УДК 004

Федотов В. А.

Студент, 4 курс, факультет «Информационные системы и технологии»

Северный Арктический Федеральный Университет, Высшая школа информационных технологий и автоматизированных систем

Россия, г Архангельск

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ ОПТОВОГО СКЛАДА НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON

Аннотация: В статье описывается разработка информационной системы для оптового склада.

Ключевые слова: Разработка, информационная система, python, PyCharm, база данных.

Fedotov V. A.

Student, 4 year, faculty "Information Systems and Technology"

Northern Arctic Federal University, Graduate School of Information Technology and Automated Systems

Russia, Arkhangelsk

DEVELOPMENT OF THE INFORMATION SYSTEM FOR WHOLESALE WAREHOUSE IN THE PYTHON PROGRAMMING LANGUAGE

Annotation: The article describes the development of an information system for a wholesale warehouse.

Keywords: Development, information system, python, PyCharm, database.

1 ВЕРБАЛЬНОЕ ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ОПТОВОГО СКЛАДА, ПРОБЛЕМА И ЦЕЛЬ

1.1 Описание предметной области

Оптовый склад занимается хранением товаров. Товары разделяют на определенные категории. Работа на складе производится следующим образом:

- поставка товара;
- товар заносится в базу.

Необходимо автоматизировать деятельность склада. Для работы с товаром и категориями в информационной системе нужно реализовать просмотр, добавление, изменение, удаление и поиск товаров.

1.2 Выделение проблемы

Проблема заключается в большом количестве товара, поступающего на оптовый склад, что усложняет ведение учета товара.

1.3 Постановка цели

Необходимо автоматизировать процесс учета товаров, поступающих на оптовый склад.

2 КОНЦЕПТУАЛЬНОЕ ОПИСАНИЕ ОПТОГОВОГО СКЛАДА

Необходимо осуществить ведение учета товаров на оптовом складе. Объектами данной предметной области будет являться:

- товары;

- категории товаров.

Определим параметры, которые будем указывать у товаров и категорий (таблица 1).

Таблица 1 – Описание свойств товаров и категорий

Наименование объекта	Параметры объекта
Товары	Идентификатор
	Наименование
	Категория
	Количество
	Единицы измерения
	Цена
Категории	Идентификатор
	Наименование
	Создатель

Определим ограничения:

- товар должен иметь только одну категорию;

- в категории может быть несколько товаров.

Схема взаимодействия объектов представлена на рисунке 1.

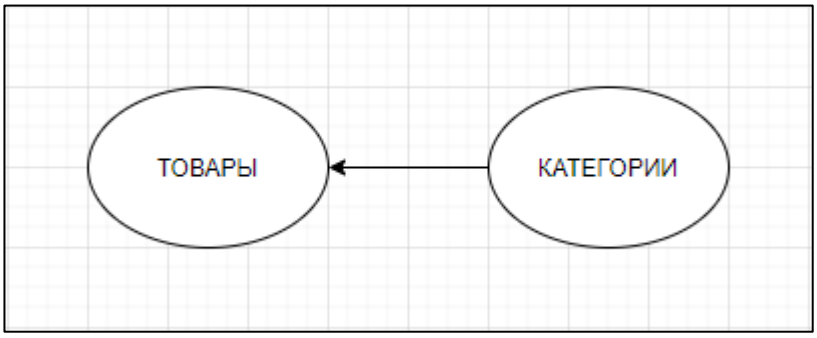

Рисунок 1 – Схема взаимодействия объектов

3 РАЗРАБОТКА НЕОБХОДИМЫХ МОДУЛЕЙ НА ОСНОВАНИИ ВЫДЕЛЕННЫХ ОБЪЕКТОВ

3.1 Описание редактора и языка программирования

Для разработки информационной системы будем использовать редактор PyCharm, который делает разработку максимально продуктивной благодаря функциям автодополнения и анализа кода, мгновенной подсветке ошибок и быстрым исправлениям. Автоматические рефакторинги помогают эффективно редактировать код, а удобная навигация позволяет мгновенно перемещаться по проекту. Редактор PyCharm предназначен для максимально продуктивной разработки на Python, JavaScript, CoffeeScript, TypeScript, CSS и популярных языках шаблонов.

В разработке будем использовать язык Python. Python – это универсальный современный язык программирования высокого уровня, к преимуществам которого относят высокую производительность программных решений и структурированный, хорошо читаемый код.

