食堂反馈系统的实现

1 前言

在这篇报告中，我将分三方面介绍本次食堂反馈系统的实现过程以及所新功能的添加的原因和实现。登入界面部分讲述在登入界面的设计过程和实现，具体部分的实现部分讲述了该系统核心部分的实现过程，最后是新功能部分，主要介绍该系统的有别于要求的新功能。讲述以思路 + 实现的方式实现。

该食堂反馈系统是以 Qt 为框架实现的，Qt 中的控件，信号和槽功能强大，使用灵活，使得该系统摆脱了黑窗口，实际操作更加方便于使用者。

2 登入界面的设计：

登入界面的实现主要集中在 LogOnWidget 这个类中。在我的代码中，LogOnWidget 是主要窗口 MainWindow 的一个子控件。

2.1 page1

由于有两种登入的属性：管理员和成员，所以我使用 Qt 中的 radiobutton 和 setChecked(true)函数来给用户选择自己的身份并设置默认选项。为了使界面更加美观，我在 MainWindow 中的绘画事件 paintevent 中对背景进行了美化，加上了背景图片和一些艺术字。功能方面，使用 lineedit 来接收使用者输入的信息。利用信号和槽的机制，lambda 表达式和 stackwidget 的功能实现页面的跳转进入 page2。当然，会更具 radiobutton 的选择进入不同的登入界面。

以下是换页操作代码和页面：

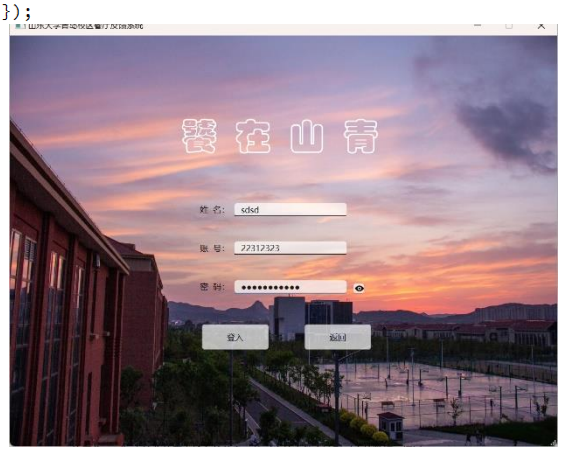
```
//换页继续操作()后续需要将不同身份带入不同的登入界面。
connect(ui->toolButtongoon,&QPushButton::clicked,this,[=](){
    if(ui->radioButton->isChecked()==1){
        AccountState=0;
        ui->stackedWidget->setCurrentIndex(1);
    }
    else {
        AccountState=1;
        ui->stackedWidget->setCurrentIndex(2);
    }

});
```

2.2 page2

使用 lineedit 来接收使用者输入的信息，从对应的文件中读取对应的数据，通过分割，将输入的内容和接收的内容相对应，如果不匹配则会弹窗提示。若成功对应则进入核心部分。

