**overview**

The Customized Feed Algorithm project is an application that provides a tailored Video and article reading experience. There are two main types of Accounts: Admins and Users. Admins manage the content and users, while Users interact with the feed, set their preferences, and engage with articles.

**LOCATION**

EUI, NEW CAPITAL, EGYPT

A person sitting in a chair with a dog and a computer screen

Description automatically generated

Videos and articles are now everywhere, trending, and all on one platform.

**trends**

**The power of:**

**Data structure**

In today's digital age, the sheer volume of information available online can be overwhelming. Users are inundated with news articles, social media posts, and other forms of content, making it challenging to find information that is both relevant and interesting to them. To address this issue, our project aims to develop a "Customized Feed Algorithm" – a personalized feed system designed to deliver tailored content to users based on their preferences and interactions.

Vol. 201

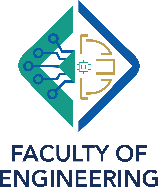
May 25, 2024

**Customized feed**

Social media anxiety

EST. 1919

A logo with text on it

Description automatically generated

* **The data structure we used:**

1. Custom vector: -

* Purpose:
* We used **CustomVector** becauseis a dynamically resizable array implementation used to store and manage data objects such as Users, Admins, and Articles.
* Methods:
* **push:** Adds an element to the end of the vector.
* **erase:** Removes an element at a specified index.
* **size:** Returns the number of elements in the vector.
* **clear:** Clears the contents of the vector.
* **operator []:** Accesses elements in the vector by index.

1. Max heap: -

* Purpose:
* Maxheap is a data structure used to store user preferences for news categories. It ensures efficient retrieval of the top preferences.
* Methods:
* **insert:** Inserts a new preference node into the heap.
* **removeMax:** Removes the node with the maximum preference value from the heap.
* **buildHeap:** Builds a heap from a given array of preference nodes.
* **heapify:** Maintains the heap property by moving nodes up or down the heap.
* **algorithms:**

1. Heap building: -

* Purpose:
* Heap building and algorithm are used to maintain the heap property in the MaxHeap data structure. This ensures that the most preferred news categories are efficiently accessible.
* Algorithm:
* **BuildHeap:** Constructs a valid heap from an unsorted array of preference nodes.
* **Heapify:** Adjusts the heap structure to maintain the heap property after removing the maximum node.

1. Filtering algorithms: -

* Purpose:
* Filtering algorithms are used to narrow down the list of articles based on specific criteria such as category and type.
* Algorithm:
* **FilterByCategory:** Filters articles based on the specified category.
* **FilterByType:** Filters articles based on the specified type (article or video).
* **Classes:**

1. Date class: -

* Purpose:
* Handles date-related operations such as date initialization, comparison, and conversion to string format.
* Methods:
* **Constructors** for initializing dates.
* **Methods** for setting and getting dates.
* **Overloaded operators** for date comparison.
* **Functions** for calculating the number of days between dates.

1. Accounts class:-

* Purpose:
* It’s the base class that the users and admins class will be inherited from.
* Methods:
* Constructors for initialization of the objects.
* Setters and getters.
* Functions for adding articles and videos.
* All those methods will be inherited by users and admins classes.

1. Users class: -

* Purpose:
* Manages user preferences and interactions with articles.
* Methods:
* **Functions** for user login and registration.
* **Methods** for managing user preferences and seen articles.
* **Operations** for liking, disliking, and retrieving articles.
* **Functions to filter the articles based on category and type.**
* **Function for building a max heap for every user based on the score of every article which is calculated by the calscore function based on likes count and publication date.**

1. Admins class: -

* Purpose:
* Manages administrative tasks such as user management and article addition.
* Methods:
* **Functions** for user registration and removal.
* **Operations** for finding users by username.

1. Article class: -

* Purpose:
* Represents news articles with attributes like title, content, category, and publication date.
* Methods:
* **Functions** for loading, saving, and displaying articles.
* **Operations** for filtering articles by category, author, or source.

