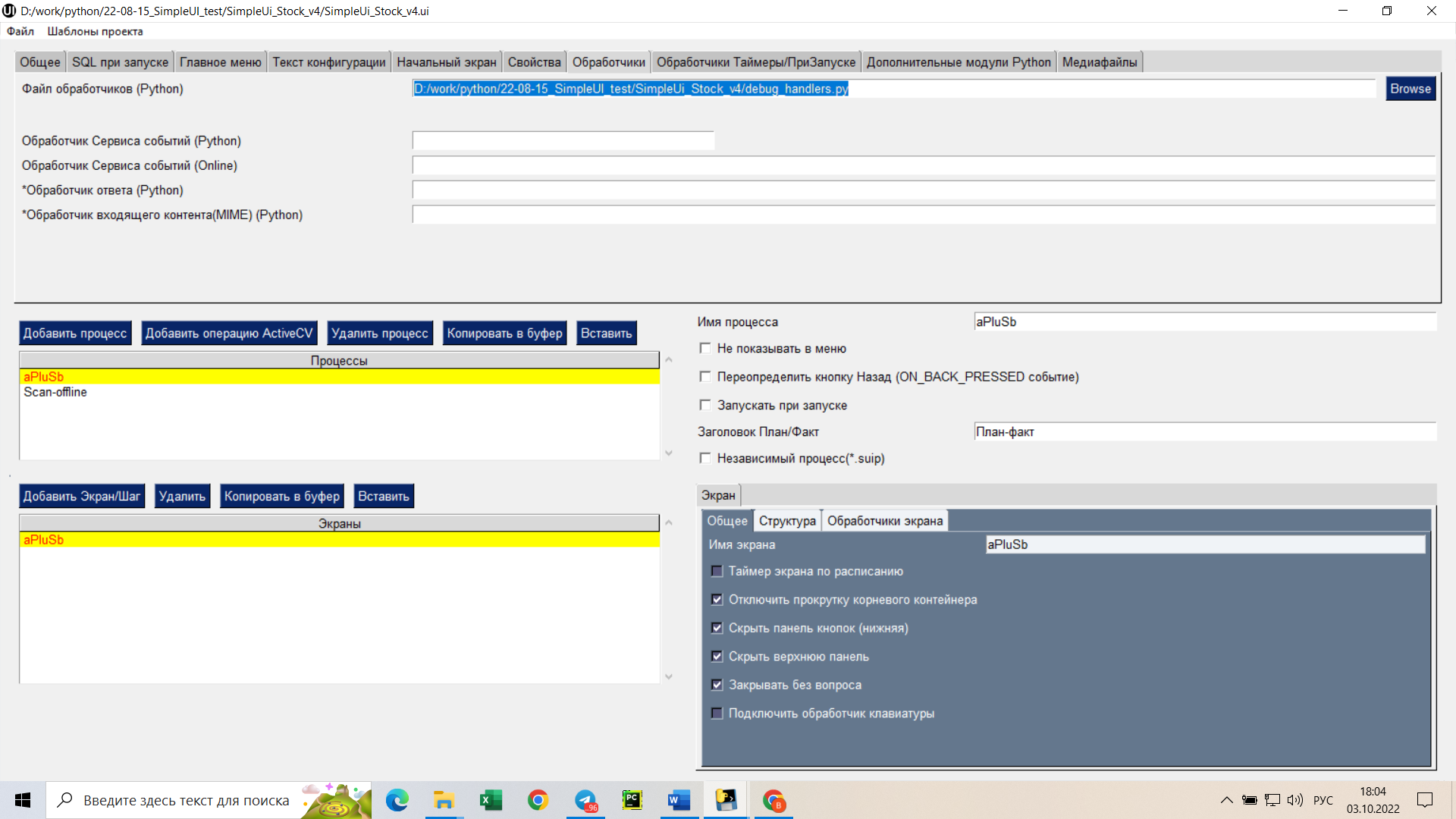
# Послед действ SimpleUi\_Stock\_v4

1. создал отдельную папку D:\work\python\22-08-15\_SimpleUI\_test\SimpleUi\_Stock\_v4
2. создал в редакторе конфигураций SimpleUi\_Stock\_v4 (перетащил папку с работающий Конфиг из SimpleStart)  
   поменял путь с учетом другого названия файла



В ПК на Конструкторе, на вкладке «Экран», выбираем «Структура» добавим новый элемент «Контейнер» (Рисунок 4)

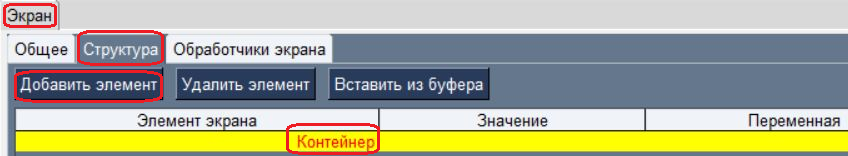


Рисунок 4 Добавление элементов экрана.

поставим «Ориентация» - «Вертикально» и «Высота» и «Ширина» – по размеру элементов. Это будет корневой контейнер экрана (ССЫЛКА описание работы с контейнером).