3.2 Разработка макета

Разработку приложения начнем с проектирования макета. Создание макета – важный этап в разработке информационной системы. Он позволяет представить интерфейс будущего приложения. Макеты для проектируемой системы представлены на рисунке 2 и рисунке 3.

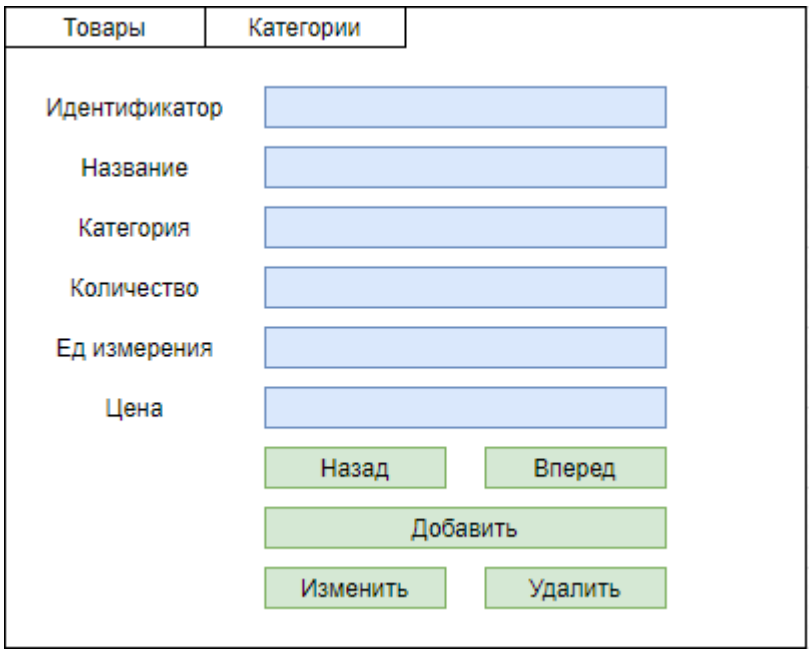

Рисунок 2 – Макет окна товаров

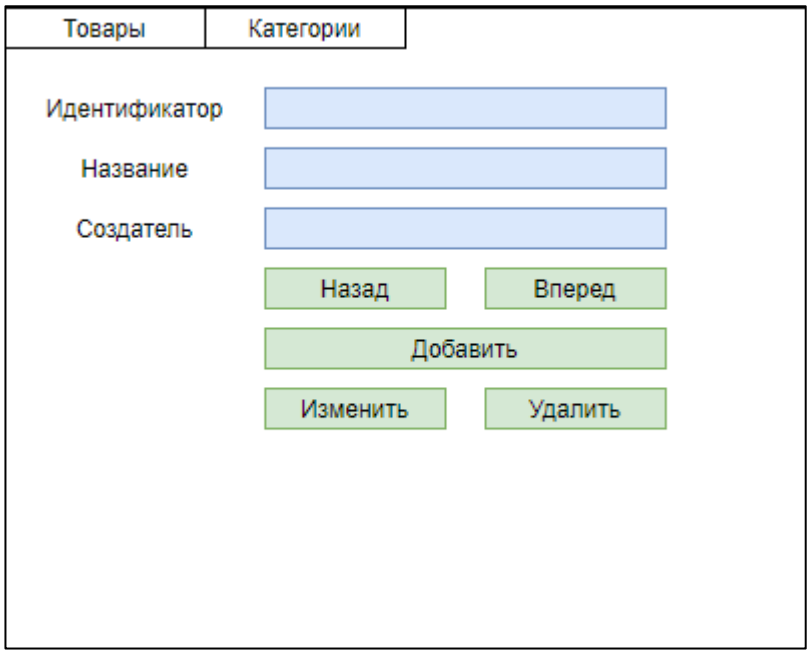

Рисунок 3 – Макет окна категорий

3.3 Разработка модулей

Создадим класс модели «Product» и в методе «__init__» инициализируем переменные для товаров. Класс модели «Product» представлен в листинге 1.

Листинг 1 – Класс модели «Product»

```
class Product(object):
  def __init__(self, name, category, quantity, unit, cost):
```

```
        self.__id = _next_id()
```

Продолжение листинга 1

```
        self.__name = name
        self.__category = category
        self.__quantity = quantity
        self.__unit = unit
        self.__cost = cost
```

Объявим свойства для атрибутов в классе модели «Product». Свойства для атрибутов представлены в листинге 2.