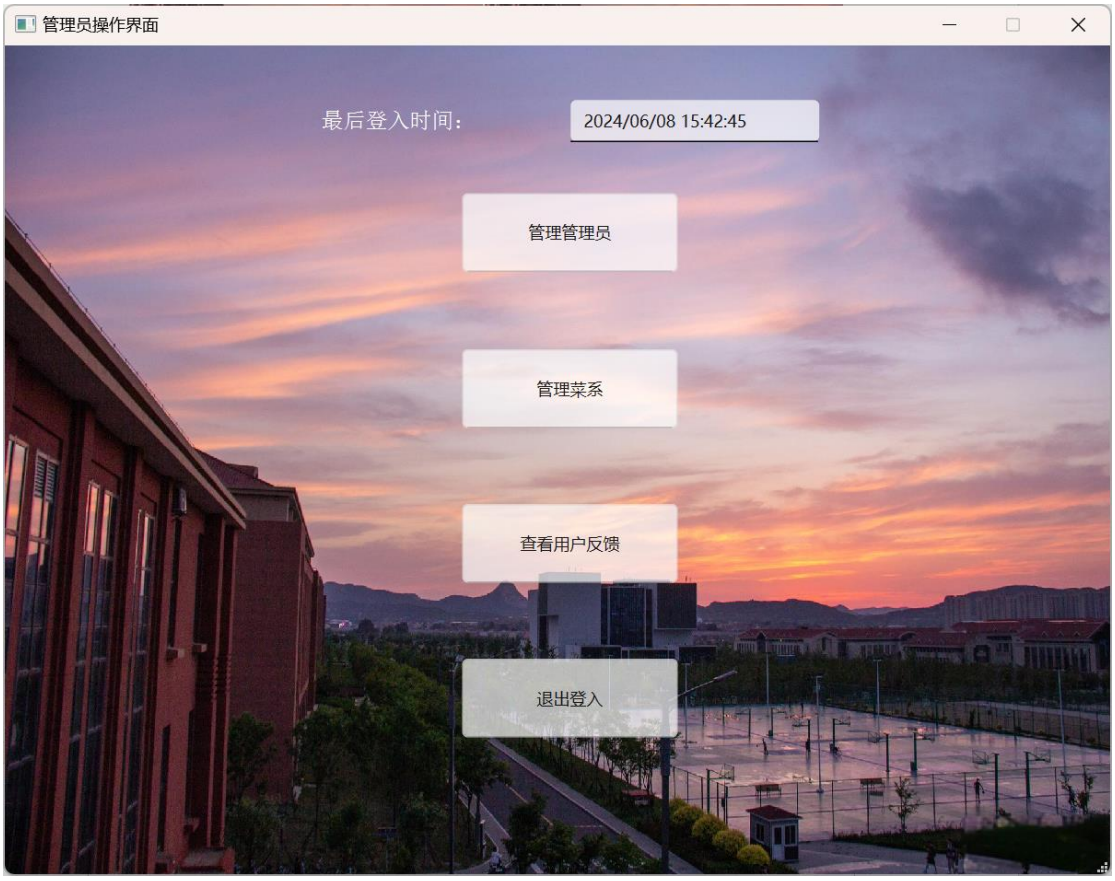
以下是登入判断的部分代码和两种不同页面：

```
//判断是否能够登录？继续按键
connect(ui->pushButtongoon,&QPushButton::clicked,this,[=](){
    QFile file;
    file.setFileName("./admin.txt");
    file.open(QIODevice::ReadOnly);
    QByteArray barrAccout,barrKey;
    while(!file.atEnd()){
        //创建临时变量储存该行的信息
        QByteArray temp=file.readLine().trimmed();
        barrAccout=temp.split(' ')[0];//依照' '分成两部分，如果两部分都匹配则返回true
        barrKey=temp.split(' ')[1];
        if(ui->lineEditname->text()==QString(barrAccout)&&ui->lineEditkey->text()==QString(barrKey)){
            //通过创建新对象来实例化新窗口。
            QTimer::singleShot(100,this,[=](){
                emit mainwindowhide();
                maWin->show();
            });
            return;
        }
    };
    //登入失败清空用户名，密码栏
    if(!ui->lineEditkey->text().isEmpty()&&!ui->lineEditname->text().isEmpty()){
        QMessageBox::information(this,"登入失败","账户或密码错误");
        ui->lineEditkey->clear();ui->lineEditname->clear();
        return;
    }
    if(ui->lineEditname->text().isEmpty()){
        QMessageBox::information(this,"登入失败","请输入用户名");
        return;
    }
    if(ui->lineEditkey->text().isEmpty())QMessageBox::information(this,"登入失败","请输入密码");
});
```



3 具体部分的实现

在具体实现的部分中，我使用了两种策略编写代码，在管理员中，由于很多函数没有复用，所以使用槽函数的 lambda 表达式来实现了，所以 managerwidget 中的代码有点长，而在 dishes 中，我将功能封装进了函数中。



3.1 管理员部分

首先描述一下我的思路：由于管理员的信息在文档中储存，我认为在登入之后，所有的管理员都具有了相同的权限，也就没有必要特地在 Administer 中设置类似账号密码的成员了。所以我将 Administer 类看做一个类代表着所有管理员，其中掌握着静态成员 NumOfAdmin 和静态成员函数 GetNumOfAdmin。另外，其他功能的实现主要放在 managerWidget 类中实现。

