**Создание приложения с использованием   
PyQt5 и SQLite**

Выполнил

Глушин Никита

Самара, 2019 г.

**1. Основные цели и задачи проекта**

Цель: создать приложение, применяя язык программирования Python, библиотеку PyQt5 и базу данных SQLite.

Задачи:

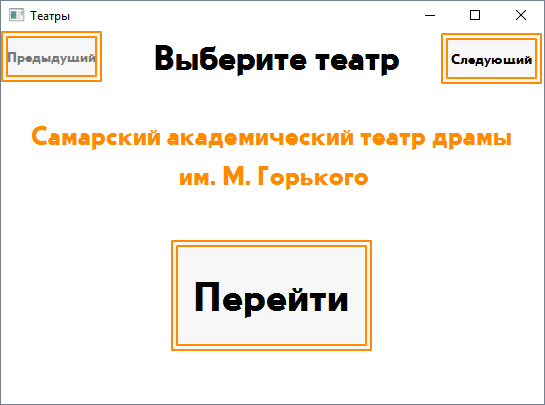
* Разработать идею
* Продумать архитектуру проекта
* Заполнить базу данных
* Написать программу
* Разработать внешний дизайн приложения

Идея: Приложение для покупки билетов в театр.

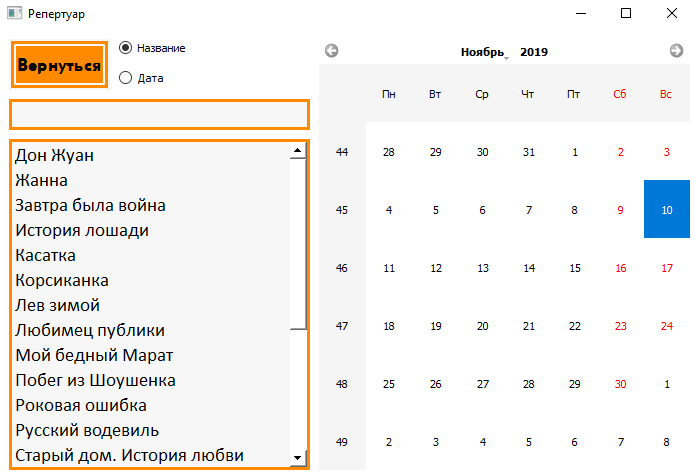
Описание: Мой проект представляет собой демонстрационную версию приложения по покупки билетов. В нем рассматривается только один театр, но приложение создано так, что для добавления театров не требуется больших усилий.

**2. Как работает приложение**

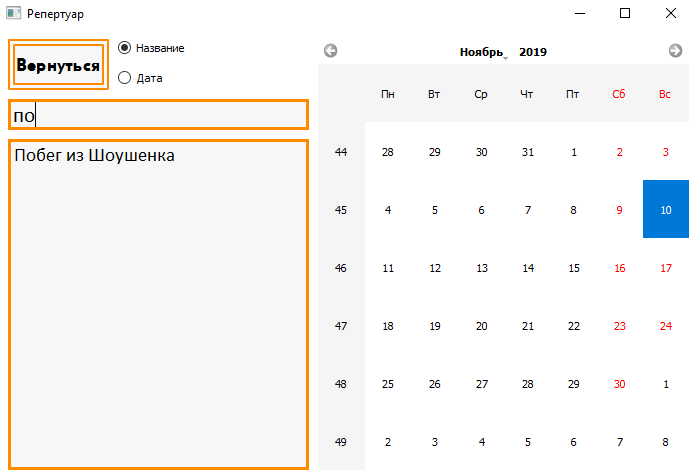
Сначала необходимо выбрать театр. При помощи кнопок «Предыдущий» и «Следующий» Вы можете листать страницы с театрами.

