

Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Комарова Светлана Юриевна
Должность: Проректор по образовательной деятельности
Дата подписания: 30.07.2026 21:16:20
Уникальный программный ключ:
43ba42f5deae4116bbfcb9ac98e391080s12mesk110uad1418e0v31

Приложение

**Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Омский государственный аграрный университет
имени П.А. Столыпина»**

Университетский колледж агробизнеса

09.02.11 Разработка и управление программным обеспечением

**ФОНД ОЦЕНОЧНЫХ СРЕДСТВ
по УП.02.01 Учебная практика**

Обеспечивающее
подразделение

преподавание

дисциплины

Инженерное отделение

Разработчик:

Преподаватель

А.В. Кортусов

**Омск
2026**

СОДЕРЖАНИЕ

1. ОБЩИЕ ПОЛОЖЕНИЯ	3
2. ОЖИДАЕМЫЕ РЕЗУЛЬТАТЫ ИЗУЧЕНИЯ	4
3. РАСПРЕДЕЛЕНИЕ ОЦЕНИВАНИЯ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ И ТИПОВ ОЦЕНОЧНЫХ МАТЕРИАЛОВ ПО ЭЛЕМЕНТАМ ЗНАНИЙ И УМЕНИЙ	5
4. ПОКАЗАТЕЛИ ОЦЕНКИ РЕЗУЛЬТАТОВ ОСВОЕНИЯ УЧЕБНОЙ ПРАКТИКИ	7

1. ОБЩИЕ ПОЛОЖЕНИЯ

1. Фонд оценочных средств (далее – ФОС) предназначен для контроля и оценки образовательных достижений обучающихся, освоивших программу УП 02.01 Учебная практика.
2. ФОС включает оценочные материалы для проведения текущего контроля и промежуточной аттестации в форме зачета .
3. ФОС позволяет оценивать знания, умения, направленные на формирование компетенций
4. ФОС разработан на основании положений основной образовательной программы по специальности 09.02.11 Разработка и управление программным обеспечением и рабочей программы УП 02.01 Учебная практика
5. ФОС является обязательным обособленным приложением к рабочей программе.

2. ОЖИДАЕМЫЕ РЕЗУЛЬТАТЫ ИЗУЧЕНИЯ

Результаты обучения (освоенные умения, навыки)	Показатели оценки образовательных результатов
проектировать модули, соответствующие бизнес задачам	Обучающийся умеет проектировать модули, соответствующие бизнес задачам
определять структуру и интерфейсы модулей, анализировать требования к модулю и определять его функциональность, проектировать архитектуру модуля	Обучающийся умеет определять структуру и интерфейсы модулей, анализировать требования к модулю и определять его функциональность, проектировать архитектуру модуля
создавать диаграммы классов, последовательностей и прочих диаграмм для визуализации проектируемого модуля	Обучающийся умеет создавать диаграммы классов, последовательностей и прочих диаграмм для визуализации проектируемого модуля
разрабатывать модули программного обеспечения с использованием различных языков программирования и технологий	Обучающийся умеет разрабатывать модули программного обеспечения с использованием различных языков программирования и технологий
применять паттерны проектирования и структуры данных для создания эффективных и масштабируемых модулей	Обучающийся умеет применять паттерны проектирования и структуры данных для создания эффективных и масштабируемых модулей
создавать интерфейсы для взаимодействия с другими модулями и системами, обеспечивая безопасность, производительность и масштабируемость при разработке модулей	Обучающийся умеет создавать интерфейсы для взаимодействия с другими модулями и системами, обеспечивая безопасность, производительность и масштабируемость при разработке модулей
интегрировать модули и компоненты, обеспечивая их взаимодействие	Обучающийся умеет интегрировать модули и компоненты, обеспечивая их взаимодействие
работать с API и устанавливать соединения между компонентами	Обучающийся умеет работать с API и устанавливать соединения между компонентами
отслеживать и устранять конфликты и ошибки интеграции	Обучающийся умеет отслеживать и устранять конфликты и ошибки интеграции
проектирования модулей ПО с учетом требований заказчика	Обучающийся владеет навыками проектирования модулей ПО с учетом требований заказчика
создания архитектурных диаграмм и спецификаций модулей	Обучающийся владеет навыками создания архитектурных диаграмм и спецификаций модулей
определения интерфейсов и взаимодействия модулей в системе	Обучающийся владеет навыками определения интерфейсов и взаимодействия модулей в системе
создание модулей программного обеспечения на различных языках программирования	Обучающийся владеет навыками создания модулей программного обеспечения на различных языках программирования
отладки и тестирования разработанных модулей	Обучающийся владеет навыками отладки и тестирования разработанных модулей

применение структурного и объектно-ориентированного программирования	Обучающийся владеет навыками применения структурного и объектно-ориентированного программирования
оптимизации кода и алгоритмов программных модулей для увеличения производительности мониторинга и анализа производительности приложений	Обучающийся владеет навыками оптимизации кода и алгоритмов программных модулей для увеличения производительности мониторинга и анализа производительности приложений
интеграции программных модулей и компонентов в единое программное решение	Обучающийся владеет навыками интеграции программных модулей и компонентов в единое программное решение
работы с интеграционными платформами и инструментами обеспечения совместимости и стабильности системы	Обучающийся владеет навыками работы с интеграционными платформами и инструментами обеспечения совместимости и стабильности системы