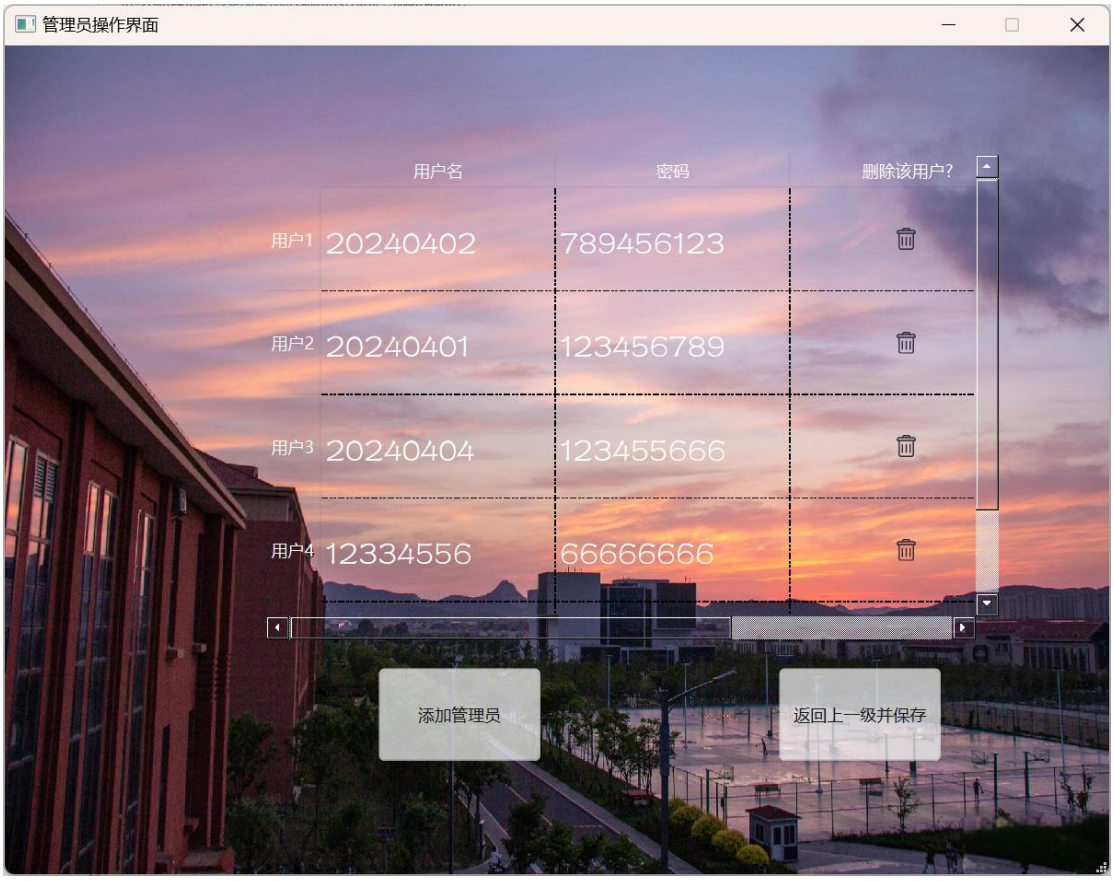
3.1.1 管理管理员界面

我认为添加管理员，删除管理员，修改管理员的密码这三个功能没有必要使用三个函数来实现，将三个功能整合成一个界面更加方便，也省去了输入需要被操作管理员信息的时间。所以使用 Qt 中 Tablewidget 来实现，添加删除图标点击即可实现删

