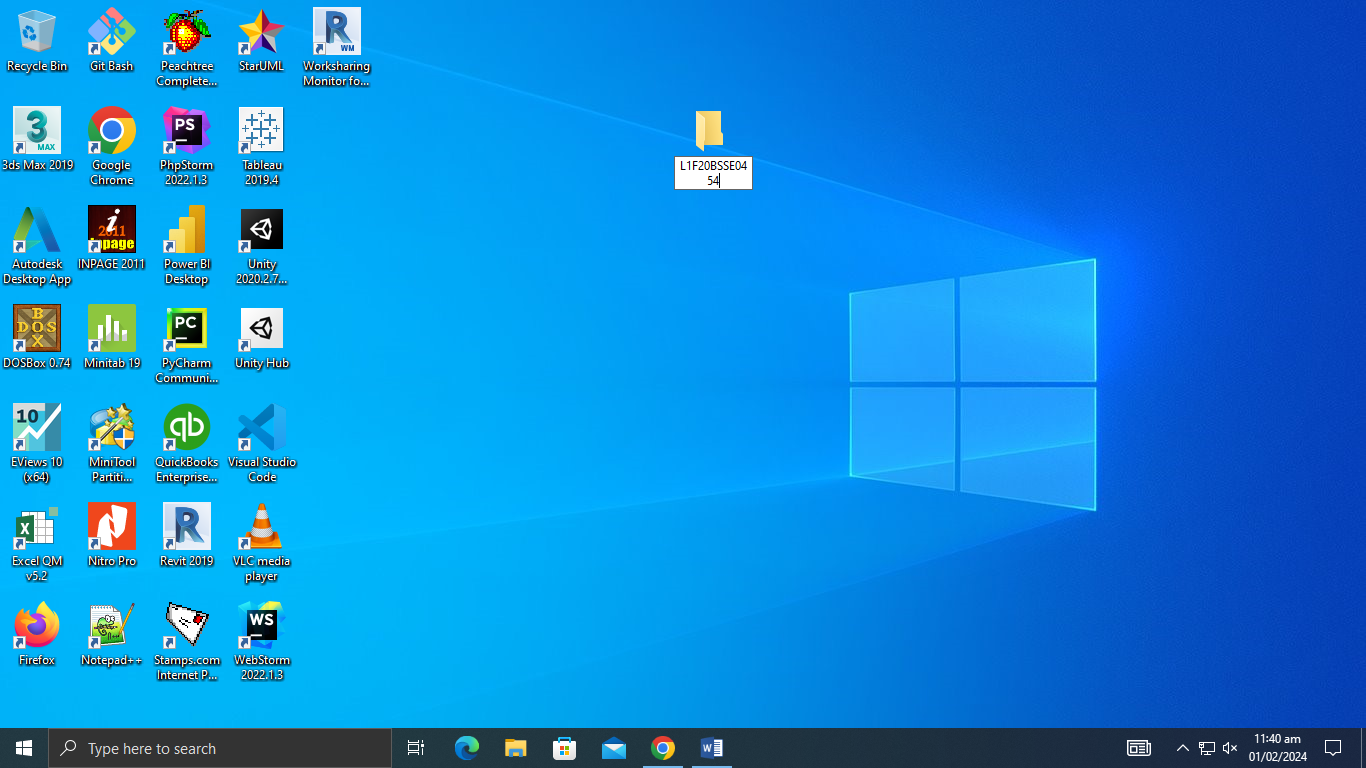
**Name**: Sheroz Akram

**Roll #:** L1F20BSSE0454

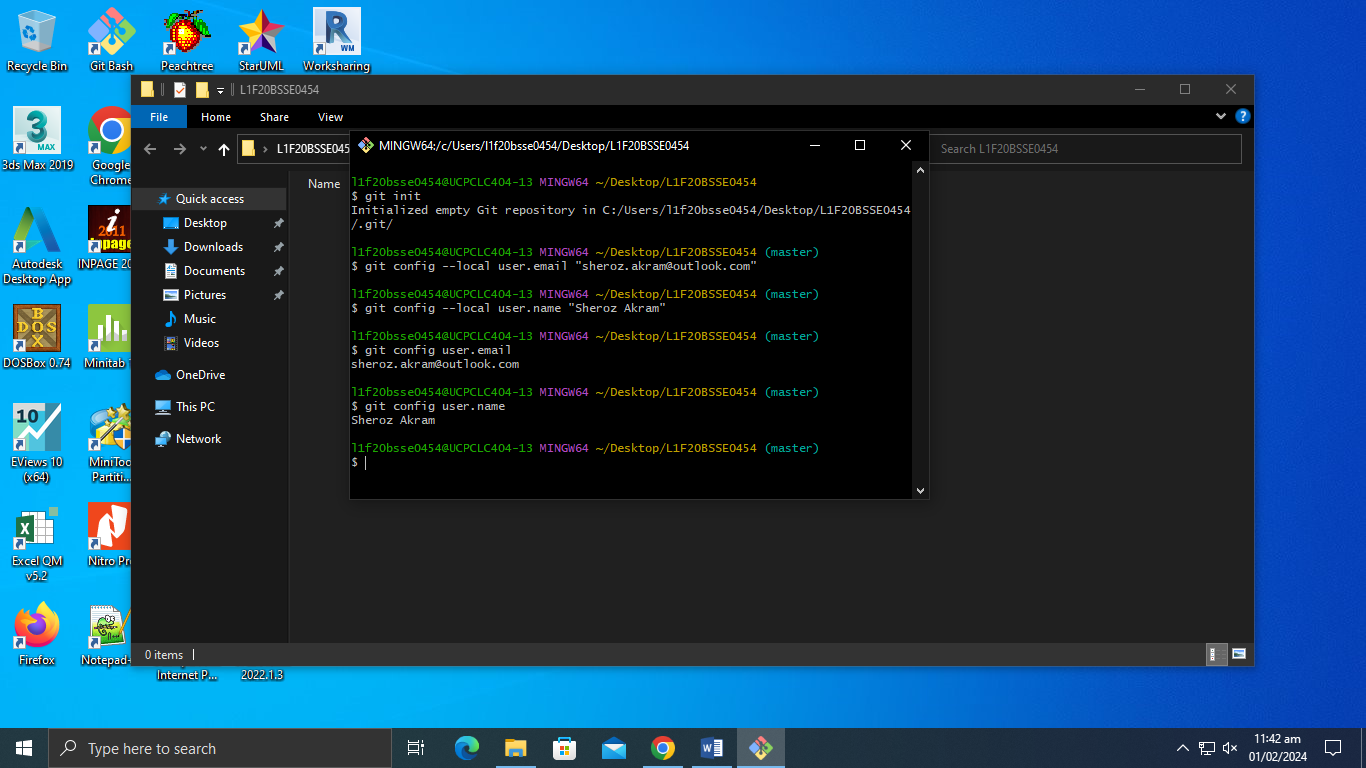
**Section:** R2

**Quesiton # 01:**