Листинг 2 – Свойства для атрибутов

```
@property
def id(self):
    return self.__id
@id.setter
def id(self, value):
    self.__id = value
@property
def name(self):
    return self.__name
@name.setter
def name(self, value):
    self.__name = value
@property
def category(self):
    return self.__category
@category.setter
def category(self, value):
    self.__category = value
@property
def quantity(self):
    return self.__quantity
@quantity.setter
def quantity(self, value):
    self.__quantity = value
@property
def unit(self):
    return self.__unit
@unit.setter
def unit(self, value):
    self.__unit = value
@property
def cost(self):
    return self.__cost
@cost.setter
def cost(self, value):
    self.__cost = value
```

Далее необходим механизм, позволяющий извещать вид об изменениях модели. Создадим абстрактный класс «AbstractModel», который используем в

качестве родителя для модели «Database». Класс «AbstractModel» представлен в листинге 3.

Листинг 3 – Класс «AbstractModel»

```
class AbstractModel(object):
  def __init__(self):
    self.listeners = []

  def addListener(self, listenerFunc):
    self.listeners.append(listenerFunc)

  def update(self):
    for eachFunc in self.listeners:
      eachFunc(self)
```

Создадим класс базы данных товаров «Database», в котором укажем название файла «product.pkl», в который заносится информация о состоянии объекта. Класс «Database» представлен в листинге 4.

Листинг 4 – Класс «Database»

```
class Database(AbstractModel):
  def __init__(self):
    super(Database, self).__init__()
    self.filename = 'product.pkl'
    self.database = {}
    self.index = 0
    try:
      self.open_database()
    except:
      self.save_database()
```

Объявим свойства используя «lambda». В листинге 5 представлено объявление свойств.

Листинг 5 – Объявление свойств

```
id = property(lambda self: self.database[self.index].id)
name = property(lambda self: self.database[self.index].name)
category = property(lambda self: self.database[self.index].category)
quantity = property(lambda self: self.database[self.index].quantity)
unit = property(lambda self: self.database[self.index].unit)
cost = property(lambda self: self.database[self.index].cost)
```

В классе «Database» создадим два метода для сериализации и десериализации. Сериализация – процесс перевода какой-либо структуры данных в последовательность битов. Обратной к операции сериализации является операция десериализации. Методы представлены в листинге 6.

Листинг 6 – Методы сериализации и десериализации

```
def open_database(self):
  with open(self.filename, 'rb') as file:
    self.database = pickle.load(file)

def save_database(self):
```

Продолжение листинга 6

```
  with open(self.filename, 'wb') as file:
    pickle.dump(self.database, file)
```

Для реализации добавления, изменения и удаления необходимо создать соответствующие методы, которые представлены в листинге 7. В метод добавления и изменения товаров передаются данные из модели. При удалении передается идентификатор.

Листинг 7 – Методы добавления, изменения и удаления

```
def add_product(self, name, category, quantity, unit, cost):
  product = Product(name, category, quantity, unit, cost)
  if product.id in self.database:
    product.id = list(self.database.keys())[-1] + 1
  self.database[product.id] = product
  self.index = list(self.database.keys())[-1]
  self.update()
  self.save_database()

def edit_product(self, id, name, category, quantity, unit, cost):
  product = self.get_product_by_id(id)
  if not product:
    raise ValueError('Товара нет')
  product.name = name
  product.category = category
  product.quantity = quantity
  product.unit = unit
  product.cost = cost
  self.update()
  self.save_database()

def delete_product(self, id):
  del self.database[id]
  self.save_database()
  self.index = list(self.database.keys())[0]
  self.update()
```

Для получения следующего и предыдущего товара необходимо создать две функции «next_product» и «prev_product». Функции представлены в листинге 8.

Листинг 8 – Функции «next_product» и «prev_product»

```
def next_product(self):
  if self.index == list(self.database.keys())[-1]:
```

```
    raise Exception("Больше записей нет!")
  else:
    try:
      self.index = self.index + 1
      while self.index <= list(self.database.keys())[-1] and
self.index not in self.database.keys():
        self.index = self.index + 1
      self.update()
```

Продолжение листинга 8

```
      return self.database[self.index]
    except Exception:
      raise Exception("Больше записей нет!")

def prev_product(self):
  if self.index <= 0:
    raise Exception("Больше записей нет!")
  else:
    try:
      self.index = self.index - 1
      while self.index >= 0 and self.index not in
self.database.keys():
        self.index = self.index - 1
      self.update()
      return self.database[self.index]
    except Exception:
      raise Exception("Больше записей нет!")
```

Для получения идентификатора товара используется метод «get_product_bi_id», представленный в листинге 9.