除功能，点击新增管理员即可添加新的一行以供编辑。

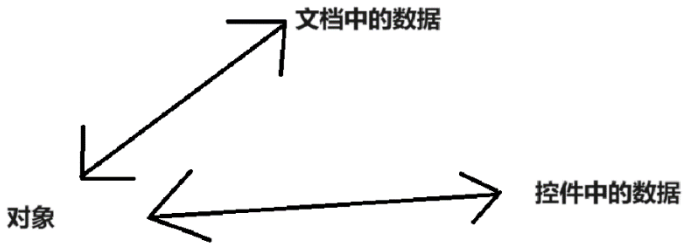
为了实现以上功能，编写的函数是每次点击管理管理员的按钮就从文档中重新读取内容，这样可以保证信息是更新的。同时根据将读到的内容填充至表格当中，而在每一次返回上一级的时候也将完成文档的更新，这样一来一回实现了数据的不丢失。

```
//Page1管理功能（）
//管理界面(每一次打开更新最新管理员信息)
connect(ui->pushButtonmana,&QPushButton::clicked,this,[=](){
    ui->stackedWidget->setCurrentIndex(1);
    //Page2其他操作
    QStringList HorHeader,VerHeader;
    HorHeader<<"用户名"<<"密码"<<"删除该用户?";
    for(int i=1;i<=Administrator::numofAdmin;i++)VerHeader<<QString("用户%1").arg(i);
    ui->tableWidget->setColumnCount(3);
    ui->tableWidget->setRowCount(Administrator::numofAdmin);
    ui->tableWidget->setHorizontalHeaderLabels(HorHeader);  ui->tableWidget->setVerticalHeaderLabels(VerHeader);

    //将文件的数据显示到屏幕上,并实现删除该行的功能
    QFile file("./admin.txt");
    file.open(QIODevice::ReadOnly);
    for(int i=0;i<Administrator::numofAdmin;i++){
        QByteArray temp=file.readLine().trimmed();
        for(int j=0;j<=1;j++){
            //添加到了控件中，就会随着控件消亡而消亡，无需手动释放。
            QTableWidgetItem *item = new QTableWidgetItem(temp.split(' ')[j]);
            ui->tableWidget->setItem(i,j,item);
        }
        //创建删除按键
        founddeleteIcon(ui->tableWidget,i);
    }
    file.close();
    for(int k=0;k<Administrator::numofAdmin;k++)ui->tableWidget->item(k,0)->setFlags(Qt::ItemFlags(~Qt::ItemIsEditable));
```