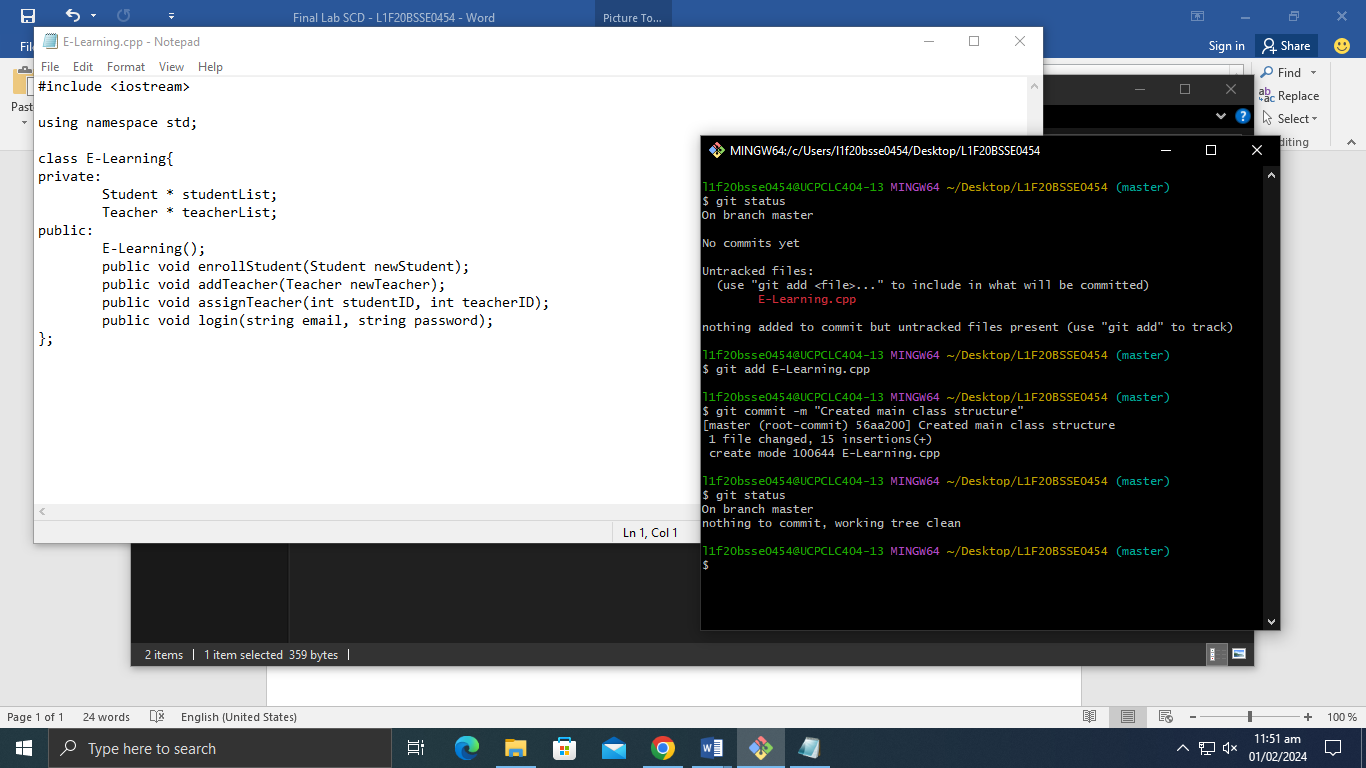
1. Create the project folder



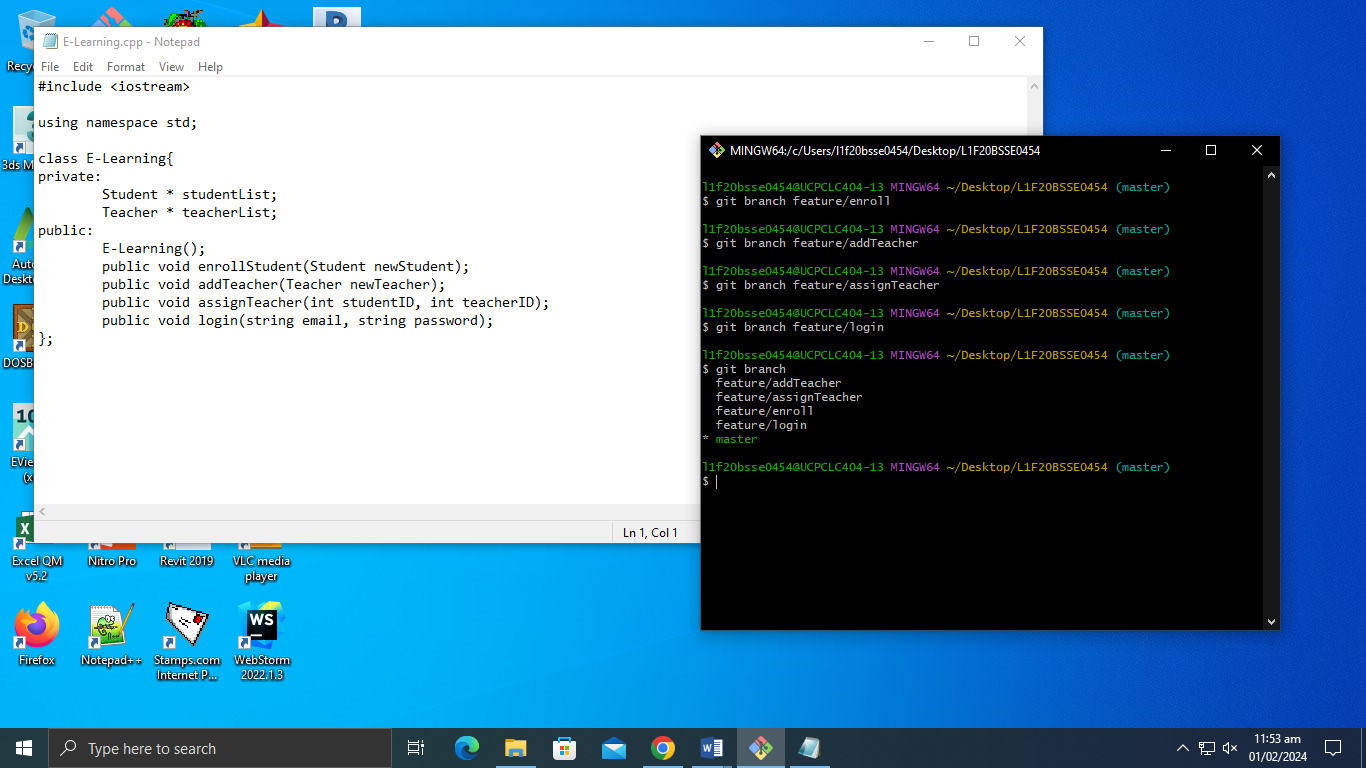
2. Initialize the repository and configure the author