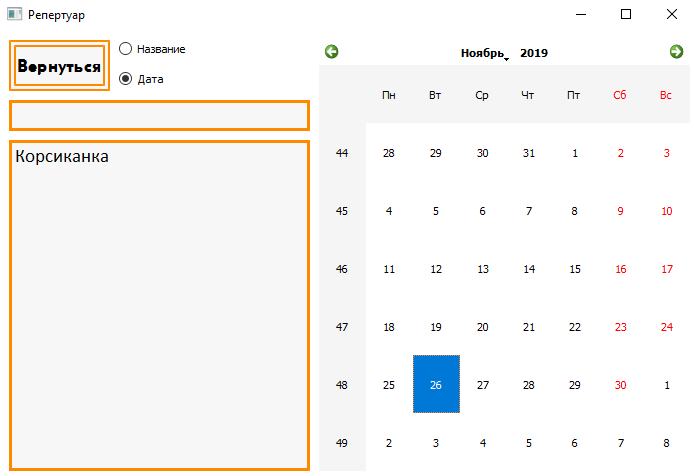


При нажатии на кнопку «Перейти» откроется окно с репертуаром театра.



Вы можете выбрать любой из спектаклей, также Вам доступен поиск по названию и по дате (необходимо выбрать один из пунктов).

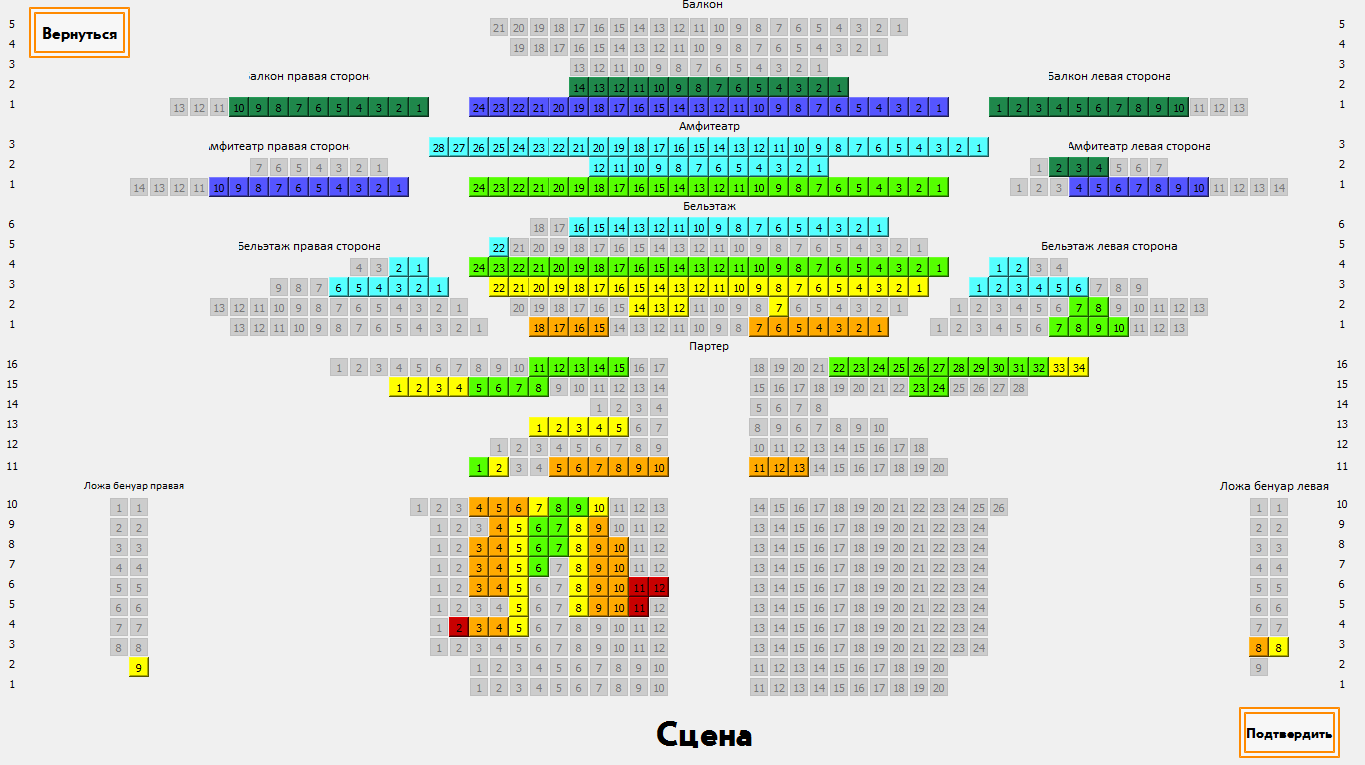




После выбора спектакля откроется соответствующее окно.