1. Mainwindow:

* Handles user login and registration.
* MainWindow (QWidget \*parent = nullptr): Constructor, sets up the UI and loads initial data.
* ~MainWindow (): Destructor, cleans up the UI.
* void on\_pushButton\_clicked (): Handles login and registration logic.
* void on\_pushButton\_2\_clicked (): Toggles between login and registration modes.

1. Userform:

* Allows users to interact with articles and videos.
* **userform (QWidget \*parent, Users& u):** Constructor, initializes the UI and loads user data.
* **~userform ():** Destructor, cleans up the UI.
* **void on\_pushButton\_3\_clicked ():** Loads the next article.
* **void on\_pushButton\_2\_clicked ():** Manages the "like" action.
* **void on\_pushButton\_clicked ():** Logs out the user.
* **void on\_pushButton\_4\_clicked ():** Filters articles based on user preferences.
* **void on\_pushButton\_5\_clicked ():** Manages the "not interested" action.
* **void on\_pushButton\_6\_clicked ():** Manages the "read aloud" or "pause video" action.
* **void on\_pushButton\_7\_clicked ():** Opens the post dialog.
* **void closeEvent (QCloseEvent \*event):** Handles application close event.

1. Adminform:

* Allows admins to manage users and content.
* **adminform (QWidget \*parent, Admins& a):** Constructor, initializes the UI and loads admin data.
* **~adminform ():** Destructor, cleans up the UI.
* **void on\_pushButton\_2\_clicked ():** Register a new user if the username is not already registered.
* **void on\_pushButton\_3\_clicked ():** Remove a new user if the username is already registered.
* **void on\_pushButton\_clicked ():** Add a new article or video.
* **void on\_pushButton\_4\_clicked ():** Logout.
* **void on\_pushButton\_5\_clicked ():** Logout.
* **void closeEvent (QCloseEvent \*event):** Handles application close event.

1. Post:

* Allows users to post new articles or videos.
* **post (QWidget parent, Users u):** Constructor, initializes the UI and sets the user context.
* **~post ():** Destructor, cleans up the UI.
* **void on\_pushButton\_clicked ():** Manages the post-submission logic.
* Functionalities: -
* Purpose:

• Implements various application functionalities like loading/saving files, user login, and menu options for both admins and users.

* Methods:

• Functions for loading and saving user/admin data.

• Operations for user login/logout and menu navigation.

* **User’s manual:**
* **Function description: -**

1. **Login:** Users can log in to the system by entering their username and password.
2. **Registration:** Admins and users can register new user accounts by providing a username and password.
3. **Article Management:** Admins and users can add new articles to the system, specifying details like title, content, category, source, author, and publication date.
4. **Preference Management:** Users can like, or dislike articles based on their preferences, and the system will recommend articles accordingly.
5. **Article Display:** Users can view articles filtered by their preferences, category, and type.
6. **Logout/Exit:** Users and admins can log out of the system or exit the application.

* **User manual: -**

1. **Login/Registration:**

* Upon launching the application, users are prompted to log in with their username and password. If they do not have an account, admins can register new users.

1. **Article and users Management:**

* Admins can add new articles to the system by providing details like title, content, category, source, author, and publication date and they also can register and remove users.

1. **Preference Management:**

* Users can like or dislike articles based on their preferences, and the system will recommend articles accordingly.

1. **Article Display and post:**

* Users can view articles filtered by their preferences, category, author, or source and publish new posts.

1. **Logout/Exit:**

* Users and admins can log out of the system or exit the application when done.
* Login and registration: -
* **Login Screen:** Enter your username and password to log in. If you don't have an account, click "Create One" to switch to the registration screen.
* A screenshot of a computer

  Description automatically generated
* **Registration Screen:** Enter your desired username and password to register a new account. Click "Register" to create your account.

