]+"\n";

dataString+="Time : "+inner[2]+"\n";

dataString+="Email : "+inner[3]+"\n\n";

}

ui->remin->setText(dataString); // to show reminders of the selected date

}

}

void MainWindow::on\_pushButton\_6\_clicked() // to search reminder

{

ui->remin->clear();

int count=0; //to count number of matched reminders

QString reminder; //to store reminder

QString title = ui->lineEdit\_6->text(); //take title of reminder

QHash<QDate, QVector<QVector<QString>>>::const\_iterator it; //to create a Qhash iterator it, :: is used for scope resolution

for (it = myHash.constBegin(); it != myHash.constEnd(); ++it) //to iterate thru all dates

{

QDate key = it.key(); //get the date

QString formattedDate = key.toString("yyyy-MM-dd"); // Define the format you want

QVector<QVector<QString>> value = it.value(); //get all the reminders for that date

for (const QVector<QString> &inner:value) //to iterate thru each reminder of a date 'key', &->reference to inner and does not make copy

{

QString title\_1=inner[0];

if (title\_1==title)

{

count+=1;

reminder+="Date : "+formattedDate+"\n";

reminder+="Title : "+inner[0]+"\n";

reminder+="Description : "+inner[1]+"\n";

reminder+="Time : "+inner[2]+"\n";

reminder+="Email : "+inner[3]+"\n\n";

}

}

}

if (count>0)

{

ui->remin->setText(reminder);

}

else

{

ui->remin->setText("No reminders found");

}

//ui->lineEdit\_6->clear();

}

void MainWindow::on\_pushButton\_2\_clicked() // to delete reminder

{

ui->remin->clear();

QString title = ui->lineEdit\_6->text(); // title of reminder to delete

int count = 0; // number of deleted reminders

QHash<QDate, QVector<QVector<QString>>>::iterator it\_del; // Use a non-const iterator to modify the hash

for (it\_del = myHash.begin(); it\_del != myHash.end(); /\* no need to increment here \*/) // Iterate through all dates

{

QVector<QVector<QString>>& value = it\_del.value(); // Get all the reminders for that date by reference

QVector<QVector<QString>>::iterator innerIt = value.begin(); // iterator for value vector

while (innerIt != value.end()) // Iterate through each reminder of a date 'key'

{

QString title\_1 = (\*innerIt)[0];

if (title\_1 == title)

{

innerIt = value.erase(innerIt); // Remove the inner vector if title matches

++count;

}

else

{

++innerIt;

}

}

// Remove the date entry if all reminders for that date were deleted

if (value.isEmpty())

{

it\_del = myHash.erase(it\_del);

}

else

{

++it\_del; // Move to the next date

}

}

if (count > 0)

{

QString fin = QString::number(count);

ui->remin->setText(fin + " reminder(s) deleted.");

}

else

{

ui->remin->setText("No reminders found.");

}

ui->lineEdit\_6->clear();

}

void MainWindow::on\_pushButton\_3\_clicked() //to open edit reminder window

{

ui->groupBox\_2->show();

ui->label\_15->move(1230,0);

ui->label\_16->resize(919,60);

this->resize(1500, 740);

int count=0;

ui->remin->clear();

QString title = ui->lineEdit\_6->text(); //take title of reminder

QHash<QDate, QVector<QVector<QString>>>::const\_iterator it; //to create a Qhash iterator it

for (it = myHash.constBegin(); it != myHash.constEnd(); ++it) //to iterate thru all dates

{

QDate key = it.key(); //get the date

QVector<QVector<QString>> value = it.value(); //get all the reminders for that date

for (const QVector<QString> &inner:value) //to iterate thru each reminder of a date 'key'

{

QString title\_1=inner[0];

if (title\_1==title)

{

count+=1;

ui->lineEdit->setText(inner[0]);//title

ui->lineEdit\_3->setText(inner[1]);//desc

ui->timeEdit\_3->setTime(QTime::fromString(inner[2])); //time

ui->lineEdit\_4->setText(inner[3]);//email

break;

}

}

if (count==1)

{

break;

}

}

}

void MainWindow::on\_pushButton\_5\_clicked() //to update reminder

{

ui->remin->clear();

QString title = ui->lineEdit\_6->text(); // title of reminder to delete

//int count = 0; // number of deleted reminders

QHash<QDate, QVector<QVector<QString>>>::iterator it\_del; // Use a non-const iterator to modify the hash

for (it\_del = myHash.begin(); it\_del != myHash.end(); /\* no need to increment here \*/) // Iterate through all dates