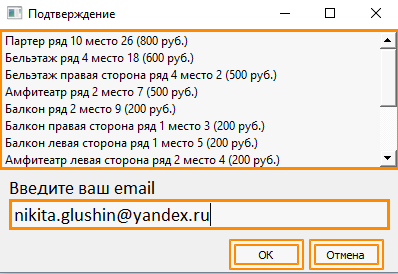


Здесь Вы можете прочитать информацию о спектакле, узнать режиссера и автора. Также Вы можете выбрать одну из доступных дат.

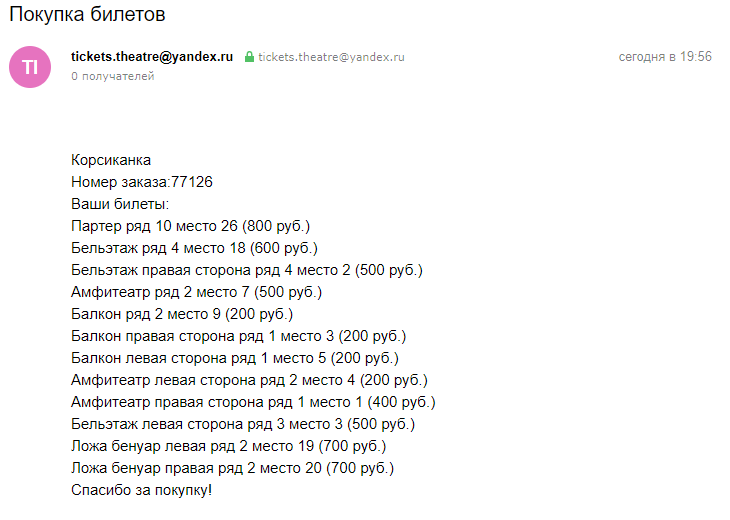


В окне со схемой зала Вам необходимо выбрать хотя бы одно место. При наведении на место Вы узнаете его стоимость, место и ряд.

Подтвердив выбор, откроется окно доставки.



Вам остается указать ваш email и нажать «ОК».



**3. Архитектура проекта и техническое описание**

Программа состоит из нескольких файлов. В файле main.py находится основная часть программы, где описаны классы форм. Accepting.py, play.py, repertoire.py, start\_page.py, theatre.py – файлы с классами дизайна форм. Файл import\_data.py используется для работы с базой данных.

Рассмотрим некоторые фрагменты программы более подробно.

**Repertoire form**

class Repertoire(QWidget, repertoire):

    def \_\_init\_\_(self, data\_base):

        super().\_\_init\_\_()

        self.setupUi(self)

        # импортирование БД

        self.data\_base = data\_base

        importer = Import\_export()

        self.result = importer.import\_table(data\_base, 'repertoire')

        self.d = dict()

        self.last\_line = ''

        # заполнение listWidget спектаклями

        for i in self.result:

            self.listWidget.addItem(i[1])

            self.d[i[1]] = i

self.listWidget.itemDoubleClicked.connect(self.push)

        self.back\_to\_start.clicked.connect(self.push\_back)

        self.rb\_name.toggled.connect(self.start\_search\_by\_name)

        self.rb\_date.toggled.connect(self.start\_search\_by\_date)

        self.search\_by\_name\_timer = QTimer()  # таймер для поиска по имени

        self.search\_by\_name\_timer.timeout.connect(self.search\_by\_name)

        self.search\_by\_date\_timer = QTimer()  # таймер для поиска по дате

        self.search\_by\_date\_timer.timeout.connect(self.search\_by\_date)

        self.back\_to\_start.installEventFilter(self)

        self.start\_search\_by\_name()

Для получения списка спектаклей мы создаем экземпляр класса Import\_export()

importer = Import\_export()

В self.result мы импортируем все спектакли из таблицы 'repertoire'

В self.d в качестве ключа мы сохраняем название спектакля, а в качестве значения все остальные необходимые данные.