3. Create the main class E-Learning and Commit the changes:

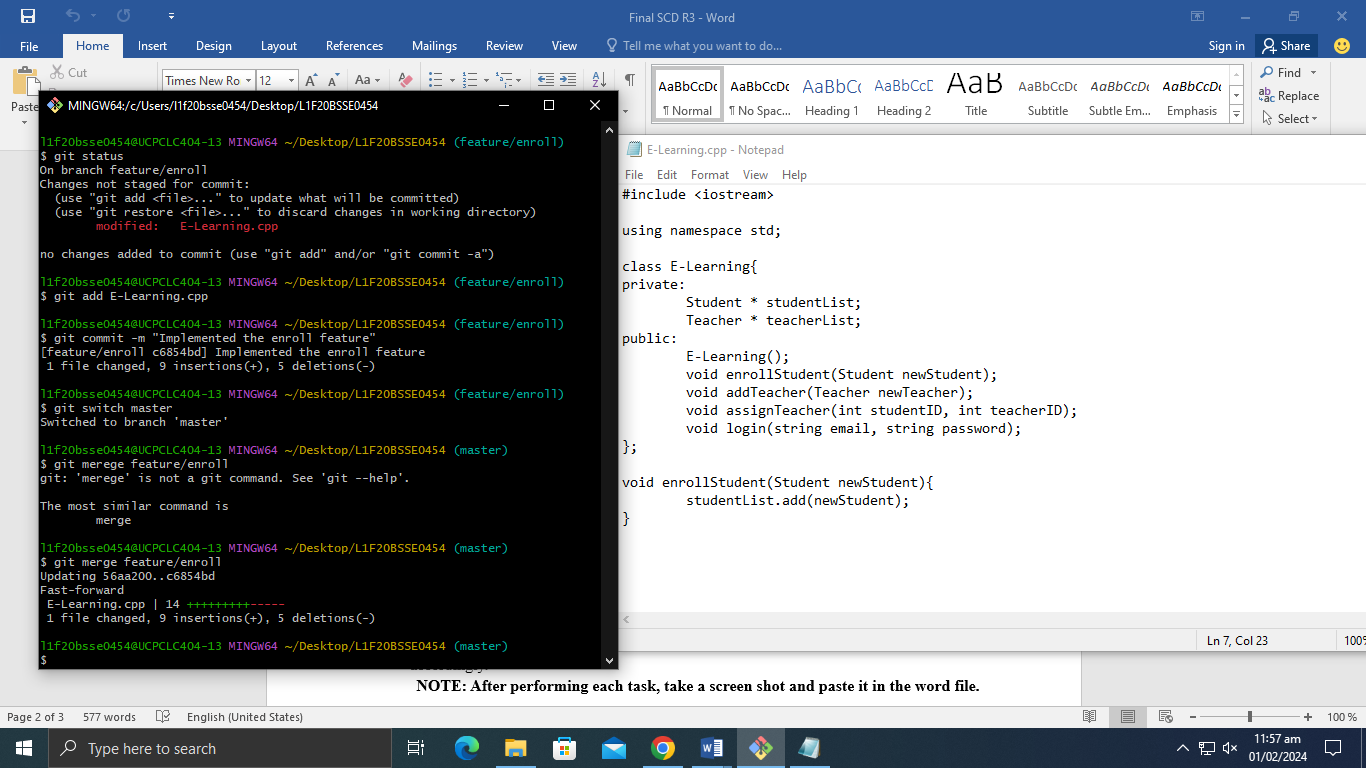


4. Create feature branches for all the feature

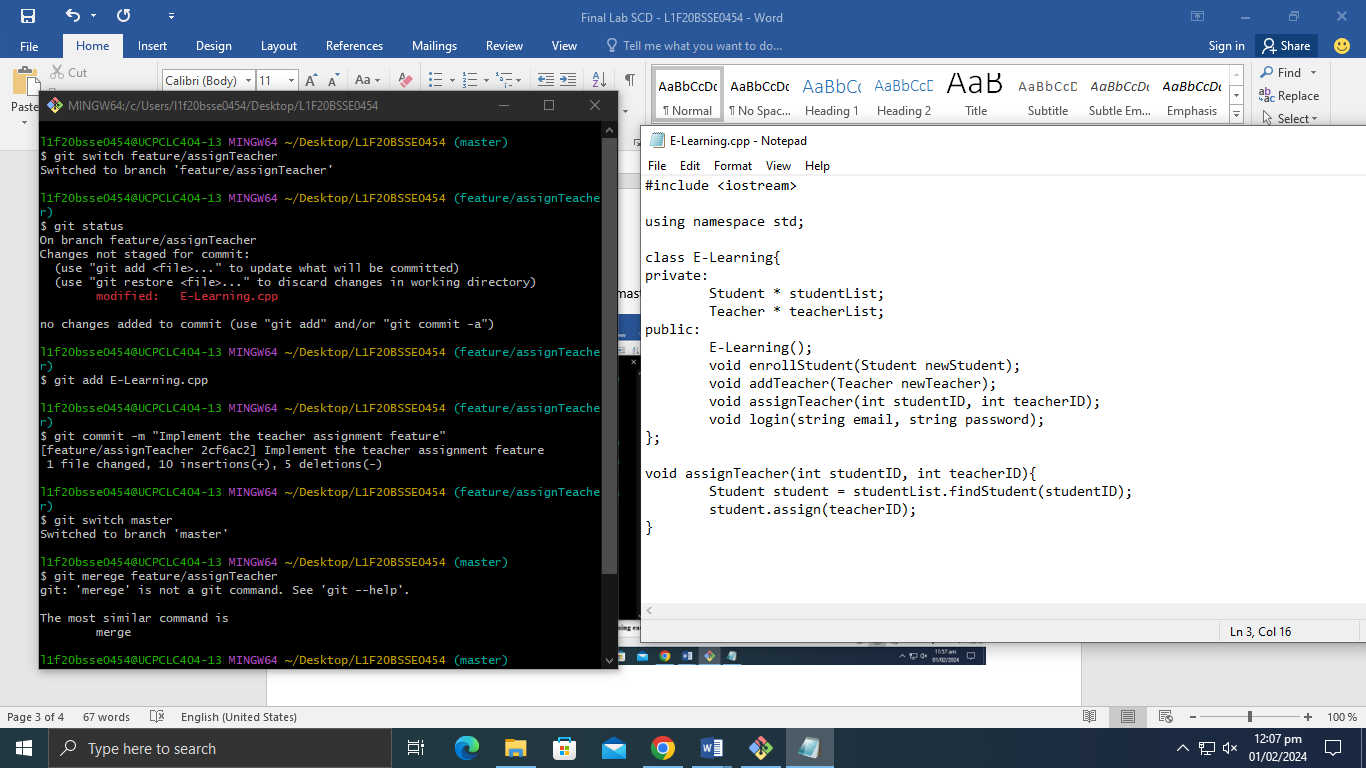


5. Merge each feature branch to the main master branch

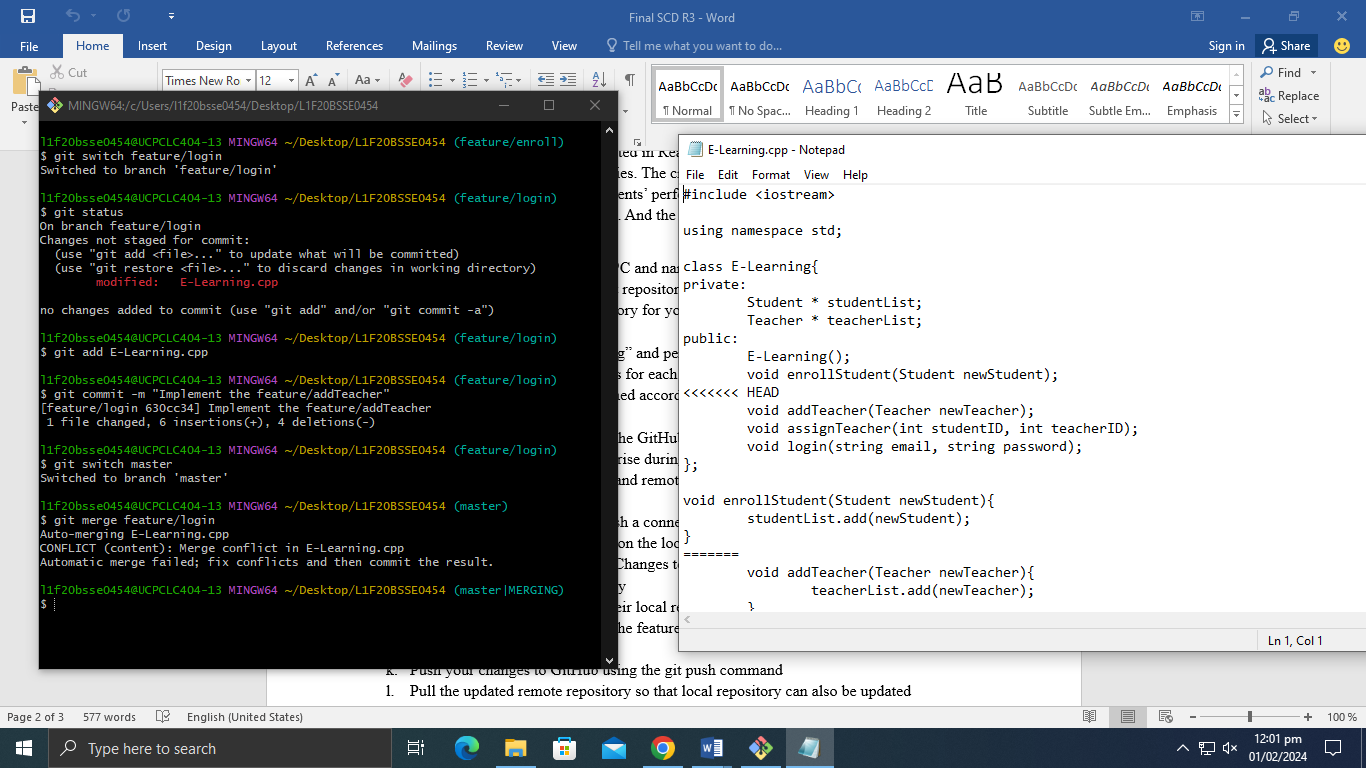
Merge the enrollStudent feature to the main master branch