Добавим в контейнер несколько элементов: «Надпись», «Поле ввода число», «Кнопка». В соответствие с Рисунок 5

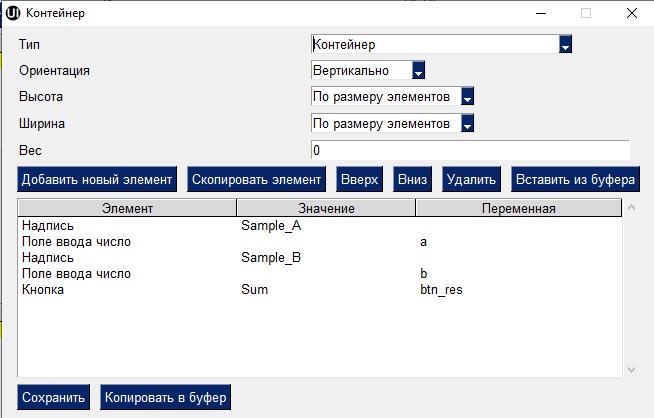


Рисунок 5 Редактирование элемента экрана – «Контейнер»

Примечание

Можно создавать разметку любой сложности вписывая в контейнер другие контейнеры – горизонтальные и вертикальные.

В конструкторе на ПК сохраняем конфигурацию, перезапускаем программу на устройстве.

В рамках этого примера мы сделаем оффлайн обработчик на Python (ССЫЛКА Все варианты обработчиков).

# Генерация файла отладки

1. В папке «Конфигурации» создаем пустой файл «debug\_handlers.py». Путь и имя файла необходимо указать в Конструкторе на вкладке «Обработчики», окно «Файл обработчиков (Python)» (Рисунок 6)

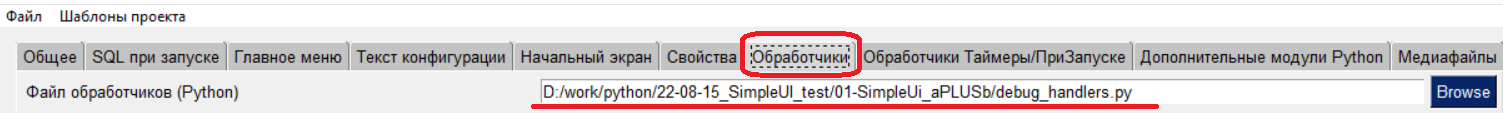


Рисунок 6 Добавление пути и имени файла в поле «Файл обработчиков (Python)»

1. В Конструкторе выбираем: «Файл» –> «Сгенерировать отладочный файл» ()

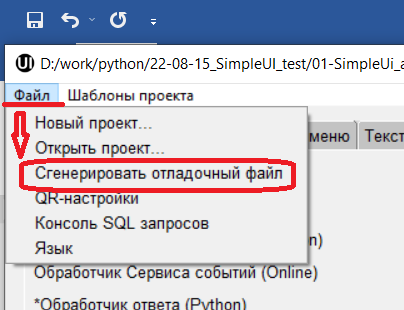


Рисунок 7 Создание отладочного файла «debug\_handlers.py»

1. В папке где расположен файл «simple\_editor.exe», появляется «debug\_handlers.py» который содержит основные параметры конфигурации и обработчики процесса (Рис. 6).

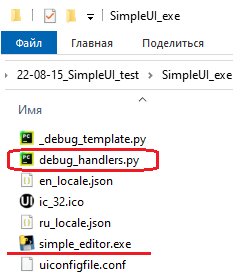


Рисунок 8 Расположение отладочного файла «debug\_handlers.py»

Все обработчики «Конфигурации» системы Simple UI необходимо помещать в файле «debug\_handlers.py», между строками:

# -BEGIN CUSTOM HANDLERS  
  
# -END CUSTOM HANDLERS

При отладке, названия обработчиков в Конструкторе и отладочном файле «debug\_handlers.py» пишутся с подчёркиванием в начале наименования – «\_ sample1\_on\_create».

Все перечисленное выше создадим в новой функцию (обработчике экрана), которая будет обрабатывать события ввода на этом экране:

**def** \_sample1\_on\_input**(**hashMap**,**\_files=**None,**\_data=**None):**

**if** hashMap.get**(**"listener"**)**=="btn\_res"**:**

hashMap.put**(**"toast"**,**str**(**int**(**hashMap.get**(**"a"**))**+int**(**hashMap.get**(**"b"**))))**

**return** hashMap

В Конструкторе для каждого экрана конфигурации необходимо указать название обработчиков отвечающих за запуск и работу экрана соответственно в окнах: «Обработчик ПриЗапуске (Python)» и «Обработчик ввода (Python)» (Рисунок 9).

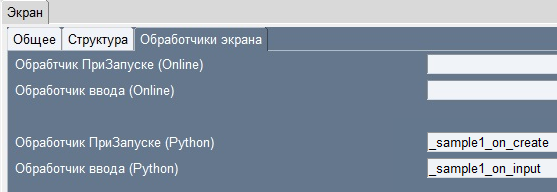


Рисунок 9 Обработчики запуска и работы экрана

Важно понимать, что при нажатии кнопки на экране, сработает обработчик ввода и экран будет перерисован, данные в полях ввода пропадут. Надо зайти в Конструктор «Экран» -> «Структура» и добавить в Значения в полях ввода переменные «a» и «b» через @ (Рисунок 10). Значение – это значения по умолчанию для полей ввода, а символ «@» в SimpleUI означает привязку переменных к элементам. С помощью него можно выводить, например надпись на экран.