На форме с репертуаром доступны два вида поиска: по имени и по дате. Чтобы сделать поиск в реальном времени, необходимо создать два экземпляра класса QTimer. За одинаковые промежутки времени они обновляют listWidget.

def search\_by\_name(self):

        # поиск по имени в реальном времени

        if not self.lineEdit.text() and self.lineEdit.text() != self.last\_line:

            self.listWidget.clear()

            self.d = dict()

            for i in self.result:

                self.listWidget.addItem(i[1])

                self.d[i[1]] = i

            self.last\_line = self.lineEdit.text()

        else:

            if self.last\_line != self.lineEdit.text():

                self.listWidget.clear()

                self.d = dict()

                for i in self.result:

                    if i[1].lower().find(self.lineEdit.text().lower()) == 0:

                        self.listWidget.addItem(i[1])

                        self.d[i[1]] = i

                self.last\_line = self.lineEdit.text()

def search\_by\_date(self):

        # поиск по дате в реальном времени

        date = self.calendarWidget.selectedDate()

        date = date.toString('dd.MM.yyyy')

        if self.last\_date != date:

            self.listWidget.clear()

            self.d = dict()

            con = sqlite3.connect('data/' + self.data\_base + '.db')

            cursor = con.cursor()

            tables = [i[1] for i in con.execute(

                "select \* from sqlite\_master where type = 'table'")]

            for i in tables:

                if date.replace('.', '') in i:

                    for j in self.result:

                        if j[1].lower().replace(' ', '') in i.lower():

                            self.listWidget.addItem(j[1])

                            self.d[j[1]] = j

        self.last\_date = date

**Play form**

Чтобы заполнить форму отдельного представления, мы получаем список всех спектаклей по датам и среди них ищем те, в которых встречается название выбранного.

con = sqlite3.connect('data/' + self.data\_base + '.db')

        cursor = con.cursor()

        # получение всех спектаклей по времени

        tables = [i[1] for i in con.execute(

            "select \* from sqlite\_master where type = 'table'")]

        selected\_item\_ = self.listWidget.selectedItems()[0].text()

        self.selected = self.d[selected\_item\_]

        selected\_item = ''

        # нахождение подходящих спектаклей

        for i in selected\_item\_.upper():

            if i in 'ЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮ':

                selected\_item += i

        self.dates = [i for i in tables if selected\_item.upper() in i]

        self.play\_form = Play\_form(self.selected, self.dates, self.data\_base)

Объявляем экземпляр класса Play\_form() , где в качестве аргументов передаем необходимые данные, возможные даты и базу данных. Далее заполняем форму.

        # импортирование картинки из БД

        with open('picture.jpg', 'wb') as f:

            f.write(selected[4])

        first\_img = Image.open('picture.jpg')

        # выставление размера изображения

        new\_img = first\_img.resize((771, 251))

        new\_img.save('picture.jpg')

        self.label.setPixmap(QPixmap('picture.jpg'))

        # запись всех доступных дат в ListWidget

        for i in dates:

            item = i[-12:]

            item = item[:2] + '.' + item[2:4] + '.' + \

                item[4:8] + ' ' + item[8:10] + ':' + item[10:]

            self.listWidget.addItem(item)

        self.director.setText(selected[3])  # импортирование режиссера

        self.author.setText(selected[6])  # импортирование автора

        self.plainTextEdit.setPlainText(selected[2])  # импортирование описания

        self.setWindowTitle(selected[1])

        self.name.setText(selected[1])  # импортирование названия

        global play\_name

        play\_name = selected[1]

**Theatre form**

# нахождение подходящей таблицы со спектаклем

        for i in tables:

            if selected\_item in i:

                if self.data\_base == 'theatre\_test':

                    self.theatre = Drama\_theatre\_form(self.data\_base, i)

                    gl\_theatre = self.theatre

                    self.theatre.showFullScreen()

В этом фрагменте мы находим нужную таблицу. Столбцами ней являются id, ряд, место и цена.