Листинг 9 – Метод «get_product_bi_id»

```
def get_product_by_id(self, id):
  if id not in self.database:
    return None
  return self.database[id]
```

Далее создадим отдельный файл с «.py» расширением, в котором будем прописывать взаимосвязь интерфейса и модели. Создадим класс «ProductView», в методе «__init__» инициализируем данные. Метод «__init__» представлен в листинге 10.

Листинг 10 – Метод «__init__»

```
class ProductView(QtWidgets.QMainWindow):
  def __init__(self):
    super(ProductView, self).__init__()
    self.ui = Ui_MainWindow()
    self.ui.setupUi(self)
    self.model = BankDatabase()
    self.modelCategory = DatabaseCategory()
    self.refreshCategory()
```

```
    self.ui.tabWidget.blockSignals(False)
    self.ui.tabWidget.currentChanged.connect
        (self.refreshCategory)
    self.model.addListener(self.updateProduct)
    self.modelCategory.addListener(self.updateCategory)
    self.ui.CategoryProduct.activated.connect
        (self.selectCategory)
    self.ui.btnAddProduct.clicked.connect(self.addProduct)
    self.ui.btnUpdateProduct.clicked.connect(self.changeProduct)
    self.ui.btnDeleteProduct.clicked.connect(self.deleteProduct)
    self.ui.btnForward.clicked.connect(self.nextProduct)
```

Продолжение листинга 10

```
self.ui.btnBack.clicked.connect(self.prevProduct)
```

Для связи методов, указанных в классе «Database» с кнопками и полями ввода, для осуществления работы с данными, создадим методы «addProduct», «changeProduct» и «deleteProduct», представленные в листинге 11.

Листинг 11 – Методы «addProduct», «changeProduct» и «deleteProduct»

```
def addProduct(self):
  try:
    self.model.add_product(str(self.ui.NameProduct.text()),
                  selectedCategoryValue,
                  int(self.ui.QuantityProduct.text()),
                  str(self.ui.UnitProduct.text()),
                  int(self.ui.CostProduct.text()))
    self.test_db()
  except Exception as e:
    self.message.setText(str(e))
    self.message.exec_()
def changeProduct(self):
  try:
    self.model.edit_product(int(self.ui.IdProduct.text()),
                 str(self.ui.NameProduct.text()),
                 selectedCategoryValue,
                 int(self.ui.QuantityProduct.text()),
                 str(self.ui.UnitProduct.text()),
                 int(self.ui.CostProduct.text()))
  except Exception as e:
    self.message.setText(str(e))
    self.test_db()
    self.message.exec_()

def deleteProduct(self):
  self.model.delete_product(int(self.ui.IdProduct.text()))
  self.test_db()
```

Для того, чтобы отображать данные необходимо создать две кнопки «Вперед» и «Назад». Для них необходимо реализовать функции «nextProduct» и «prevProduct». Функции представлены в листинге 12.

Листинг 12 – Функции «nextProduct» и «prevProduct»

```
def nextProduct(self):
  try:
    self.model.next_product()
  except Exception as e:
    self.message.setText(str(e))
    self.message.exec_()

def prevProduct(self):
  try:
    self.model.prev_product()
  except Exception as e:
    self.message.setText(str(e))
```

Продолжение листинга 12

```
    self.message.exec_()
```

Разработка модели категорий и класса базы данных категорий разработаны аналогичным с товарами образом.

3.4 Интерфейс приложения

Для более понятного использования приложения представим интерфейс. Окно товаров и окно категорий изображено на рисунке 4 и рисунке 5 соответственно.