{

QVector<QVector<QString>>& value = it\_del.value(); // Get all the reminders for that date by reference

QVector<QVector<QString>>::iterator innerIt = value.begin(); // iterator for value vector

while (innerIt != value.end()) // Iterate through each reminder of a date 'key'

{

QString title\_1 = (\*innerIt)[0];

if (title\_1 == title)

{

innerIt = value.erase(innerIt); // Remove the inner vector if title matches

//++count;

}

else

{

++innerIt;

}

}

// Remove the date entry if all reminders for that date were deleted

if (value.isEmpty())

{

it\_del = myHash.erase(it\_del);

}

else

{

++it\_del; // Move to the next date

}

}

ui->remin->clear();

QString title2 = ui->lineEdit->text();

QString description2 = ui->lineEdit\_3->text();

QTime time2 = ui->timeEdit\_3->time();

QString strTime=time2.toString("hh:mm:ss");

QString email2 = ui->lineEdit\_4->text();

QDate date = ui->calendarWidget->selectedDate();

QString formattedDate = date.toString("yyyy-MM-dd");

QString dataString;

dataString = QString();

QVector<QString> dataArray; //vector with string of reminder details

dataArray.append(title2);

dataArray.append(description2);

dataArray.append(strTime);

dataArray.append(email2);

QVector<QVector<QString>> dateArray; // nested vector with string inside inner vector

dateArray.append(dataArray);

myHash[date]=dateArray;

QVector<QVector<QString>> value = myHash.value(date); //value stores vector with reminders of a date

for (const QVector<QString> &inner:value) //

{

dataString+="Date : "+formattedDate+"\n";

dataString+="Title : "+inner[0]+"\n";

dataString+="Description : "+inner[1]+"\n";

dataString+="Time : "+inner[2]+"\n";

dataString+="Email : "+inner[3]+"\n\n";

}

ui->label\_11->setText(dataString);

ui->remin->setText("Updated successfully");

}

void MainWindow::on\_pushButton\_4\_clicked() //cancel update

{

ui->groupBox\_2->hide();

ui->label\_15->move(960,0);

ui->label\_16->resize(649,60);

this->resize(1230, 740);

}

void MainWindow::on\_pushButton\_7\_clicked() //close after update

{

ui->groupBox\_2->hide();

ui->label\_15->move(960,0);

ui->label\_16->resize(649,60);

this->resize(1230, 740);

ui->label\_11->clear();

}

void MainWindow::on\_pushButton\_clicked() // to add task to To-Do list

{

QString task=ui->lineEdit\_7->text();

ToDo.append(task);

QCheckBox \* newCheckbox = new QCheckBox(task, this);

newCheckbox->setChecked(false);

ui->verticalLayout->addWidget(newCheckbox);

ui->lineEdit\_7->clear();

}

void MainWindow::on\_pushButton\_8\_clicked() // to delete task from To-Do list

{

QString taskToDelete = ui->lineEdit\_7->text(); // Get the text input for the task to delete

// Iterate through the widgets in the vertical layout

for (int i = 0; i < ui->verticalLayout->count(); ++i)

{

QCheckBox\* checkbox = qobject\_cast<QCheckBox\*>(ui->verticalLayout->itemAt(i)->widget());

if (checkbox && checkbox->text() == taskToDelete)

{

// If a checkbox with matching text is found, remove it from the layout and delete it

ui->verticalLayout->removeWidget(checkbox);

delete checkbox;

break; // Exit the loop after deleting the checkbox

}

}

ui->lineEdit\_7->clear();

}

void MainWindow::updateLabel() //to show quotes dynamically with timer

{

static int currentIndex = 0;

const QStringList textArray = {

"Believe you can, and you're halfway there. - Theodore Roosevelt",

"Don't watch the clock; do what it does. Keep going. - Sam Levenson",

"Success is not final, failure is not fatal: It is the courage to continue that counts. - Winston Churchill",

"Hard work beats talent when talent doesn't work hard. - Tim Notke",

"The only way to do great work is to love what you do. - Steve Jobs",

"You are never too old to set another goal or to dream a new dream. - C.S. Lewis",

"The future belongs to those who believe in the beauty of their dreams. - Eleanor Roosevelt",

"The only limit to our realization of tomorrow will be our doubts of today. - Franklin D. Roosevelt",

"What lies behind us and what lies before us are tiny matters compared to what lies within us. - Ralph Waldo Emerson",

"Your time is limited, so don't waste it living someone else's life. - Steve Jobs",