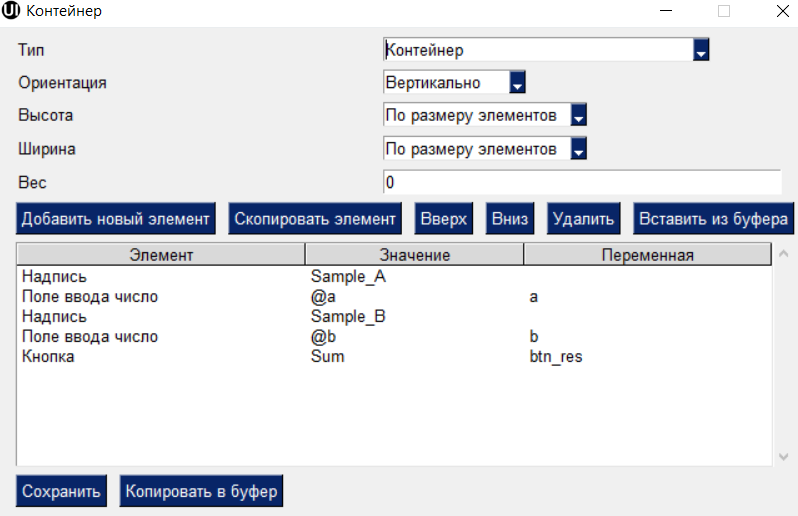


Рисунок 10 Контейнер экрана.

Для визуального оформления экрана добавим еще один обработчик, который отвечает за заполнение экрана в момент его открытия:

**def** \_sample1\_on\_create**(**hashMap**,**\_files=**None,**\_data=**None):**

**if** **not** hashMap.containsKey**(**"a"**):**

hashMap.put**(**"a"**,**""**)**

**if** **not** hashMap.containsKey**(**"b"**):**

hashMap.put**(**"b"**,**""**)**

**return** hashMap

Необходимо указать имя обработчиков в конструкторе на закладке Обработчики (Рисунок 9), перезапустить конфигурацию и проверить работоспособность.

# Использование debugger PyCharm при отладке конфигурации

При необходимости проведения пошаговой проверки работоспособности Конфигурации с использованием debugger’a PyCharm (на ПК и Устройстве). Отслеживая на ПК в debugger’е действий совершаемых в Конфигурации на Устройстве. Требуется повести следующие действия:

1. Открываем на ПК файл «debug\_handlers.py» в PyCharm.
2. Проводим проверку IP и номера портов на ПК и устройстве. В настройках приложений на ПК и устройстве должен быть указан адрес одной и той же подсети (Рисунок 11).