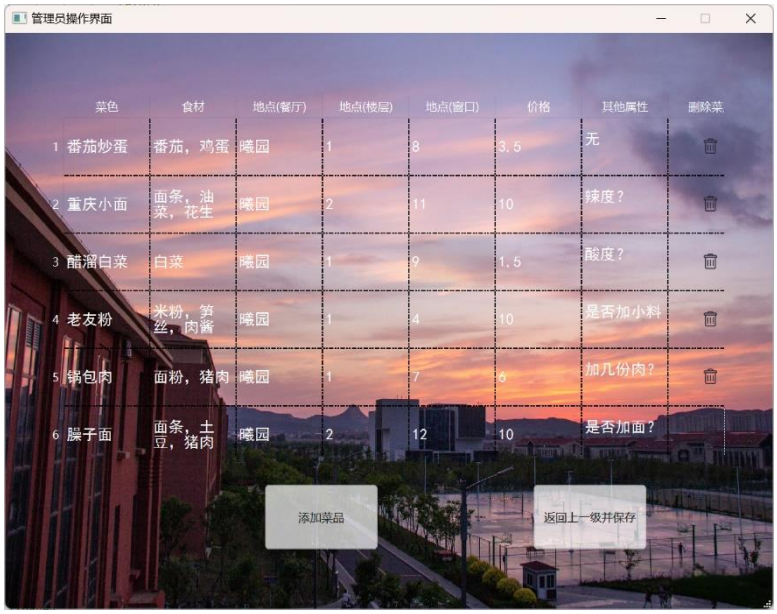
3.1.2 管理菜色界面

菜色部分由于需要实现由文本到控件上数据的转换的问题，在两者间加入了对象便于两者的转换于是有了多个转化函数的编写(这将在菜色部分具体讲解)。



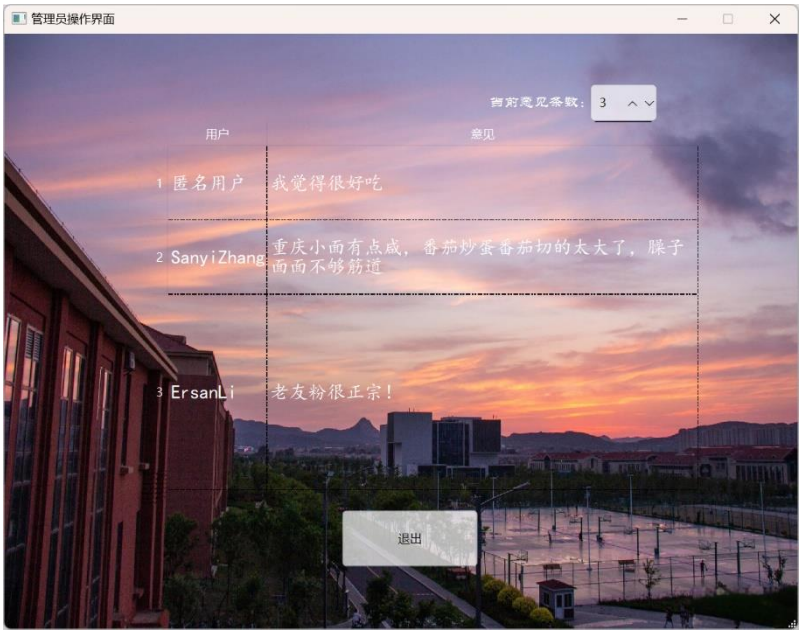
在新建菜品部分与新建联系人是相同的思路，删除、文件保存也是同理。

```
//page3操作菜色方面进行封装操作
connect(ui->pushButtondishes,&QPushButton::clicked,this,[=](){
    ui->stackedWidget->setCurrentIndex(2);
    ui->widgetmenu->ReFreshMenu(0);
});

//添加新一行以编辑新菜品
connect(ui->pushButtonsnewdish,&QPushButton::clicked,this,[=](){
    ui->widgetmenu->adddish();
});
//退出操作并保存
connect(ui->pushButtonbacksave_2,&QPushButton::clicked,this,[=](){
    if(ui->widgetmenu->ifEmpty()){
        QMessageBox::critical(this,"警告","存在菜品信息为空，请完善所有信息！");
        return;
    }
    else{
        ui->widgetmenu->RefreshFile();
        ui->stackedWidget->setCurrentIndex(0);
    }
});
```



具体效果如左图。

3.1.3 管理评论界面



管理评论就比较简单实现了，从文件中按照在成员部分输入的格式来进行读取操作即可(输入格式的控制在下文介绍)，

右图就是实际效果

最后还有退出登入的实现，相对简单就不详细介绍了。

3.2 菜品部分(这是其他部分的补充)

菜品部分代码主要在 dishes 类和 dishwidget 类中实现：

dishes 类中有许多成员与成员函数，包括多种组成部分。创建菜品类的目的：在管理员中我们介绍了需要菜品类作为文件和控件信息的转换的中介。

dishes 类中有字符串：菜名，其他属性；结构体：place 其有三个字符串成员，和构造函数；浮点型：价格；

与管理员类相同，我们也赋予了 dishes 静态变量 NumofDishes 和静态成员函数 GetNumOfDishes()来方便后续的操作。

```
void dishes::FromObjectToFile(){
    QFile file("./dishes.txt");
    file.open(QIODevice::Append);
    file.write(DishName.toUtf8());
    file.write(" ");
    while(Ingredients.size()>1){
        file.write(Ingredients.front().toUtf8());
        file.write(",");
        Ingredients.pop_front();
    }file.write(Ingredients.front().toUtf8());Ingredients.pop_front();file.write(" ");
    file.write(Place.restaurant.toUtf8()); file.write(",");file.write(Place.floors.toUtf8()); file.write(",");file.write(Place.windows.toUtf8());file.write(" ");
    file.write(QByteArray::number(Price));file.write(" ");
    file.write(OtherAttributes.toUtf8());
    file.close();
}
```

```
deque<dishes>* dishes::FromFileToObject(){
    deque<dishes>* d1=new deque<dishes>;
    QFile file("./dishes.txt");
    file.open(QIODevice::ReadOnly);
    QByteArray bytearrya;
    for(int i=1;i<=NumOfDishes;i++){
        bytearrya=file.readLine();
        dishes dish;

        dish.DishName=bytearrya.split(' ')[0];

        int temp=0;
        while(bytearrya.split(' ')[1].split(',').size()>temp)dish.Ingredients<<bytearrya.split(' ')[1].split(',')[temp++];

        dish.Place.restaurant=bytearrya.split(' ')[2].split(',')[0];
        dish.Place.floors=bytearrya.split(' ')[2].split(',')[1];
        dish.Place.windows=bytearrya.split(' ')[2].split(',')[2];

        dish.Price=bytearrya.split(' ')[3].toDouble();

        dish.OtherAttributes=bytearrya.split(' ')[4];
        if(dish.OtherAttributes[dish.OtherAttributes.size()-1]!='\n')dish.OtherAttributes.append('\n');
        d1->push_back(dish);
    }
    file.close();
    return d1;
}
```