A screenshot of a computer

Description automatically generated

* **User interface: -**
* **Main User Interface:** View articles or videos, like or dislike content, and post new content.

A screenshot of a video game

Description automatically generated

* **Reading an article:** Articles are displayed with their title, category, date, author, and content. Use the "Read Aloud" button to listen to the article.

A screenshot of a computer

Description automatically generated

* **Watching a Video:** Videos are displayed in a video widget.

A screenshot of a video game

Description automatically generated

* **Posting a Video or Article:** To post a video or article, write the title, the category, the source, or the article you want to post, and click “Add.”.
* A screenshot of a video game

  Description automatically generated

* **Admin interface: -**
* **Admin Dashboard:** Manage users and content.

A screenshot of a computer

Description automatically generated

* Posting content: -
* **Post Dialog:** Choose to post an article or video by filling out the required fields and clicking "add".

A screenshot of a computer

Description automatically generated

* Design process:

1. **Requirement Analysis:**

* User functionality
* Users should be able to log in and register.
* Users can read articles or watch videos.
* Users can filter the content
* Users can like or dislike content.
* Users can post new content (articles or videos).
* Admin functionality
* Admins should be able to log in.
* Admins can manage users and content.

1. **Architectural design:**

* The application is structured into several modules, each handling different aspects of the system:
* **MainWindow:** Handles user login and registration.
* **UserForm:** Allows users to interact with content.
* **AdminForm:** Allows admins to manage users and content.
* **Post:** Allows users to post new articles or videos.
* **Data Models:** Accounts, Article, Date, which manage data storage and retrieval.

1. **Data Structure Selection:**

* Chose appropriate data structures like custom vectors for dynamic storage and max heaps for efficient preference management.

1. **Class Design:**

* Designed classes such as Accounts, Users, Admins, Articles, and Date to encapsulate related functionalities and data.

1. **Algorithm Design:**

* Developed algorithms for user authentication, article filtering, preference management, and heap operations.
* Implementation details:

1. **Coding:**

* Implemented the classes, methods, and algorithms in C++ using appropriate data structures and libraries.

1. **Testing:**

* Conducted unit tests for each module and integration tests for system components to ensure functionality, correctness, and efficiency.
* Custom vector time and space complexties:

\* Constructor: customVector()

\* - Time: O(1)

\* - Space: O(1)

\*

\* Copy Constructor: customVector(const customVector& v)

\* - Time: O(n), where n is the size of the vector being copied.

\* - Space: O(n)

\*

\* Copy Assignment Operator: customVector& operator=(const customVector& v)

\* - Time: O(n), where n is the size of the vector being copied.

\* - Space: O(n)

\*

\* Destructor: ~customVector()

\* - Time: O(1)

\* - Space: O(1)

\*

\* getcapacity(), getcurrent(), setcurrent(size\_t c)

\* - Time: O(1)

\* - Space: O(1)

\*

\* push(T data)

\* - Average Time: O(1) (amortized)

\* - Worst-case Time: O(n) (when capacity needs to be doubled)

\* - Space: O(n)

\*

\* insert(size\_t index, T data)

\* - Time: O(n)

\* - Space: O(n)

\*

\* erase(size\_t index)

\* - Time: O(n)

\* - Space: O(1)

\*

\* clear()

\* - Time: O(n)

\* - Space: O(1) (as it frees the allocated memory)

\*

\* operator[](size\_t index), const T& operator[](size\_t index) const

\* - Time: O(1)

\* - Space: O(1)

\*

\* size() const

\* - Time: O(1)

\* - Space: O(1)

\*

\* empty() const

\* - Time: O(1)

\* - Space: O(1)

\*

\* print() const

\* - Time: O(n)

\* - Space: O(1)

\*

\* begin(), end()

\* - Time: O(1)

\* - Space: O(1)

* Max heap time and space complexities:

\* Constructor: MaxHeap()