for i in range(len(result)):

            if result[i][3] == 0:

                self.buttons[i].setEnabled(False)

            else:

                # изменение цвета кнопки в зависимости от ее цены

                self.buttons[i].setToolTip(

                    result[i][1] + ' место ' +

                    str(result[i][2]) + '\n(' + str(result[i][3]) + ' руб.)')

                if result[i][3] == 900:

                    self.buttons[i].setStyleSheet(

                        'background-color: rgb(200, 0, 0)')

                elif result[i][3] == 800:

                    self.buttons[i].setStyleSheet(

                        'background-color: rgb(255, 170, 0)')

                elif result[i][3] == 700:

                    self.buttons[i].setStyleSheet(

                        'background-color: rgb(255, 255, 0)')

                elif result[i][3] == 600:

                    self.buttons[i].setStyleSheet(

                        'background-color: rgb(85, 255, 0)')

                elif result[i][3] == 500:

                    self.buttons[i].setStyleSheet(

                        'background-color: rgb(85, 255, 255)')

                elif result[i][3] == 400:

                    self.buttons[i].setStyleSheet(

                        'background-color: rgb(85, 85, 255)')

                elif result[i][3] == 300:

                    self.buttons[i].setStyleSheet(

                        'background-color: rgb(85, 0, 127)')

                elif result[i][3] == 200:

                    self.buttons[i].setStyleSheet(

                        'background-color: rgb(31, 135, 75)')

            self.buttons[i].clicked.connect(self.push)

Чтобы иметь возможность обращаться к каждой кнопке, нам необходимо добавить их в список. Из-за большого количества мест этот фрагмент был добавлен в файл с классом дизайна формы. Далее мы проходимся по всем кнопкам и присваиваем подходящий для нее цвет. Если в базе данных у данного места стоит нулевая стоимость, то соответствующая кнопка становится недоступной.

**Accepting form**

        global play\_name

        global gl\_theatre

        if '@' not in self.lineEdit.text():

            self.label\_2.setText('Неверный формат email')

        else:

            import random

            n = str(random.randint(1, 99999))

            text = play\_name + '\nНомер заказа:' + n + '\nВаши билеты:\n'

            n = self.listWidget.count()

            for j in [str(self.listWidget.item(i).text()) for i in range(n)]:

                text += j + '\n'

            text += 'Спасибо за покупку!'

            smtp\_host = 'smtp.yandex.ru'

            login, password = 'tickets.theatre@yandex.ru', 'password\_to\_app'

            msg = MIMEText(text, 'plain', 'utf-8')

            msg['Subject'] = Header('Покупка билетов', 'utf-8')

            msg['From'] = login

            msg['To'] = ", ".join(self.lineEdit.text())

            sender = smtplib.SMTP(smtp\_host, 587)

            sender.set\_debuglevel(1)

            sender.starttls()

            sender.login(login, password)

            try:

                sender.sendmail(

                    msg['From'], self.lineEdit.text(), msg.as\_string())

                self.label\_2.setText('')

                gl\_theatre.close()

                self.close()

                sender.quit()

                exporter = Import\_export()

                for i in self.chosen:

                    exporter.update\_table(self.data\_base, self.table, "UPDATE " +

                                        self.table + " SET price = 0 WHERE id = "

                                        + i.objectName()[1:])

            except:

                self.label\_2.setText('Произошла ошибка')

В данном фрагменте выполняется отправка email. Используется сервер Yandex для автоматической отправки email. При ошибке отправки приложение сообщает пользователю об этом и отменяет покупку.

**4. Заполнение базы данных**

База данных заполнялась при помощи парсеров parser\_placex.py, parser\_repertoire.py и update\_data.py, которые получали данные с сайта театра. Парсинг выполняется при помощи библиотек bs4 и Selenium.

**5. Вывод**

Таким образом, это приложение для покупки билетов, которое позволяет без больших усилий подключать к данному сервису новые театры путем добавления новых баз данных, заполненных по тому же принципу, что и стандартная.

Чтобы сделать приложение максимально функциональным, необходимо загрузить базу данных на удаленный сервер и добавить реальную покупку.