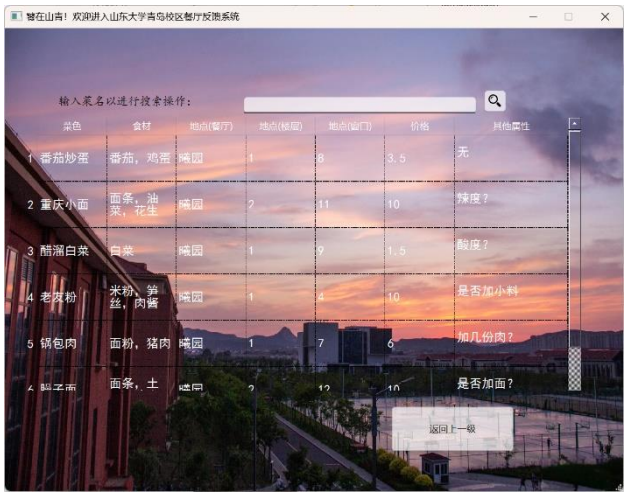
3.3 成员部分

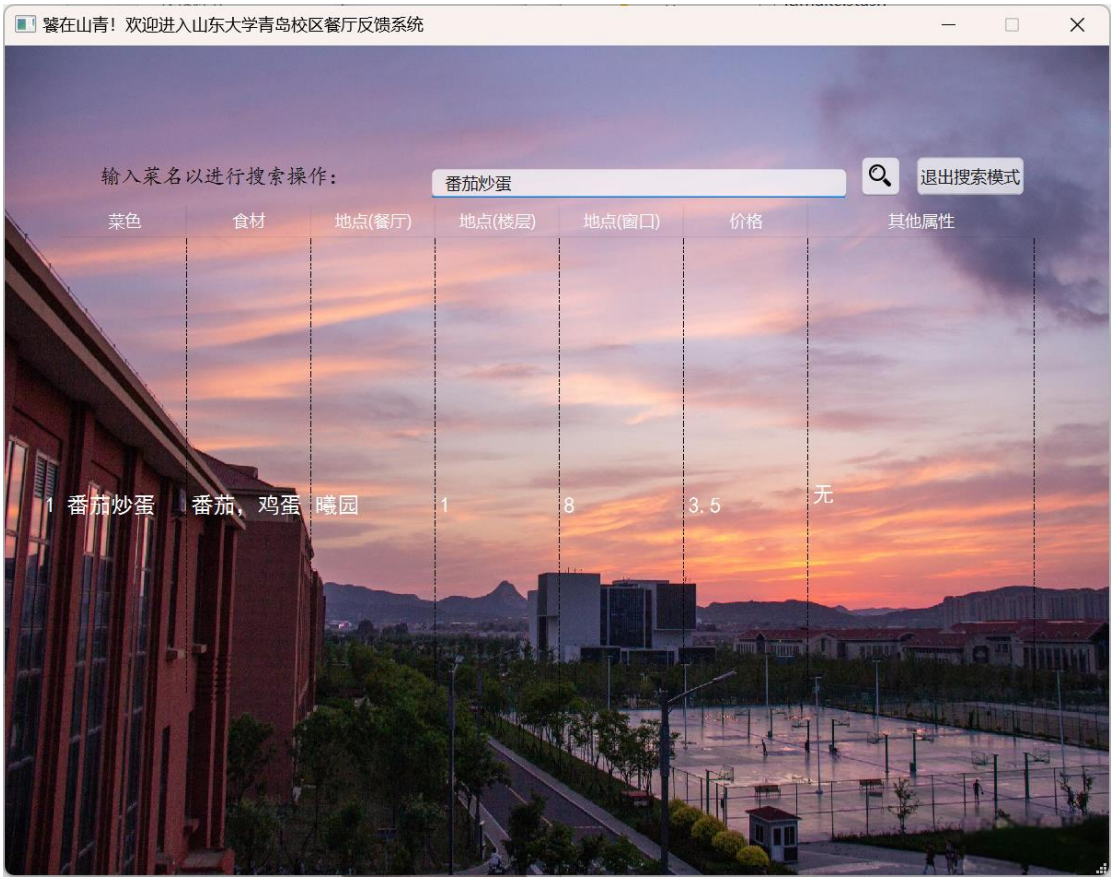
成员部分我的想法大体与设计管理员类时相同，有所不同的是我想记录一下当前登入的姓名以满足后面新增部分的需求。