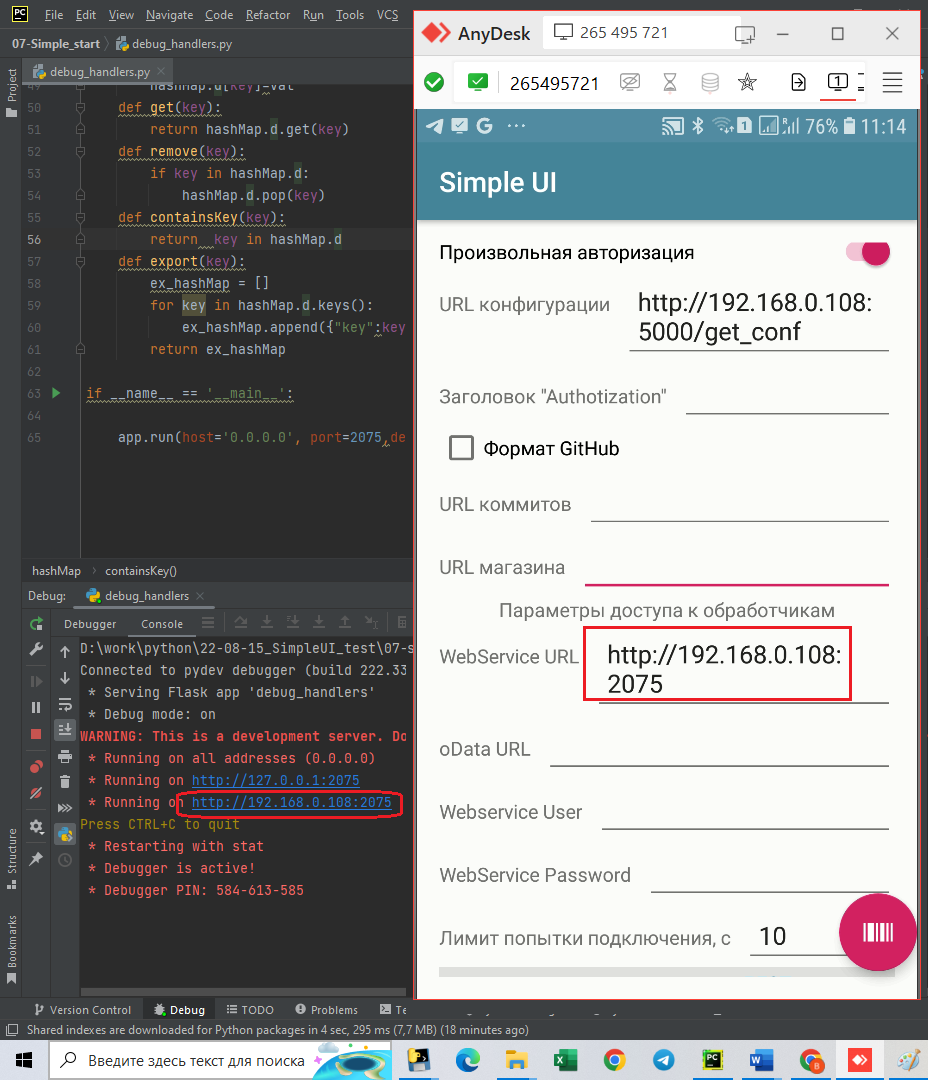


Рисунок 11 Сравнение адреса подсети.

1. Проводим Обновление /перезапуск Конфигурации на устройстве
2. Проверяем в командной строке Конструктора (дополнительное окно Конструктора на ПК) - установлена связь или нет (Рисунок 12).

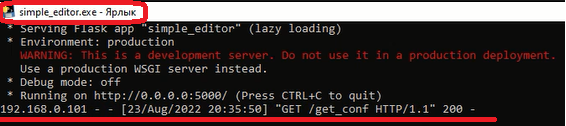


Рисунок 12 Дополнительное окно Конструктора

1. Выполняем действия в Конфигурации на устройстве
2. Проверяем реакцию на работу Конфигурации в debugger PyСharm на ПК.

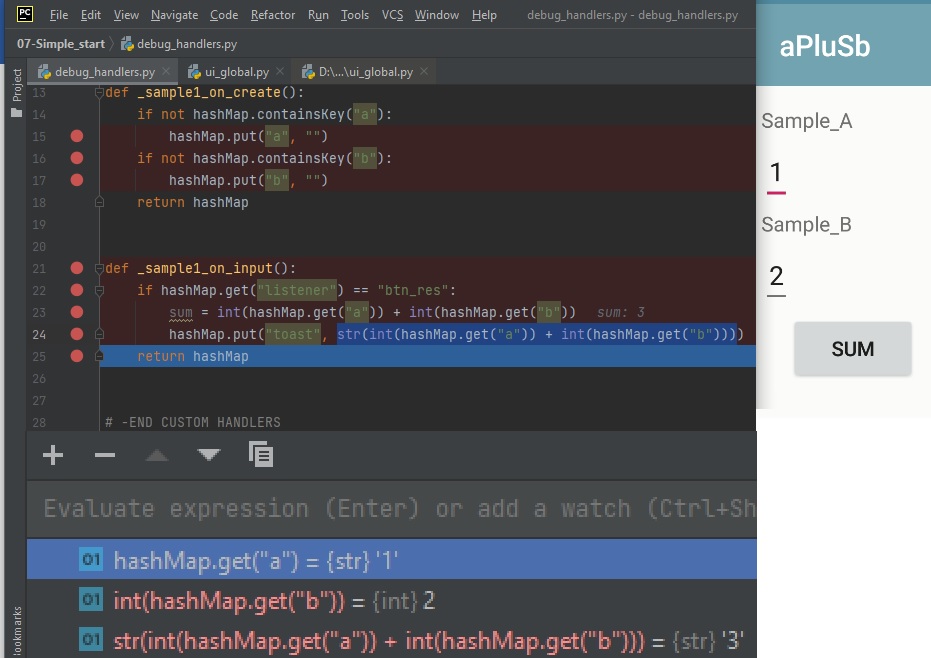


Рисунок 13 Результат пошаговой отладки Конфигурации в debugger PyСharm на ПК и устройстве

# Добавление таблицы. Работа с сканером, несколькими экранами, SQL через ORM оффлайн обработчиками.

Этот процесс будет содержать два экрана. На первом происходит сканирование штрихкода. На нем же отображаются ранее отсканированные записи, в виде таблицы. На втором экране будет происходить ввод количества и добавление записей в SQL СУБД приложения. (ССЫЛКА Все варианты *SQL СУБД в Simple UI)*

# Первый экран. Сканирование штрихкода

Добавляем процесс, переименовываем первый экран в «Scan-offline». Он будет просто содержать надпись-подсказку и таблицу.