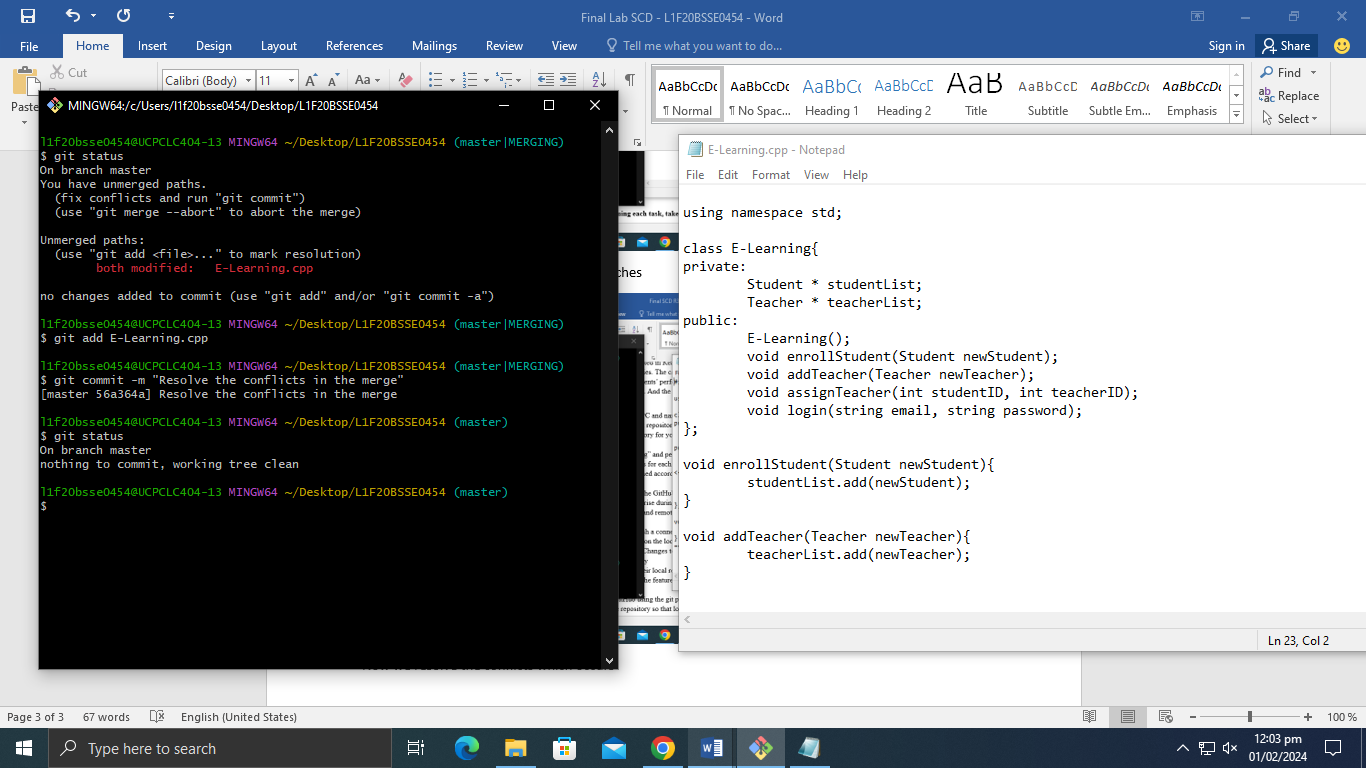
Merge the assignTeacher feature branch



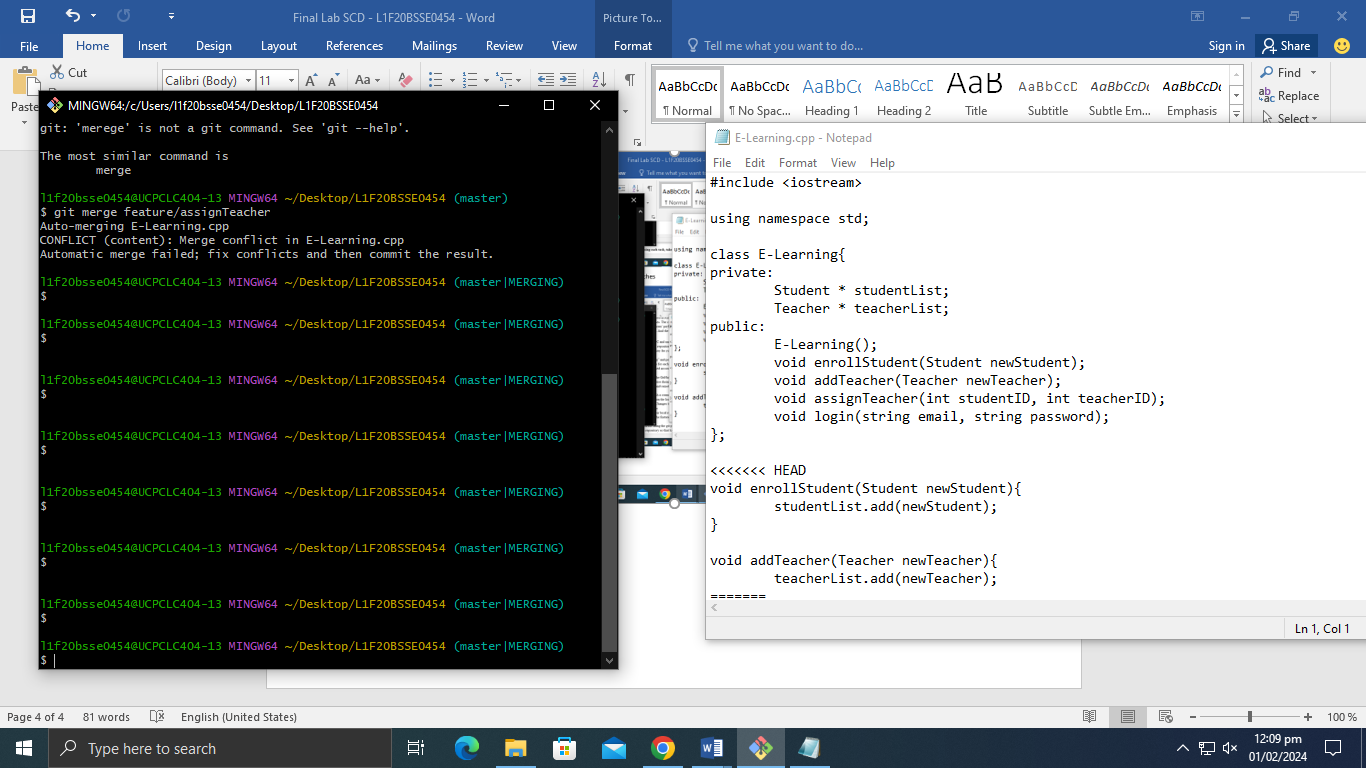
6. Conflicts occurs the merging of the branches



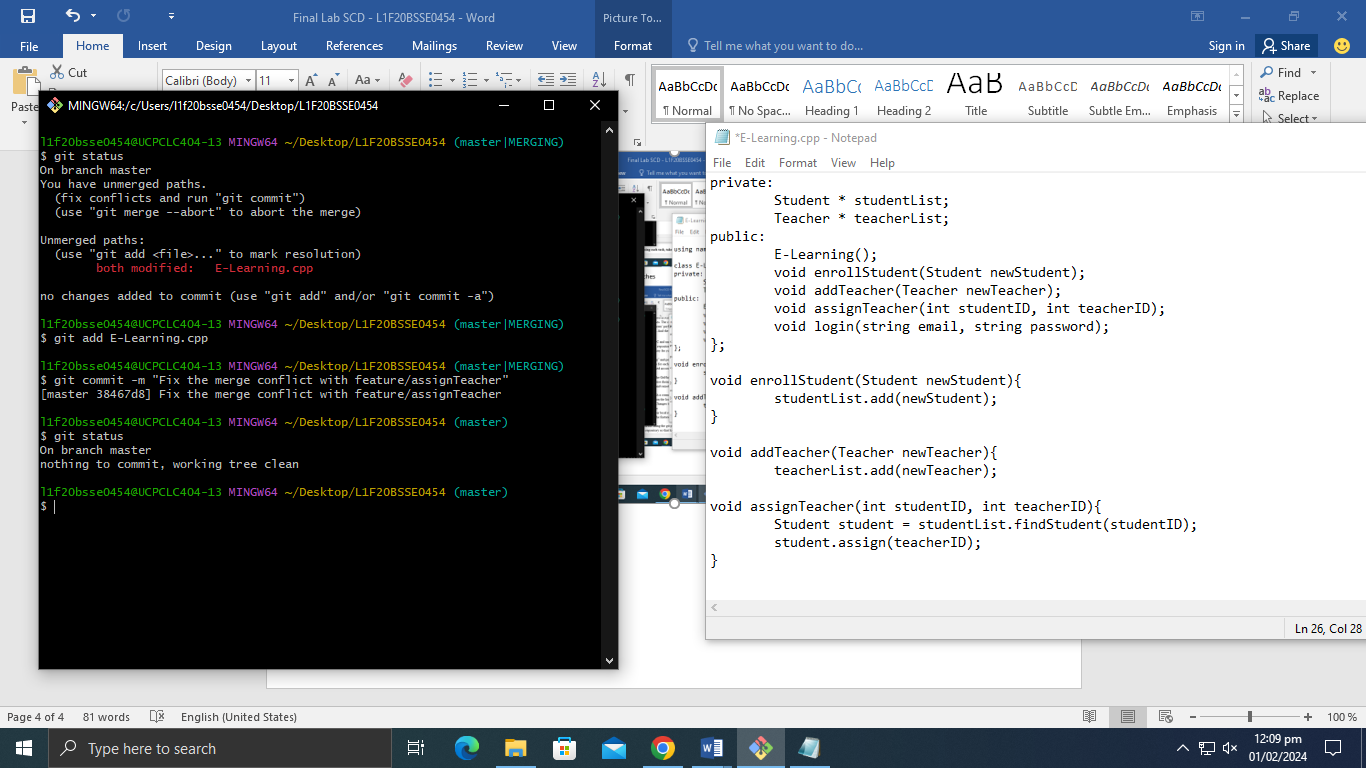
Now we resolve the conflicts which occurs