"It always seems impossible until it's done. - Nelson Mandela",

"Don't be pushed around by the fears in your mind. Be led by the dreams in your heart. - Roy T. Bennett",

"The harder you work for something, the greater you'll feel when you achieve it. - Unknown",

"You miss 100% of the shots you don't take. - Wayne Gretzky",

"The only way to do great work is to love what you do. - Steve Jobs",

"Success is not the key to happiness. Happiness is the key to success. If you love what you are doing, you will be successful. - Albert Schweitzer",

"The biggest risk is not taking any risk. In a world that is changing quickly, the only strategy that is guaranteed to fail is not taking risks. - Mark Zuckerberg",

"The only person you are destined to become is the person you decide to be. - Ralph Waldo Emerson",

"You have within you the strength, the patience, and the passion to reach for the stars to change the world. - Harriet Tubman",

"Believe in yourself and all that you are. Know that there is something inside you that is greater than any obstacle. - Christian D. Larson"

};

// Update the label text with the current string from the array

ui->label\_5->setText(textArray[currentIndex]);

// Increment the index for the next iteration

currentIndex = (currentIndex + 1) % textArray.size();

}

void MainWindow::durationChanged(qint64 duration)

{

Mduration= duration/1000;

ui->horizontalSlider->setMaximum(Mduration);

}

void MainWindow::positionChanged(qint64 progress)

{

if(!ui->horizontalSlider->isSliderDown())

{

ui->horizontalSlider->setValue(progress/1000);

}

updateduration(progress/1000);

}

QList<QString> playlist;

int currentSongIndex = -1;

void MainWindow::loadsong(const QString &folderPath)

{

QDir dir(folderPath);

QStringList filters;

filters << "\*.mp3"; // You can add more file extensions if needed

dir.setNameFilters(filters);

QStringList fileList = dir.entryList(QDir::Files);

for (const QString& file : fileList) {

QString filePath = dir.absoluteFilePath(file);

playlist.append(filePath);

}

if (!playlist.isEmpty()) {

currentSongIndex = 0; // Start playing the first song

playCurrentSong();

}

}

void MainWindow::playCurrentSong()

{

if (currentSongIndex >= 0 && currentSongIndex < playlist.size()) {

QString currentSong = playlist[currentSongIndex];

MPlayer->setSource(QUrl::fromLocalFile(currentSong));

QFileInfo fileInfo(currentSong); //to print current song details

ui->label\_12->setText(fileInfo.fileName());

}

else {

// Invalid currentSongIndex, handle this case (e.g., show a message)

QMessageBox::critical(this, tr("Error"), tr("Invalid song index."));

}

}

void MainWindow::playPreviousSong()

{

if (currentSongIndex > 0 && currentSongIndex < playlist.size()) {

currentSongIndex--;

playCurrentSong();

}

else {

// Already at the beginning of the playlist, handle this case (e.g., show a message)

QMessageBox::information(this, tr("Start of Playlist"), tr("Already at the beginning of the playlist."));

}

}

void MainWindow::on\_pushButton\_Volume\_clicked()

{

if(IS\_Muted==false)

{

ui->pushButton\_Volume->setIcon(style()->standardIcon(QStyle::SP\_MediaVolumeMuted));

IS\_Muted=true;

audioOutput->setMuted(true);

}

else

{

ui->pushButton\_Volume->setIcon(style()->standardIcon(QStyle::SP\_MediaVolume));

IS\_Muted=false;

audioOutput->setMuted(false);

}

}

/\*

void MainWindow::on\_actionOpen\_File\_triggered()

{

//QString FileName = QFileDialog::getOpenFileName(this,tr("Select Audio File"),"",tr("MP3 Files(\*.mp3)"));

//MPlayer->setSource(QUrl(FileName));

//QFileInfo fileinfo(FileName);

//ui->label->setText(fileinfo.fileName());

QString folderPath = QFileDialog::getExistingDirectory(this, tr("Select Folder"), "", QFileDialog::ShowDirsOnly);

if (!folderPath.isEmpty()) {

loadsong(folderPath);

}

}

\*/

void MainWindow::on\_pushButton\_Stop\_clicked()

{

MPlayer->stop();

}

void MainWindow::on\_pushButton\_Pause\_clicked()

{

MPlayer->pause();

}

void MainWindow::on\_pushButton\_Play\_clicked()

{

MPlayer->play();

}

void MainWindow::updateduration(qint64 duration)

{

QString timestr;

if(duration || Mduration)

{

QTime CurrentTime((duration / 3600) % 60,(duration/60)% 60,duration % 60,(duration \* 1000) % 1000);