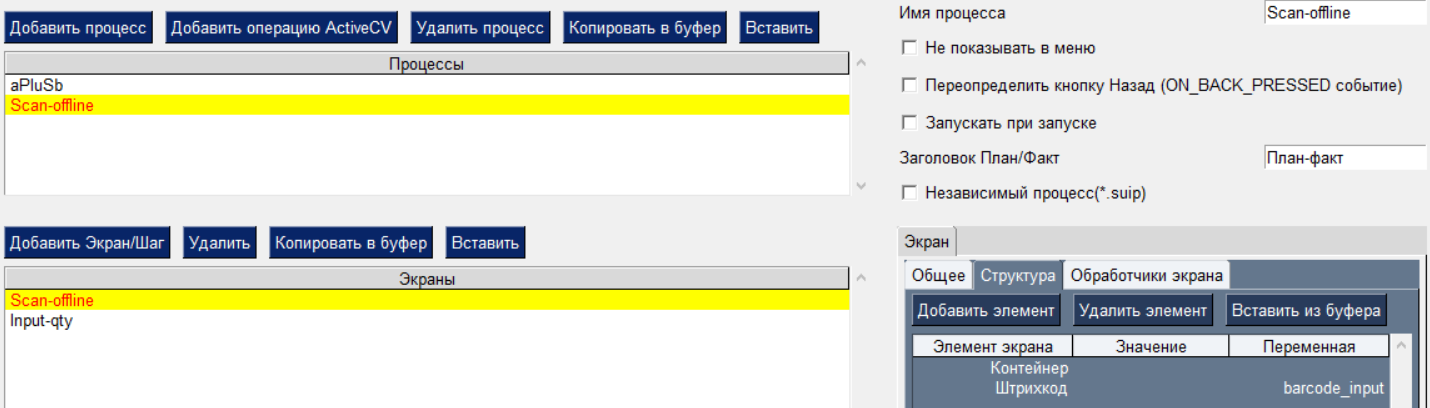


Рисунок 14 Первый экран и структура - "Сканирование штрихкода"

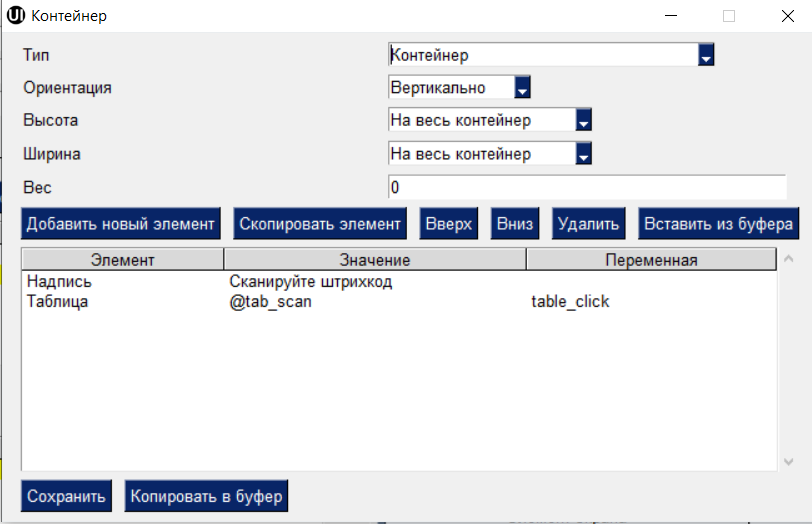


Рисунок 15 Контейнер в структуре первого экрана «Scan-offline» - "Сканирование штрихкода"

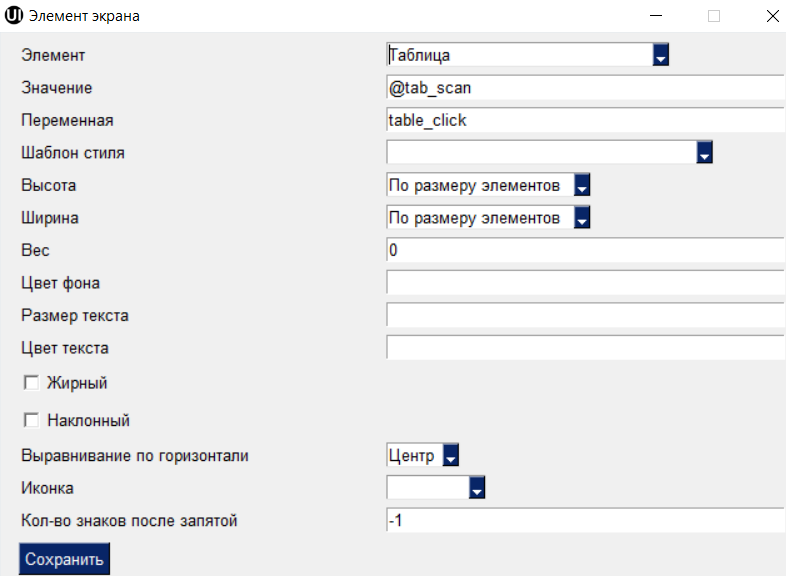


Рисунок 16 Таблица в структуре первого экрана «Scan-offline» - "Сканирование штрихкода"

Также на этом экране размещаем элемент Штрихкод который будет сканировать в переменную «barcode\_input».

Обязательно указывать переменную в которую будет сканироваться штрихкод, иначе управляющий элемент просто не будет размещен на экране.

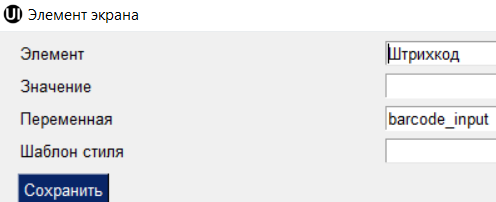


Рисунок 17 Элемент экрана «Штрихкод» в структуре первого экрана «Scan-offline» - "Сканирование штрихкода"

Сканирование будет происходить камерой устройства при нажатии на кнопку сканирования внизу справа.