New Conflicts occur during the merging of feature/assignTeacher to main master branch

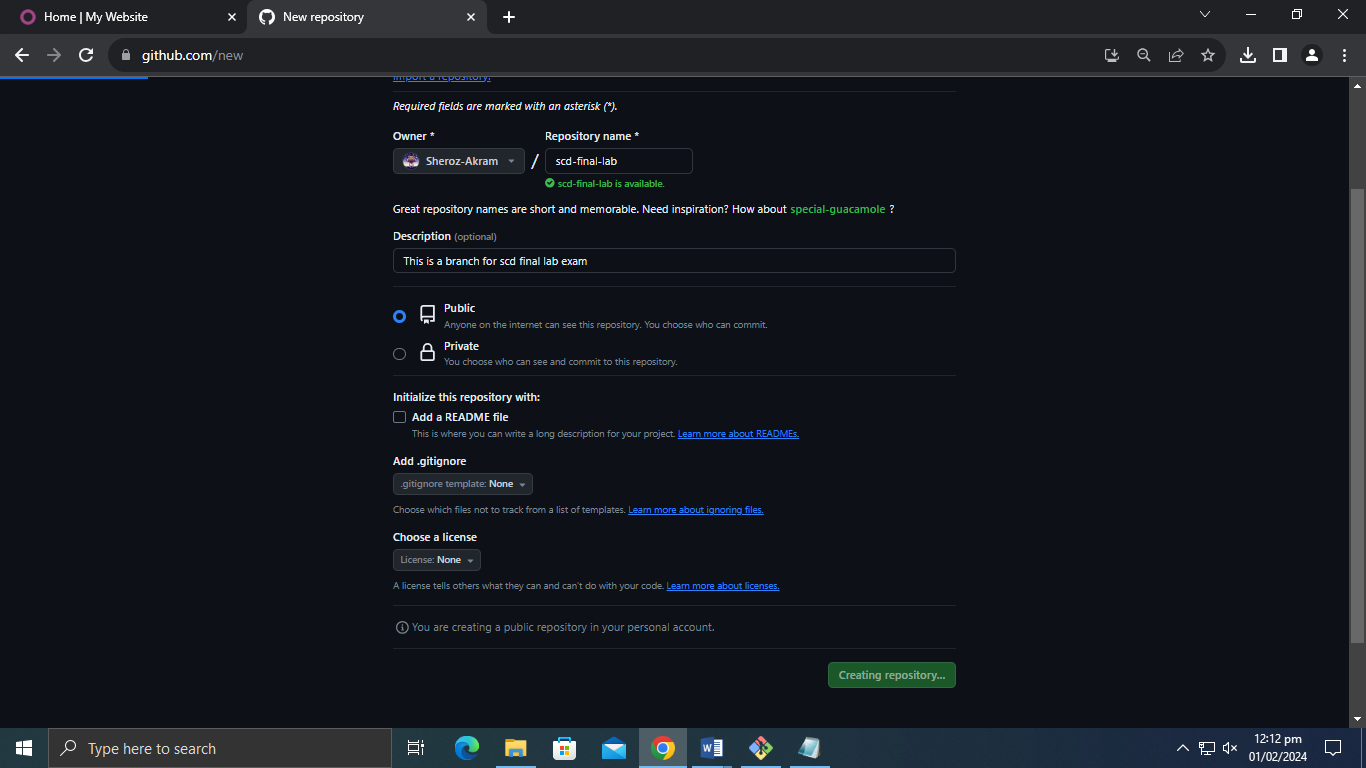


Now we resolve the conflicts manually by fix the class

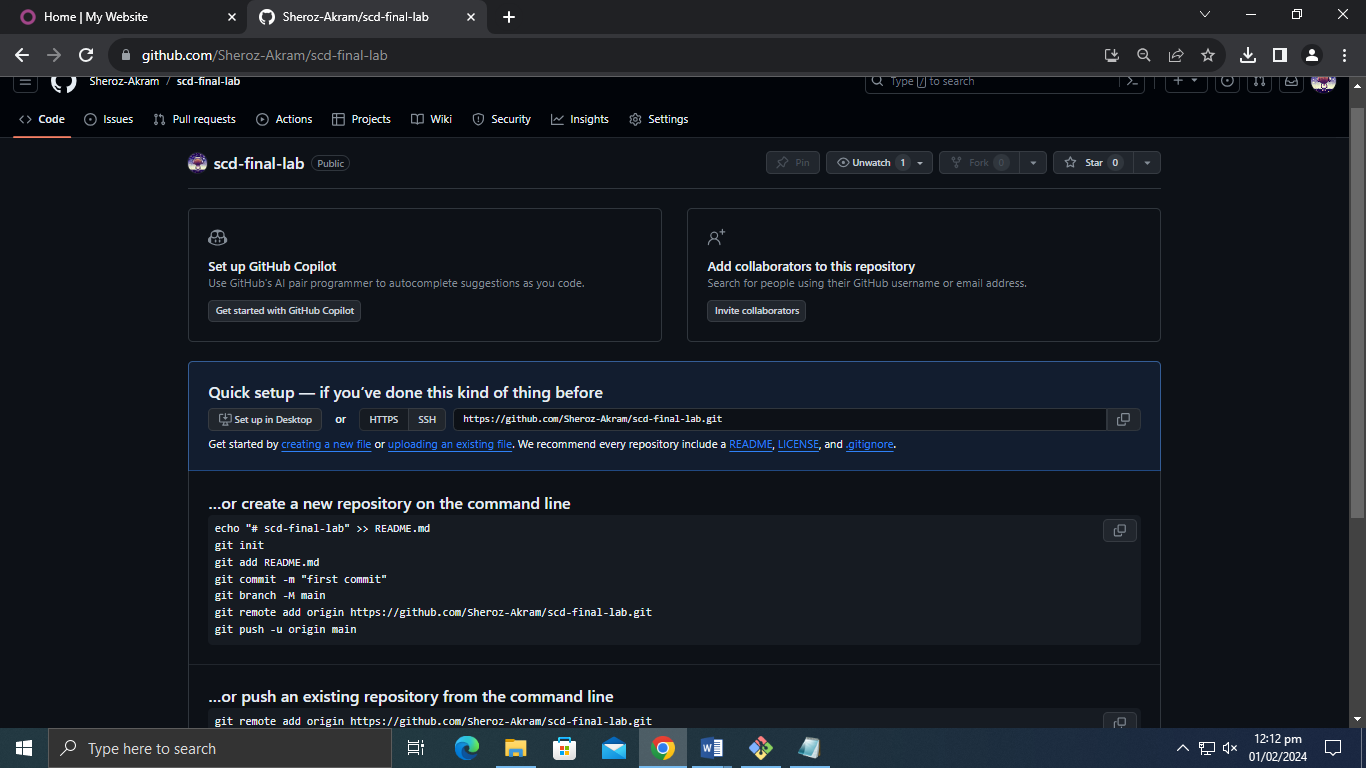


7. Create the Remote and Local repository. Set the name of local default to main

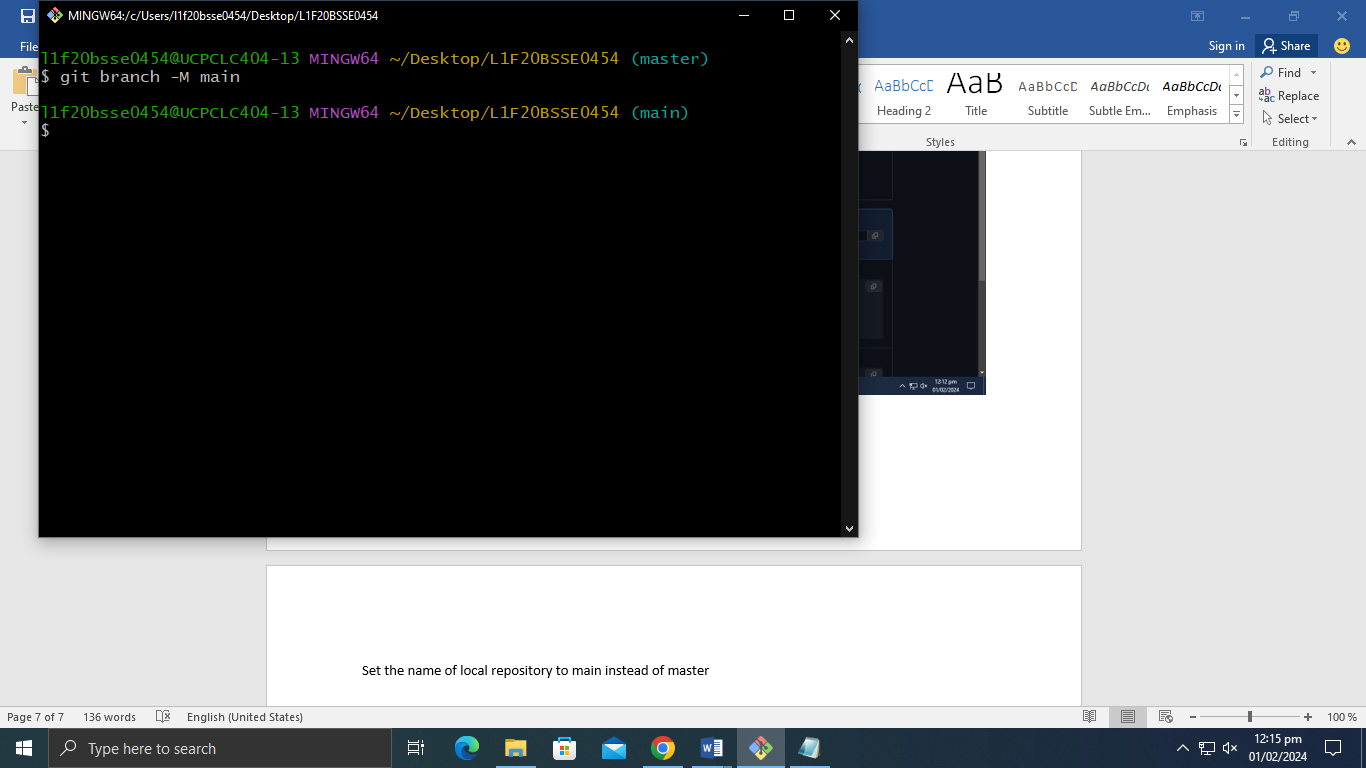
Create Remote Repository:



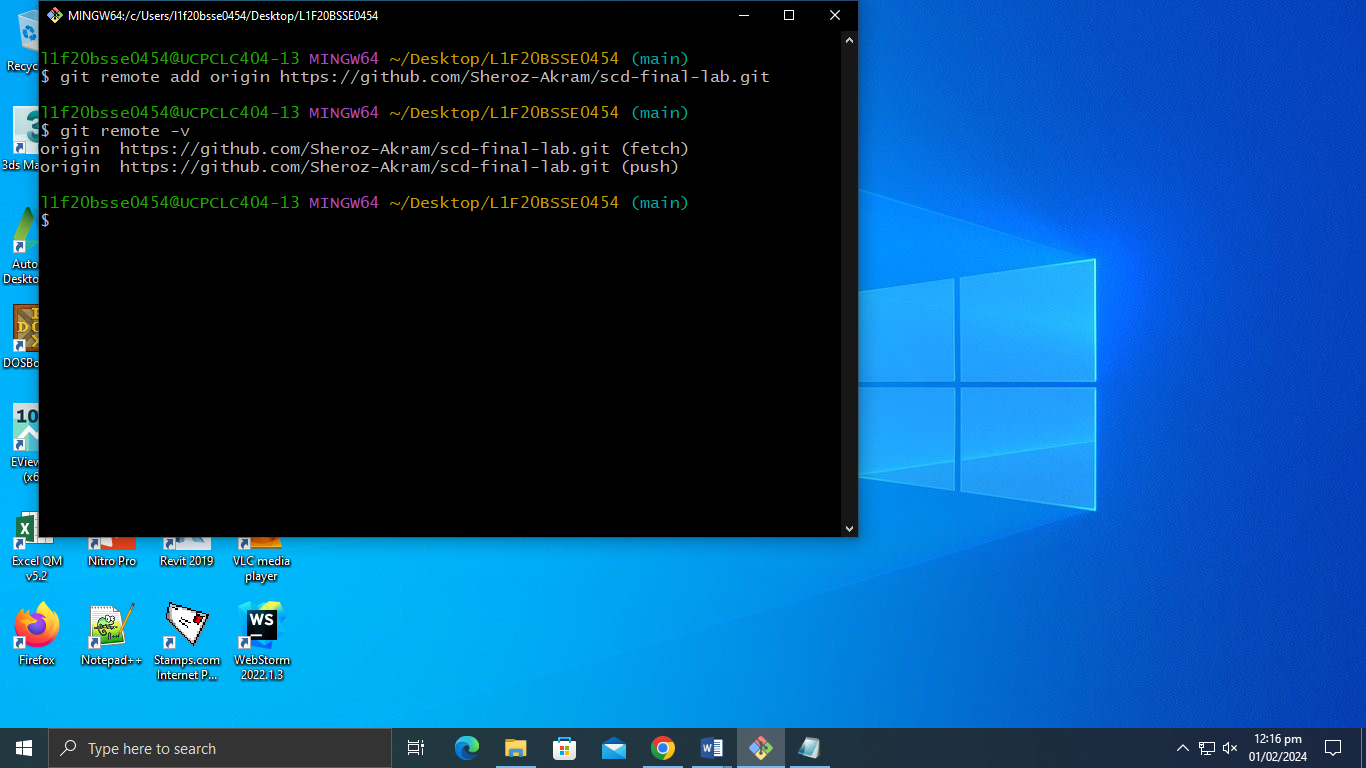
Remote repository is created



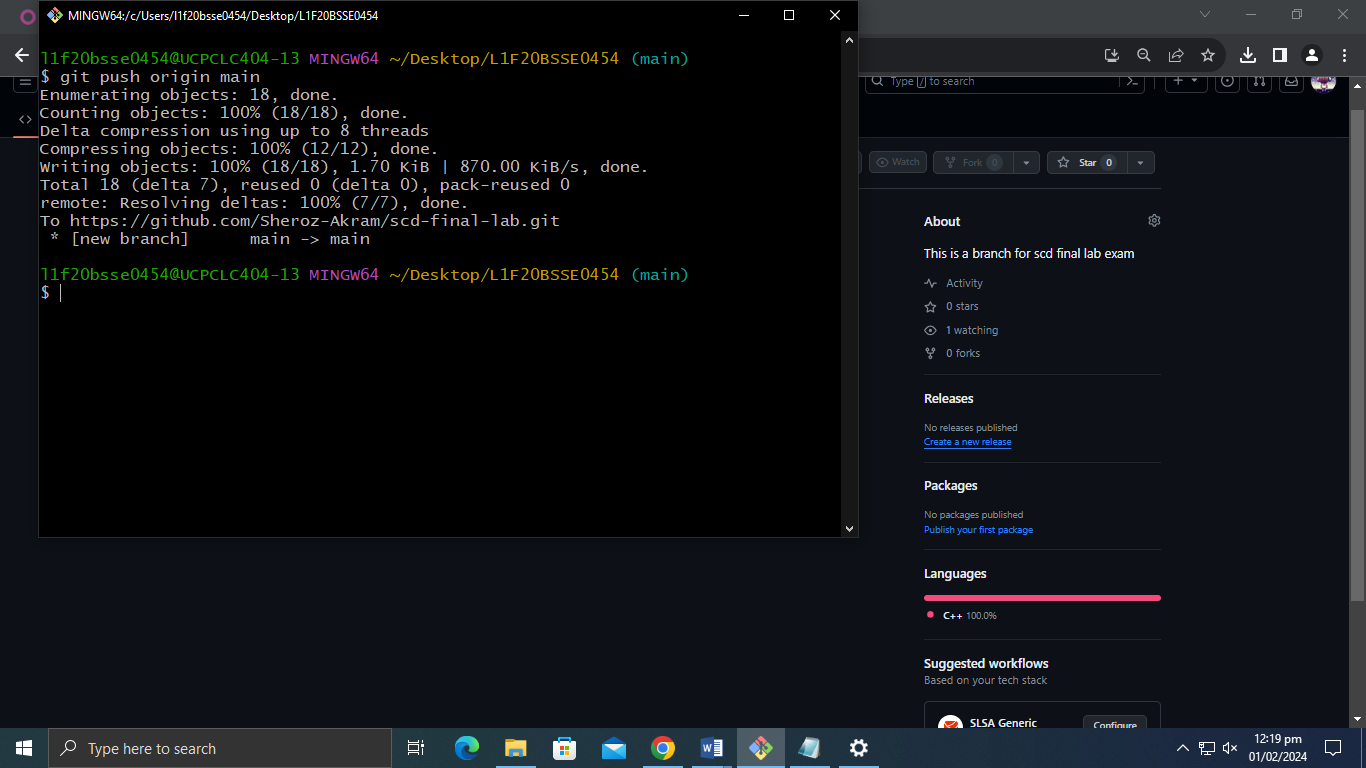
Set the name of local repository to main instead of master