3.3.1 查看菜系

首先我想先通过上面 dishes 的函数实现将文本转化成 Tablewidget 上的 item，至于搜索菜系引入一个编辑框来储存输入的内容，将内容与文本内的第一个部分进行比较，若找到则刷新 Tablewidget，仅将找到的部分显示在屏幕上。但是由于我们是通过退出按键进行保存，所以在进入搜索模式时，需要将该按钮隐藏，等待退出后再进行显示并重新将所有菜品输出到表格内。

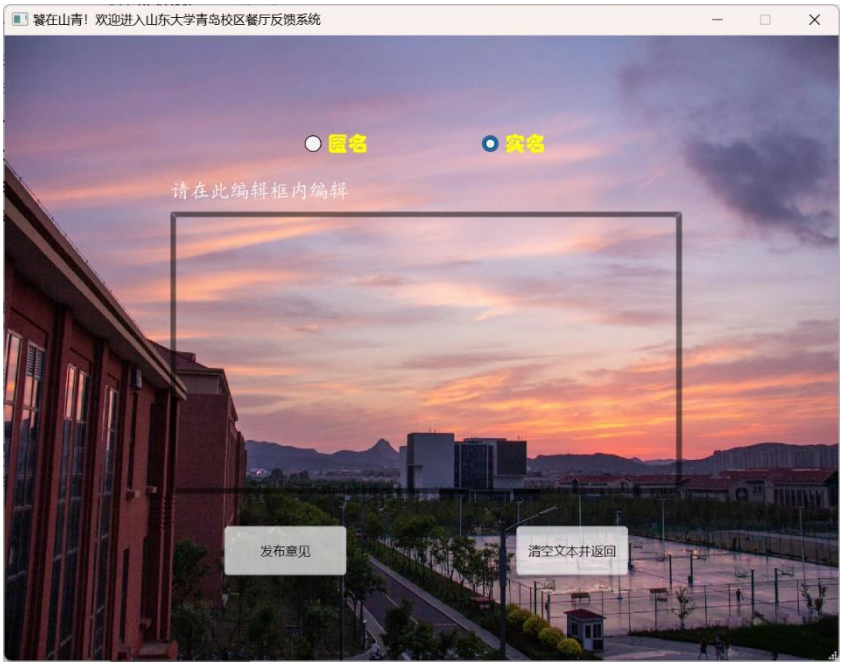
```
//查看菜系
connect(ui->pushButtonsearch,&QPushButton::clicked,this,[=](){
    ui->stackedWidgetusers->setCurrentIndex(1);
    ui->widgetmenu_2->ReFreshMenu(1);
});

connect(ui->pushButtonusersback,&QPushButton::clicked,this,[=](){
    ui->stackedWidgetusers->setCurrentIndex(0);
});

//建立搜索模式移除保存按键的联系
connect(ui->widgetmenu_2,&DishWidget::EnterSearchMode,this,[=]{
    ui->pushButtonusersback->hide();
});

connect(ui->widgetmenu_2,&DishWidget::OutSearchMode,this,[=](){
    ui->pushButtonusersback->show();
});
```





3.3.2 提交意见

提交意见部分的实现就比较简单，只需将编辑框内的部分输入到文档中即可，只需要注意输入的格式，我选择的是在评论和评论间用##COMMENT##字符串进行隔开，姓名和文本用##TEXT##进行分隔来进行区分。