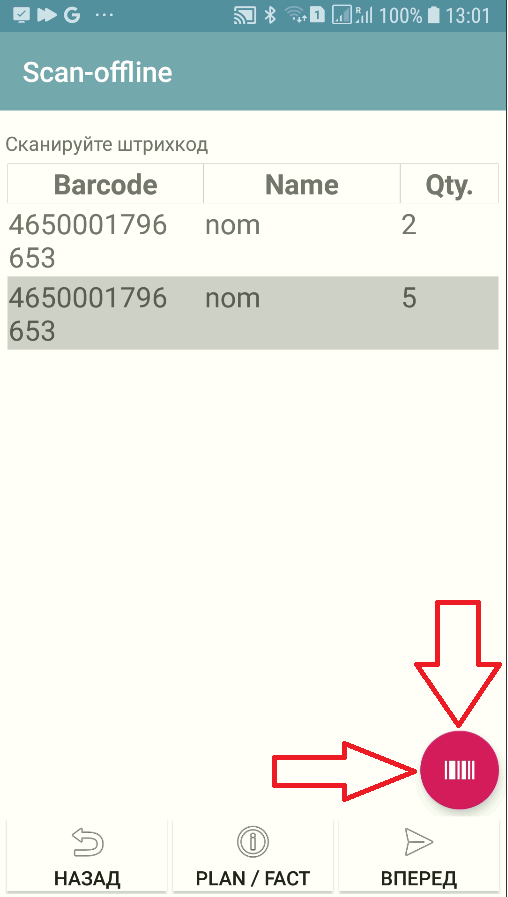


Рисунок 18 Расположение кнопки «Сканирование» на первом экране «Scan-offline» - "Сканирование штрихкода"

Этого достаточно, теперь нужно написать обработчики ввода.

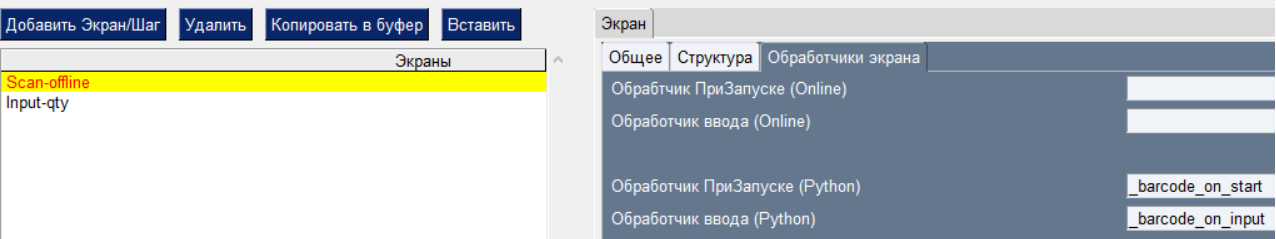


Рисунок 19 Экраны и обработчики экрана “Scan-ofline” » - "Сканирование штрихкода"

Все переменные, которые есть на этот момент заходят в виде стека и выходят в виде стека, только в данном случае это происходит в виде секции hashmap в json запросе. Для работы со стеком переменных тут сделан класс hashMap.

Сам обработчик «\_barcode\_on\_input» определяет событие сканирования, читает штрихкод и открывает экран «Input-qty», где вводится параметр «количество».

def \_barcode\_on\_input(hashMap, \_files=None, \_data=None):

if hashMap.get('listener') == 'barcode':

hashMap.put('ShowScreen', 'Input-qty')

return hashMap

Также, поскольку у нас есть второй экран, его нужно показать после того как штрихкод будет обработан и товар определен. Делается это командой «ShowScreen», где в качестве аргумента используется имя экрана. Можно например не переключать экран если товар не найден (в примере он переключается в любом случае по сканированию). Т.е. пока команды не будет – экран не переключится сам.

# Второй экран. Формирование и запись в таблицу базы данных.

Структура второго экрана содержит элементы позволяющие ввести количество в таблицу.

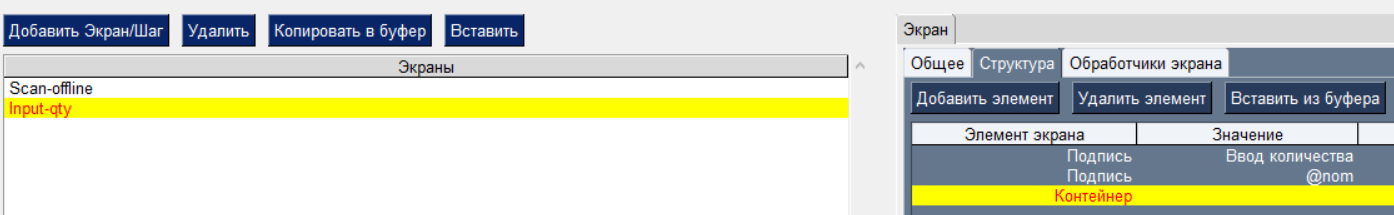


Рисунок 20 Второй экран и структура - "Сканирование штрихкода"