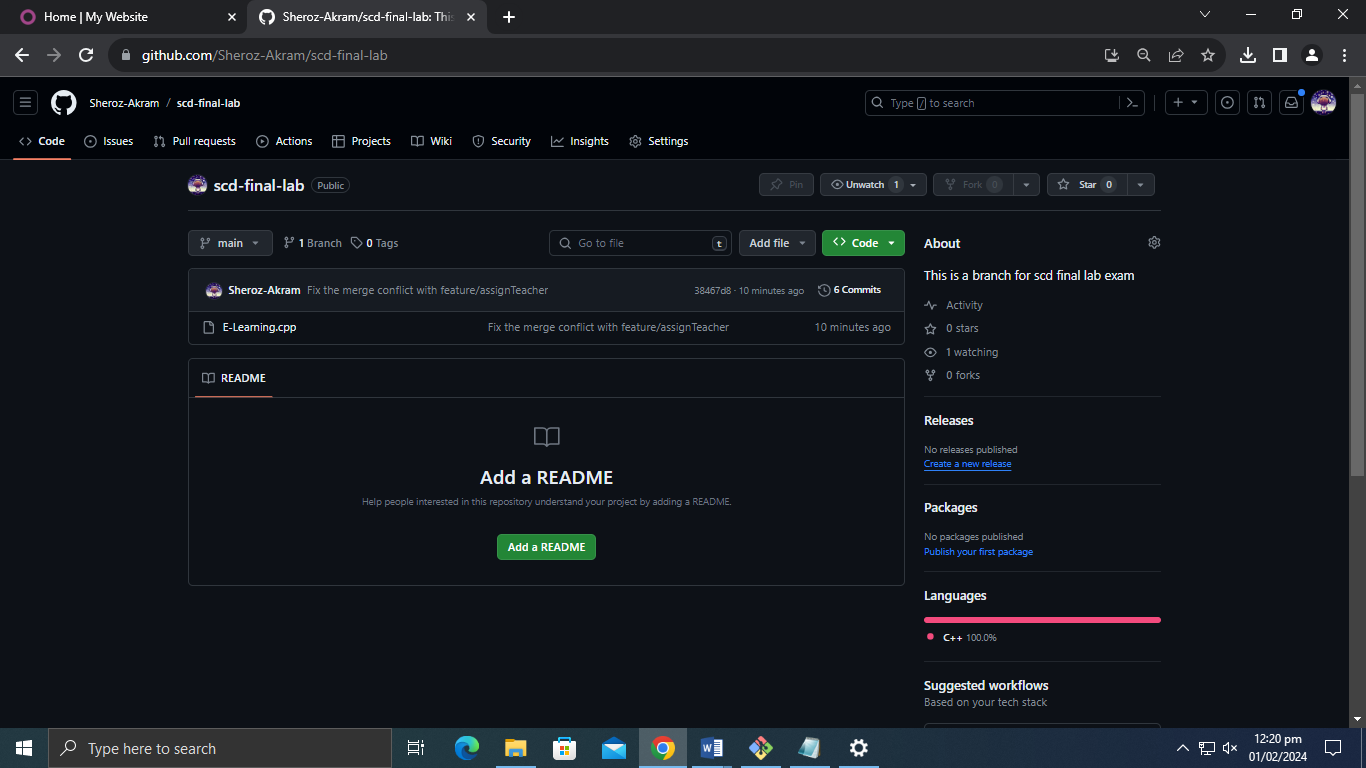
8. Make the connection between remote and local repository



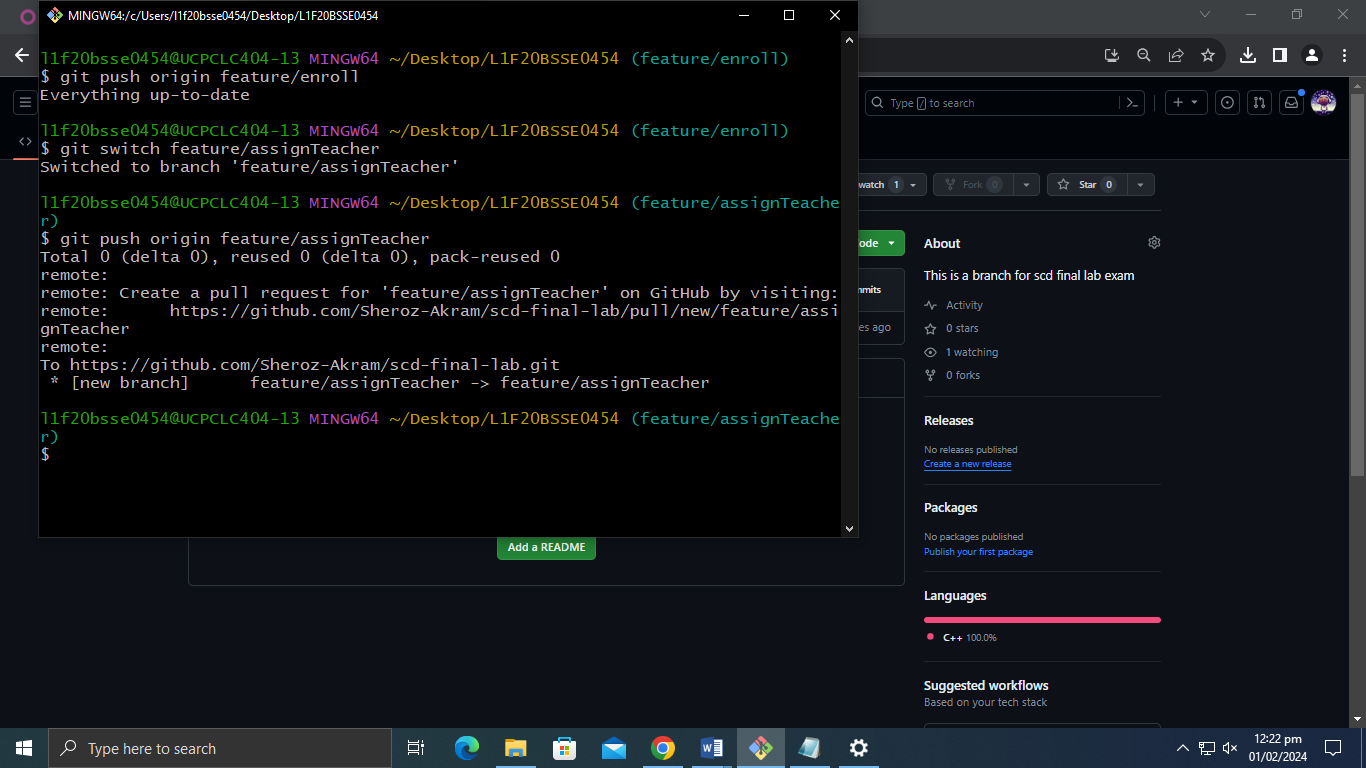
9. Push local repository to the remote repository