4 添加的新功能部分和优化部分

1 登入界面实现点击释放实现密码可见性的切换

利用两个 lambda 表达式

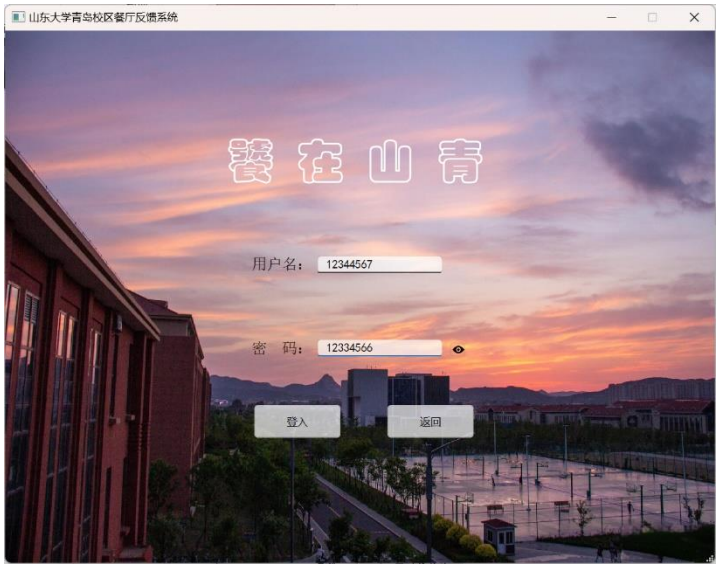
按下时将格式设置为可见，

松开时将格式设置为密码模

式即可实现这种比较常见的

密码输入形式，保证隐私同

时更实用




```
//按下，释放就改变模式。
connect(ui->toolButtoneye,&QPushButton::pressed,this,[=](){
    ui->lineEditkey->setEchoMode(QLineEdit::Normal);
});
connect(ui->toolButtoneye,&QPushButton::released,this,[=](){
    ui->lineEditkey->setEchoMode(QLineEdit::Password);
});
connect(ui->toolButtoneye_2,&QPushButton::pressed,this,[=](){
    ui->lineEditkey_2->setEchoMode(QLineEdit::Normal);
});
connect(ui->toolButtoneye_2,&QPushButton::released,this,[=](){
    ui->lineEditkey_2->setEchoMode(QLineEdit::Password);
});
```

2 管理员的操作更方便

管理员可以实现双击编辑框来编辑菜色，而在成员中是不会对菜色进行改变，使得管理员对菜色的管理更加方便。删除管理员只需要轻击图标，其功能已经被封装进函数中。

3 异常处理

对绝大多数异常情况进行了处理，使用 Messagebox 进行提醒并阻止异常情况。

以下为部分实例：

```
//登入失败清空用户名，密码栏
if(!ui->lineEditkey->text().isEmpty()&&!ui->lineEditname->text().isEmpty()){
    QMessageBox::information(this,"登入失败","账户或密码错误");
    ui->lineEditkey->clear();ui->lineEditname->clear();
    return;
}
if(ui->lineEditname->text().isEmpty()){
    QMessageBox::information(this,"登入失败","请输入用户名");
    return;
}
if(ui->lineEditkey->text().isEmpty())QMessageBox::information(this,"登入失败","请输入密码");
```

```
ManagerWindow::connect(toolButton,&QToolButton::clicked,tableWidget,[=](){
    if(Administrator::numofAdmin==1)QMessageBox::critical(tableWidget,"警告","至少应该有一个管理员",QMessageBox::Ok);
    else if(QMessageBox::Ok==QMessageBox::information(tableWidget,"提示：","确定删除该管理员吗",QMessageBox::Ok|QMessageBox::Close,QMessageBox::(
        int row = tableWidget->indexAt(toolButton->pos()).row();//利用pos获取Point
        tableWidget->removeRow(row); // 删除该行
        Administrator::numofAdmin--;
    }
});
```

4 引入了实名匿名机制。

在意见部分引入了实名匿名机制。并会及时显示当前意见的条数，这就是前面讲述的设置静态成员对象评论条数的原因。

```
void MemberWindow::fromEdittiFile(QRadioButton *radiobutton,QTextEdit * textedit){
    QFile file("./comments.txt");
    file.open(QIODevice::Append);
    file.write("##COMMENT##");
    if(radiobutton->isChecked()==1){
        file.write("匿名用户##TEXT##");
    }
    else{
        file.write(Member::NowLogonName.toUtf8());
        file.write("##TEXT##");
    }
    QString text=textedit->toPlainText();
    file.write(text.toUtf8());
    Member::NumOfComments++;
    file.close();
  ui->spinBox->setValue(Member::NumOfComments);
```

5 管理员最后登入时间的显示。

由于上面文件保存是每一次返回时都会刷新，我们可以记录文件最后一次修改的的时间实现告知管理员上一次登入实行操作的时间是什么，使得操作更加的安全，可

控制。

```
ui->lineEditTime->setText(QFileInfo("./admin.txt").lastModified().toString("yyyy/MM/dd hh:mm:ss"));
ui->lineEditTime->setEnabled(false);
```