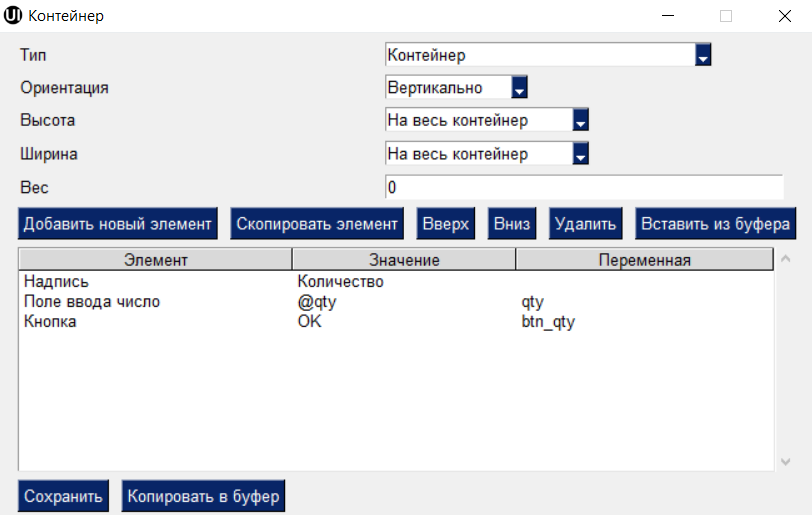


Рисунок 21 Контейнер в структуре второго экрана «Input-qty» - "Сканирование штрихкода"

Второй экран содержит логику работы с СУБД.

Для работы с таблицами СУБД сделаем дополнительный общий модуль «ui\_global» , пример в комплекте разработчика в «ui\_global.py» в котором определяется подключение к СУБД, создаются таблицы и происходит инициализация generate\_mapping (все это относится к Pony ORM (ССЫЛКА Другие варианты передачи файла конфигурации)).

Важно то, что мы создали объект Record который также является таблицей Record

**class** Record**(**db.Entity**):**

barcode = Required**(**str**)**

name = Required**(**str**)**

qty = Required**(**int**)**

Нам надо указать его на закладке Дополнительный Python – модули.

Рисунок 22 Добавление модуля в элементе Конфигурации - "Сканирование штрихкода"

Теперь мы можем использовать модуль «ui\_global» и вызывать его в обработчиках. Добавим его в основной файл «debug\_handlers.py» и создадим обработчик

**import** ui\_global

**def** init\_on\_start**(**hashMap**,**\_files=**None,**\_data=**None):**

ui\_global.init**()**

**return** hashMap

Теперь нам надо чтобы init\_on\_start вызывалась при запуске системы, укажем вызов на закладке «Обработчики таймеров/при запуске», указав в качестве периода -1 (положительные значения это интервал таймера в миллисекундах, а -1 означает однократное выполнение)

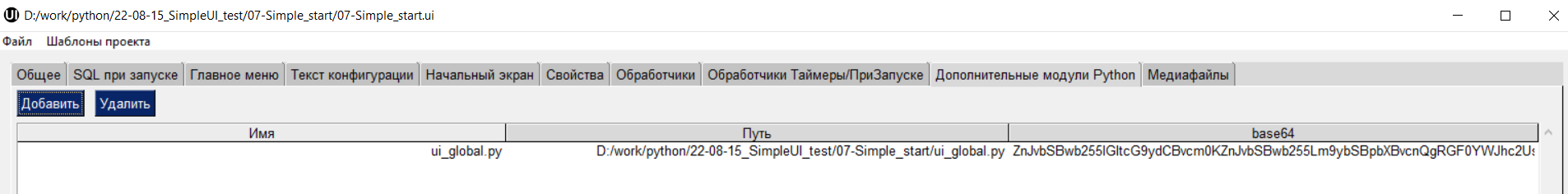


Рисунок 23 Создание таблицы перед началом работы Конфигурации на устройстве

На этом этапе при запуске приложения создается таблица Record

Добавляем в «debug\_handlers.py», обработчик на Python который будет записывать введенные ранее переменные barcode, nom и qty в таблицу

def \_input\_qty(hashMap, \_files=None, \_data=None):

if hashMap.get('listener') == 'btn\_qty':

with db\_session:

p = Record(barcode=hashMap.get('barcode\_input'), name= hashMap.get('nom'), qty=hashMap.get('qty'))

commit()

hashMap.put('ShowScreen', 'Scan-offline')

hashMap.put('toast', 'Добавлено')

return hashMap

Предполагается, что сделаны импорты:

**import** ui\_global

**from** pony.orm.core **import** db\_session

**from** pony **import** orm

**from** pony.orm **import** Database**,**Required**,**Set**,**select**,**commit

Обработчик, помимо добавления записи, также выводит тост и переключает на предыдущий экран, замыкая цикл.