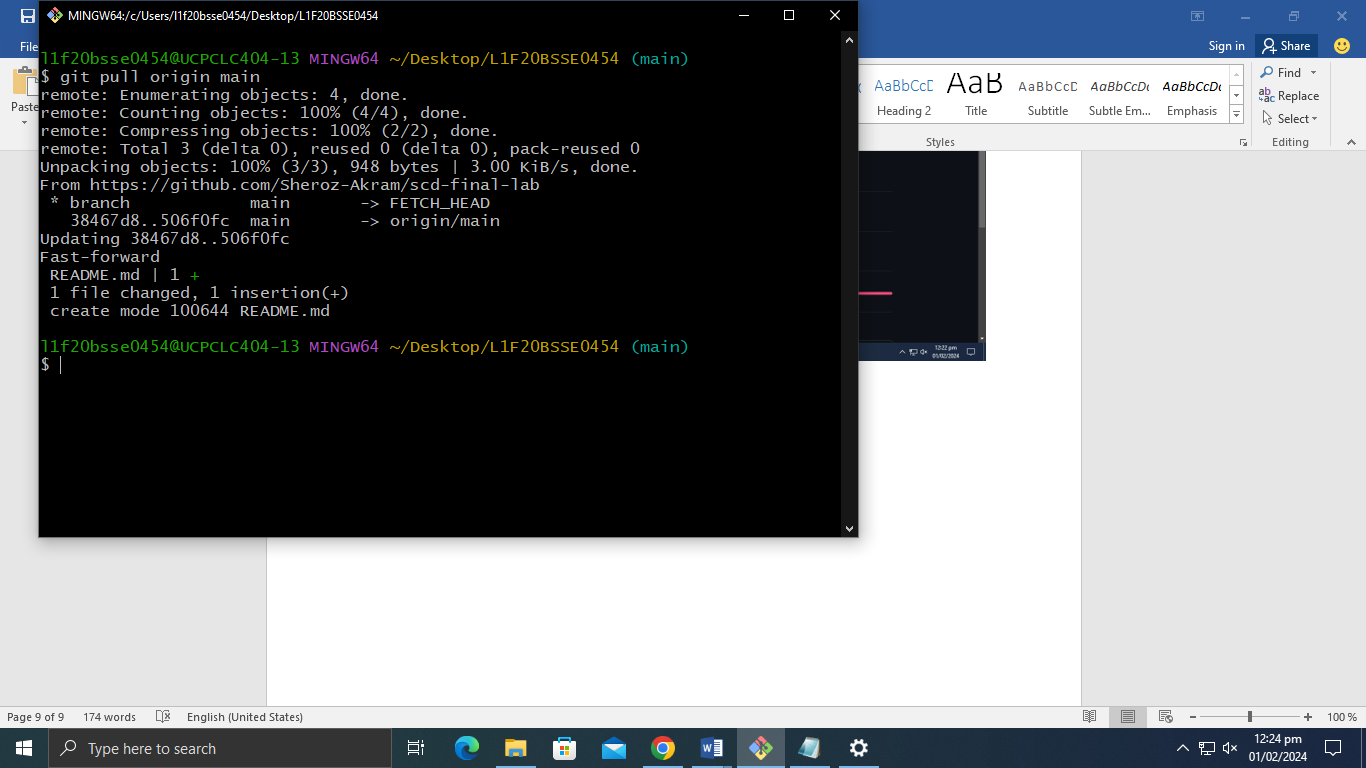
Changes has been made to the remote repository



10. Push the feature to the remote repository



11. Pull the remote repository



**Question # 02:**

**Answer**

**Part A:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Smell Name** | **Line No** | **Description** |
| 1 | Long Parameter List | 19 | The add product function in the inventory class have a very long parameter list which is a code smell and can create confusion among developers |
| 2 | Primitive Obsession | 7 to 9 | The product class uses primitive data types to store the information of the product instead we have to create domain specific class. |
| 3 | Mysterious Name | 10 | The constructor in the product class uses mysterious name which makes it harder to understand what variable is for what so we have rename it to meaning full names |
| 4 | Large Class | 15 to 28 | The inventory class is very large and violates the SRP principle. It stores the information of items and also display them. Instead to have to create sub class. |
| 5 | Feature Envy | 15 to 28 | The inventory class will be changed too frequently for even a single change we have to made to the whole program. |

**Part B:**

**1. Long Parameter List:**

Here is the refactored code for that smell in line 19. We have to pass the object instead of a long parameter list which was previous used. We pass the Product instead of name, quantity and price.

|  |
| --- |
| void addProduct(Product newProduct){  products.push\_back(newProduct)  } |

**2. Primitive Obsession:**

From the line 7 to 9 we use the primitive data types to store the information but instead we have to create domain specific class to store the data. Here is the refactored code for the code smell.

|  |
| --- |
| class ItemCost {  public:  int quantity;  double price;  }  class Product {  public:  string name;  ItemCost itemTotalCost;  } |

**3. Mysterious Names:**

In line 10 we use the mysterious names for the constructor which name it harder to understand what variable if for what. We have to rename the variables and give them meaningful names which makes sense to the developer.

|  |
| --- |
| Product(string productName, int productQuantiry, double productPrice): name(productName), quantity(productQuantity), price(productPrice); |

**4. Large Class:**

The inventory class from line 15 to 28 is very large and violates the SRP principle. This is a god class and we have to extract a new class to fix this code smell

|  |
| --- |
| class InventoryStore{  private:  vector<Product> products;  public:  void addProduct(Product newProduct){  products.push\_back(newProduct);  }  }  class DisplayInventor{ public:  void display\_inventory(vector<Product> \* products){  cout << “Inventory:\n”;  for(auto &product : products){  product.displayProduct();  }  }  } |

**5. Feature Envy:**

We use the same method as we used for the large class which is to extract class. This class also voidlates our SRP principle. This will separate the concerns of the Inventory class to different classes and does remove this smell feature envy.

Here is the complete refactored code:

|  |
| --- |
| #include <iostream>  #include <vector>  #include <string>  using namespace std;  class ItemCost {  public:  int quantity;  double price;  void display(){  cout << ", Quantity:" << quantity << ", Price: " << price << endl;  }  }  class Product {  public:  string name;  ItemCost itemTotalCost;  Product(string name, ItemCost itemCost): name(name), itemTotalCost(itemCost) {}  void displayProduct(){  cout << "Name: " << name;  itemTotalCost.display();  }  }  class InventoryStore{  public:  vector<Product> products;  void addProduct(Product newProduct){  products.push\_back(newProduct);  }  }  class DisplayInventory{  public:  void display\_inventory(vector<Product> \* products){  cout << “Inventory:\n”;  for(auto &product : products){  product.displayProduct();  }  }  }  int main(){  Inventory inventory;  Product p1 = Product("item1, ItemCost(10, 5.99));  Product p2 = Product("item2, ItemCost(20, 10.99));  Product p3 = Product("item3, ItemCost(5, 15.99));  inventory.addProduct(p1);  inventory.addProduct(p2);  inventory.addProduct(p3);    DisplayInventory displayInv = DisplayInventory();  displayInv.display\_inventory(inventory.products);  } |