QTime totalTime((Mduration / 3600) % 60,(Mduration/60)% 60,Mduration % 60,(Mduration \* 1000) % 1000);

QString format = "mm:ss";

if(Mduration >3600)

format ="hh:mm:ss";

ui->label\_14->setText(CurrentTime.toString(format));

ui->label\_13->setText(totalTime.toString(format));

}

}

void MainWindow::on\_pushButton\_nextSong\_clicked()

{

if (currentSongIndex < playlist.size() - 1) {

currentSongIndex++;

playCurrentSong();

}

else {

// No more songs in the playlist, handle this case (e.g., show a message)

QMessageBox::information(this, tr("End of Playlist"), tr("No more songs in the playlist."));

}

}

void MainWindow::on\_pushButton\_previousSong\_clicked()

{

playPreviousSong();

}

void MainWindow::on\_actionOpen\_Audio\_File\_triggered()

{

QString folderPath = QFileDialog::getExistingDirectory(this, tr("Select Folder"), "", QFileDialog::ShowDirsOnly);

if (!folderPath.isEmpty()) {

loadsong(folderPath);

}

}

void MainWindow::fetchWeatherData()

{

QNetworkAccessManager \*manager = new QNetworkAccessManager(this);

connect(manager, &QNetworkAccessManager::finished, this, &MainWindow::onWeatherDataReceived);

// Replace "YOUR\_API\_KEY" with your OpenWeatherMap API key

QString apiKey = "c8e2df94491adbde9bb93df1271efea6";

QString city = "Noida";

QString urlStr = "https://api.openweathermap.org/data/2.5/weather?q=" + city + "&appid=" + apiKey;

QUrl url(urlStr);

QNetworkRequest request(url);

manager->get(request);

}

void MainWindow::onWeatherDataReceived(QNetworkReply \*reply)

{

if (reply->error() == QNetworkReply::NoError) {

QByteArray responseData = reply->readAll();

QJsonDocument jsonResponse = QJsonDocument::fromJson(responseData);

if (!jsonResponse.isNull()) {

QJsonObject jsonObj = jsonResponse.object();

if (jsonObj.contains("main") && jsonObj["main"].isObject()) {

double temperatureKelvin = jsonObj["main"].toObject()["temp"].toDouble();

double temperatureCelsius = temperatureKelvin - 273.15;

QJsonValue weatherValue = jsonObj["weather"].toArray()[0].toObject()["description"];

QString weatherDesc = weatherValue.toString();

QLabel \*weatherLabel = ui->label\_15;

if (weatherLabel) {

QString displayText = "Temperature: " + QString::number(temperatureCelsius, 'f', 2) + "°C\n";

displayText += "Weather: " + weatherDesc;

weatherLabel->setText(displayText);

}

// Display the weather description on the label

}

} else {

qDebug() << "Error: " << reply->errorString();

}

reply->deleteLater();

}

}

void MainWindow::updateTime()

{

if(dateTimeLabel) {

// Get current date and time

QDateTime currentDateTime = QDateTime::currentDateTime();

// Display date and time in a specific format

QString dateTimeStr = currentDateTime.toString("yyyy-MM-dd hh:mm:ss");

// Update the label with the current date and time

dateTimeLabel->setText(dateTimeStr);

}

}

QVector<QString> quickNotes; //dynamic array to store quick notes

void MainWindow::on\_pushButton\_9\_clicked()

{

QString note = ui->plainTextEdit->toPlainText(); //stores the note

quickNotes.append(note);

/\*QString finalText;

for (const QString& note : quickNotes)

{

finalText.append(note).append("\n");

}\*/

// Display the final text in the QPlainTextEdit

ui->plainTextEdit->setPlainText(note);

}

void MainWindow::on\_pushButton\_10\_clicked()

{

ui->plainTextEdit->setPlainText("");

}

void MainWindow::on\_pushButton\_11\_clicked()

{

QString currentText = ui->plainTextEdit->toPlainText();

if (currentText.isEmpty())

{

// If the plainTextEdit is clear, display the last element of quickNotes

if (!quickNotes.isEmpty())

{

ui->plainTextEdit->setPlainText(quickNotes.last());

}

else

{

ui->plainTextEdit->setPlainText("No notes available.");

}

}

else

{

// If there is text in plainTextEdit, find the current text in quickNotes and display the previous text from the list

int index = quickNotes.indexOf(currentText);

if (index > 0)

{

ui->plainTextEdit->setPlainText(quickNotes[index - 1]);

}

else

{

ui->plainTextEdit->setPlainText("No previous note available.");