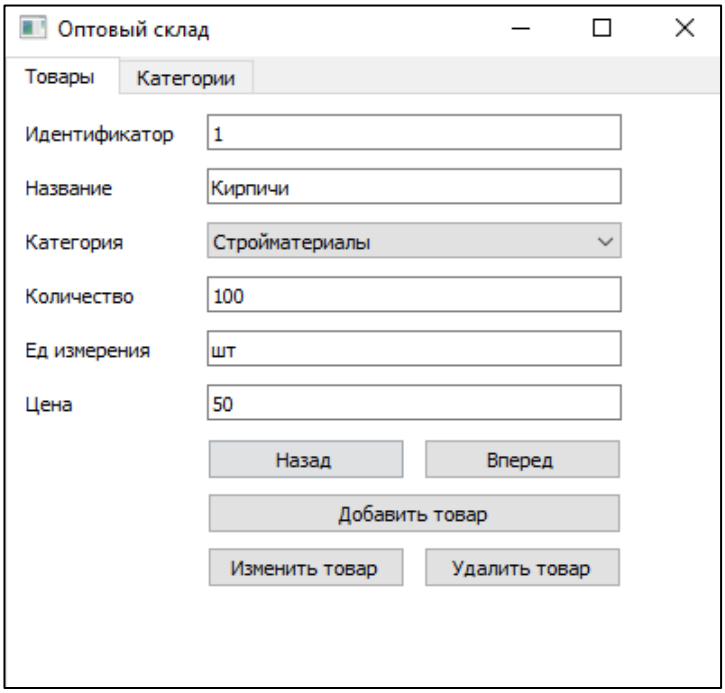

Рисунок 4 – Окно товаров

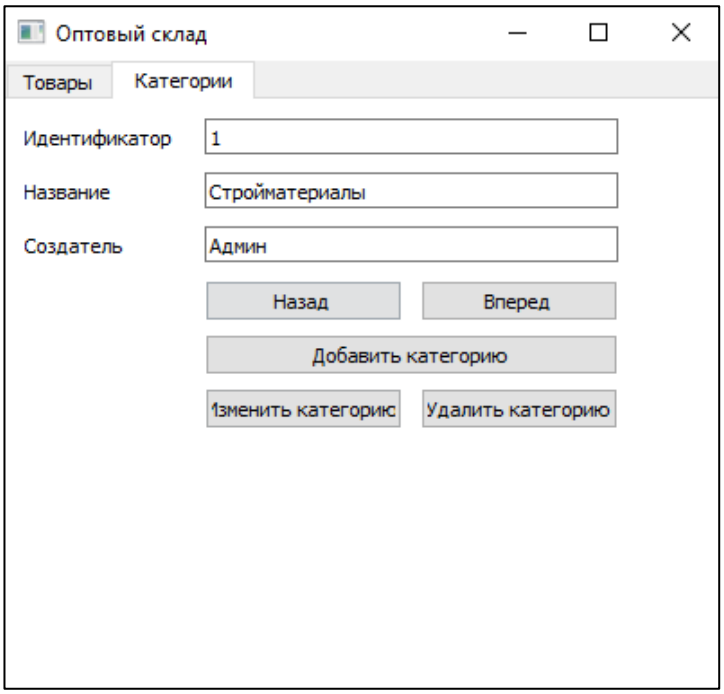

Рисунок 5 – Окно категорий

ЗАКЛЮЧЕНИЕ

В конечном итоге была создана программа для автоматизации рабочей деятельности оптового склада. Было разработано:

- вкладки для переключения между «Товарами» и «Категориями»;
- функции просмотра, добавления, изменения и удаления;
- отображение данных из БД;
- осуществлен выбор категории при создании и изменении продукта из таблицы «Категории» с помощью combobox.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Маккинни, У. Python и анализ данных / У. Маккинни ; перевод с английского А. А. Слинкина. — 2-ое изд., испр. и доп. — Москва : ДМК Пресс, 2020. — 540 с. — ISBN 978-5-97060-590-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/131721 (дата обращения: 20.05.2020). — Режим доступа: для авториз. пользователей.

2. Мартин, О. Байесовский анализ на Python : руководство / О. Мартин ; перевод с английского А. В. Снастина. — Москва : ДМК Пресс, 2020. — 340 с. — ISBN 978-5-97060-768-8. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/140585 (дата обращения: 20.05.2020). — Режим доступа: для авториз. пользователей.

3. Саммерфилд, М. Python на практике : учебное пособие / М. Саммерфилд ; перевод с английского А. А. Слинкин. — Москва : ДМК Пресс, 2014. — 338 с. — ISBN 978-5-97060-095-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/66480 (дата обращения: 20.05.2020). — Режим доступа: для авториз. пользователей.