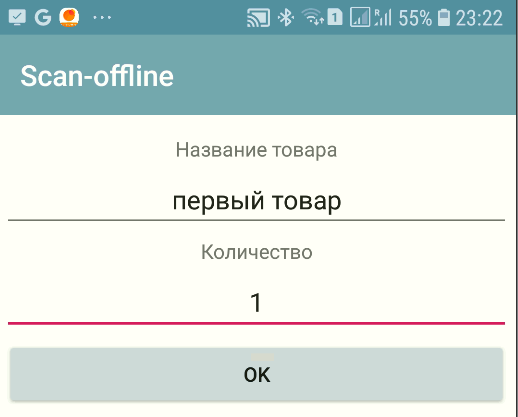


Рисунок 24 Добавление названия и количества товара

Остается только прописать обработчики на закладке Обработчики и проверить работу.

У нас остался еще один обработчик, который будет выводить таблицу Record на экране.

Для этого мы размещаем на первом экране таблицу и пишем обработчик «\_barcode\_on\_start» который прочитает записи в Record. Переменная таблицы (как и других сложных экранных форм) – это JSON объект, описывающий структуру и данные. Добавляем таблицу с тремя столбцами (штрихкод, наименование и количество) – массив columns и добавляем в rows записи из базы данных.

table = **{**

"type"**:** "table"**,**

"textsize"**:** "20"**,**

"columns"**:** **[**

**{**

"name"**:** "barcode"**,**

"header"**:** "Barcode"**,**

"weight"**:** "2"

**},**

**{**

"name"**:** "name"**,**

"header"**:** "Name"**,**

"weight"**:** "2"

**},**

**{**

"name"**:** "qty"**,**

"header"**:** "Qty."**,**

"weight"**:** "1"

**}**

**]**

**}**

*#work with SQL via Pony ORM*

query = select**(**c **for** c **in** ui\_global.Record**)**

rows=**[]**

**for** record **in** query**:**

rows.append**({**"barcode"**:**record.barcode**,**"name"**:**record.name**,**"qty"**:**record.qty**})**

table**[**'rows'**]** =rows

hashMap.put**(**"table"**,**json.dumps**(**table**))**

**return** hashMap

Запускаем Конфигурацию в отладчике и проверяем работоспособность всех элементов.

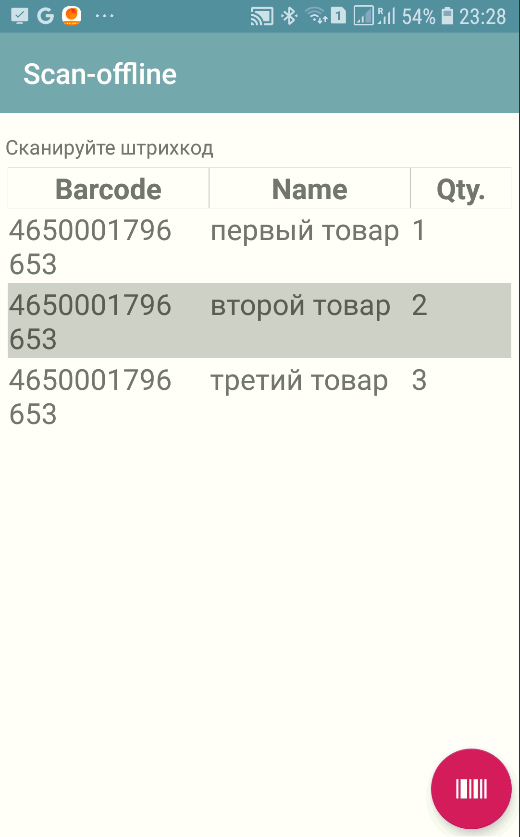


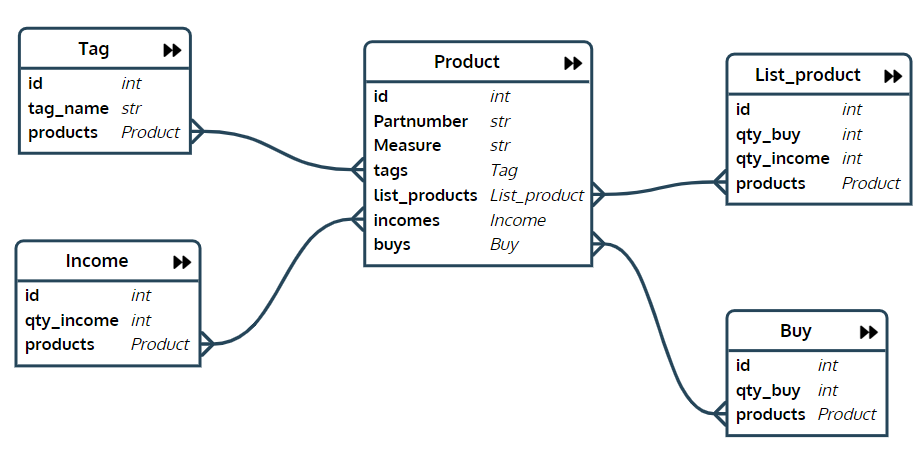
Рисунок 25 Первый экран «Scan-offline» в работающей Конфигурации

# Формирую меню 1го ур-ня

Создал 4ре новых процесса Закупки/Поступления/Отчет о поступлениях/Товары

Проверил работоспособность (Конфигур работает без запуска debug handler.py on PC)

# Добавление БД



Где Закупки/ Buy

Поступления/ Income

Отчет о поступлениях/ List\_product

Товары / Product

Проверка работоспособности

# Создание процесса Закупки

# Вопросы