\* - Time: O(1)

\* - Space: O(1)

\*

\* Constructor with customVector: MaxHeap(customVector<maxHeapNode> v)

\* - Time: O(n)

\* - Space: O(n)

\*

\* parent(int x), left\_child(int x), right\_child(int x)

\* - Time: O(1)

\* - Space: O(1)

\*

\* swap(maxHeapNode &x, maxHeapNode &y)

\* - Time: O(1)

\* - Space: O(1)

\*

\* heapifyup(size\_t index)

\* - Time: O(log n)

\* - Space: O(1)

\*

\* heapifydown(size\_t index)

\* - Time: O(log n)

\* - Space: O(1)

\*

\* insert(maxHeapNode value)

\* - Time: O(log n)

\* - Space: O(1)

\*

\* extract\_max()

\* - Time: O(log n)

\* - Space: O(1)

\*

\* is\_a\_max\_heap()

\* - Time: O(n)

\* - Space: O(1)

\*

\* min\_maxheap()

\* - Time: O(n)

\* - Space: O(1)

\*

\* peaktop()

\* - Time: O(1)

\* - Space: O(1)

\*

\* isEmpty() const

\* - Time: O(1)

\* - Space: O(1)

\*

\* build\_heap()

\* - Time: O(n)

\* - Space: O(1)

\*

\* clear()

\* - Time: O(1)

\* - Space: O(1)

\*

\* heap\_sort()

\* - Time: O(n log n)

\* - Space: O(1)

* How to run:

There are two interfaces to this application

* Consol:
* And this doesn’t contain all the features and we can run it using only the main and the classes and functionalities.
* GUI:
* In GUI we used Qt Creator 6.7.0 and while setting it up chose desktop development.
* Open a new project and put all the classes (Accounts, Article, customVector, MaxHeap) in the project (don’t put the main and functionalities)
* Put all the files in the GUI\_QT6 in the project (don’t take the CMakeLists)
* From CMakeLists take

find\_package(QT NAMES Qt6 Qt5 REQUIRED COMPONENTS Widgets TextToSpeech Multimedia MultimediaWidgets)

find\_package(Qt${QT\_VERSION\_MAJOR} REQUIRED COMPONENTS Widgets TextToSpeech)

And

set(PROJECT\_SOURCES

main.cpp

mainwindow.cpp

mainwindow.h

mainwindow.ui

json.h

Accounts.cpp

Accounts.h

Users.cpp

Admins.cpp

MaxHeap.h

Date.h

Date.cpp

MaxHeap.cpp

Article.cpp

customVector.h

adminform.h

adminform.cpp

adminform.ui

Article.h

userform.h

userform.cpp

userform.ui

post.ui

post.h

post.cpp

)

And

find\_package(Qt6 COMPONENTS Core Gui Widgets Multimedia MultimediaWidgets TextToSpeech REQUIRED)

# Add the executable

# Include the directories for Qt

include\_directories(${Qt6Core\_INCLUDE\_DIRS} ${Qt6Gui\_INCLUDE\_DIRS} ${Qt6Widgets\_INCLUDE\_DIRS} ${Qt6Multimedia\_INCLUDE\_DIRS} ${Qt6MultimediaWidgets\_INCLUDE\_DIRS})

target\_link\_libraries(“YOUR PROJECT NAME” PRIVATE Qt${QT\_VERSION\_MAJOR}::Widgets Qt${QT\_VERSION\_MAJOR}::MultimediaWidgets Qt${QT\_VERSION\_MAJOR}::TextToSpeech)

replace “YOUR PROJECT NAME” with your’s

* Take all the three JSON files and bro.ico from cmake-build-debug directory and put it with the .exe
* Take the Video directory from GUI\_QT>Release and put it with the Dsa\_gui.exe file.
* If you want to run the application without doing all of these steps just go to GUI\_QT>Release and run the Dsa\_gui.exe file