}

}

}

void MainWindow::on\_pushButton\_12\_clicked()

{

QString currentText = ui->plainTextEdit->toPlainText();

if (currentText.isEmpty())

{

// If the plainTextEdit is clear, display the last element of quickNotes

if (!quickNotes.isEmpty())

{

ui->plainTextEdit->setPlainText(quickNotes.last());

}

else

{

ui->plainTextEdit->setPlainText("No notes available.");

}

}

else

{

// If there is text in plainTextEdit, find the current text in quickNotes and display the next text from the list

int index = quickNotes.indexOf(currentText);

if (index < quickNotes.size()-1)

{

ui->plainTextEdit->setPlainText(quickNotes[index + 1]);

}

else

{

ui->plainTextEdit->setPlainText("No new note available.");

}

}

}

#### 6.2.2 main.cpp

#include "mainwindow.h"

#include <QApplication>

#include <QTimer>

#include <QTime>

#include <QStringList>

#include<QtMultimedia/QtMultimedia>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

w.move(10,20);

return a.exec();

}

## Chapter 7: Results

#### 7.1 Functioning of Date, Time and weather widget

#### 

#### 7.2 Functioning of Quote Generator

A screenshot of a phone

Description automatically generated A screenshot of a cellphone

Description automatically generated

The quote changes after every 10 seconds

#### 7.3 Functioning of Music Player

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Select the directory where your music folder is made.

A screenshot of a video player

Description automatically generated

The first song in the folder i.e the song that is on index 0 of the playlist is played .

A screenshot of a computer

Description automatically generated

A message box is displayed if there are no more songs in the list when the nest button is clicked .

A screenshot of a computer

Description automatically generated

A message box is displayed when we are already at the beginning of the playlist.

A screenshot of a video player

Description automatically generated

The mute button mutes the audio output while the music is still playing and the progress bar gets updated.

A screenshot of a music player

Description automatically generated

#### 7.4 Functioning of the Quick Notes

A note is entered and saved.

A screenshot of a computer

Description automatically generated

A note is written and not saved.

A screenshot of a computer

Description automatically generated

On pressing the undo button we shift to the previous note.

A screenshot of a computer

Description automatically generated

On pressing the clear button .

A screenshot of a computer

Description automatically generated

#### 7.5 Functioning of the To-Do List

On pressing the add task button.

A screenshot of a computer task list

Description automatically generated

A screenshot of a computer task

Description automatically generated

Deleting a Task

A screenshot of a computer

Description automatically generated A screenshot of a computer task list

Description automatically generated

#### 7.6 Functioning of the calendar widget and reminders

Add reminders for a specific date

A screenshot of a computer

Description automatically generated

Search and Edit Reminders

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Chapter 8: Conclusion

EffiSync is a powerful culmination of innovative features and a thoughtful design, enclosed within a productivity application developed in C++ using the Qt Creator framework. By integrating essential tools for task management, entertainment, and information retrieval, EffiSync aims to modify and enhance the user experience in the realm of productivity applications.

The project's emphasis on data structure implementation and object-oriented design has resulted in a modern and efficient productivity application. Using advanced data structures ensures efficient organization and retrieval of information, contributing to the application's quick and sleek response. The usage of object-oriented principles ensures modularity and code maintainability, allowing for future enhancements and updates.

EffiSync not only serves as a functional tool for managing reminders, to-do lists, and calendar events but also ensures a positive and motivational atmosphere through the inclusion of a music player and motivational quotes. The incorporation of real-time information, such as date, time, and weather updates, enhances the application's utility and relevance in users' daily lives.

As EffiSync provides users with a holistic and personalized productivity experience, the project stands as a testament to the effective utilization of C++ and Qt Creator. It is not merely an application but a dynamic platform that integrates comprehensive functionality, user interface, and efficient coding practices to create an imperative solution for individuals seeking an intuitive productivity tool.

In conclusion, EffiSync is certain to make a significant impact by elevating the standard for productivity applications, offering a blend of functionality, aesthetics, and performance. The journey of developing EffiSync has not only resulted in a feature-rich application but also in a deeper understanding of the harmony between technology and user-centric design in the pursuit of enhancing daily productivity.

Chapter 9: **Bibliography**

1. <https://www.youtube.com/@ProgrammingKnowledge>
2. <https://www.youtube.com/@QtStudios>
3. <https://www.youtube.com/@wearesciber>
4. <https://www.youtube.com/@qtwithketan>
5. <https://www.thesaurus.com/>
6. <https://en.wikipedia.org/wiki/Main_Page>