3. РАСПРЕДЕЛЕНИЕ ОЦЕНИВАНИЯ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ И ТИПОВ ОЦЕНОЧНЫХ МАТЕРИАЛОВ ПО ЭЛЕМЕНТАМ УМЕНИЙ И НАВЫКОВ

Содержание курса	Форма контроля	Умения	Навыки
Текущий контроль			
Раздел 1. Организационный этап			
Прохождение вводного инструктажа. Получение и обсуждение задания на практику.	Проверка отчета по учебной практике	Уо 01.01, Уо 09.01, Уо 09.02	-
Раздел 2. Основной этап			
Решение прикладных задач:	наблюдение и оценка в процессе практики; анализ отчетной документации; экспертная оценка выполнения индивидуальных заданий.	У 2.1.1, У 2.1.2, У 2.1.3, У 2.2.1, У 2.2.2, У 2.2.3, У 2.3.1, У 2.3.2, У 2.3.3	Н 2.1.1, Н 2.1.2, Н 2.1.3, Н 2.2.1, Н 2.2.2, Н 2.2.3, Н 2.2.4, Н 2.3.1, Н 2.3.2
Раздел 3. Заключительный этап			
Оформление введения. Оформление основной части. Оформление заключения. Оформление списка использованных источников и приложений. Оформление отчета и приложений. Прохождение собеседования (зачет)	Проверка отчета по учебной практике, собеседование	Уо 01.01, Уо 09.01, Уо 09.02	-
Промежуточный контроль			
Зачет	Проверка отчета по учебной практике, собеседование	У 2.1.1, У 2.1.2, У 2.1.3, У 2.2.1, У 2.2.2, У 2.2.3, У 2.3.1, У 2.3.2, У 2.3.3, Уо 01.01, Уо 09.01, Уо 09.02	Н 2.1.1, Н 2.1.2, Н 2.1.3, Н 2.2.1, Н 2.2.2, Н 2.2.3, Н 2.2.4, Н 2.3.1, Н 2.3.2

4. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОЦЕНКИ УМЕНИЙ, НАВЫКОВ

4.1. Оценочные средства, применяемые для текущего контроля.

4.1.1 Разработайте модуль для управления заказами интернет-магазина на Python с использованием SQLite:

1. Создайте классы: `Customer`, `Product`, `Order`, `OrderItem`.
2. Реализуйте CRUD-операции для всех сущностей.
3. Разработайте валидацию данных (email, телефон, цена).
4. Реализуйте бизнес-логику:
 - a. Создание заказа с автоматическим расчетом суммы.
 - b. Проверка наличия товаров на складе.
 - c. Изменение статуса заказа (новый, в обработке, отправлен, доставлен).

Пример кода:

models.py

```
from dataclasses import dataclass
from typing import Optional, List
import sqlite3
from datetime import datetime
import re

@dataclass
class Customer:
    id: Optional[int] = None
    full_name: str = ""
    email: str = ""
    phone: str = ""
    address: str = ""

@dataclass
class Product:
    id: Optional[int] = None
    name: str = ""
    description: str = ""
    price: float = 0.0
    stock: int = 0
    category: str = ""
```

```

@dataclass
class Order:
    id: Optional[int] = None
    customer_id: int = 0
    order_date: str = ""
    status: str = "НОВЫЙ"
    total_amount: float = 0.0

@dataclass
class OrderItem:
    id: Optional[int] = None
    order_id: int = 0
    product_id: int = 0
    quantity: int = 0
    price_at_time: float = 0.0

class Validator:
    @staticmethod
    def validate_email(email: str) -> bool:
        return bool(re.match(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$', email))

    @staticmethod
    def validate_phone(phone: str) -> bool:
        return bool(re.match(r'^\+?[0-9]{10,15}$', phone))

    @staticmethod
    def validate_price(price: float) -> bool:
        return price > 0

    @staticmethod
    def validate_stock(stock: int) -> bool:
        return stock >= 0

class OrderRepository:
    def __init__(self, db_path='store.db'):

```

```
self.db_path = db_path
self._init_db()
```

```
def _init_db(self):
```

```
    with sqlite3.connect(self.db_path) as conn:
```

```
        conn.execute("""
```

```
            CREATE TABLE IF NOT EXISTS customers (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                full_name TEXT NOT NULL,
                email TEXT UNIQUE NOT NULL,
                phone TEXT,
                address TEXT
```

```
            )
```

```
        """)
```

```
        conn.execute("""
```

```
            CREATE TABLE IF NOT EXISTS products (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                description TEXT,
                price REAL NOT NULL,
                stock INTEGER NOT NULL,
                category TEXT
```

```
            )
```

```
        """)
```

```
        conn.execute("""
```

```
            CREATE TABLE IF NOT EXISTS orders (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                customer_id INTEGER NOT NULL,
                order_date TEXT NOT NULL,
                status TEXT DEFAULT 'новый',
                total_amount REAL NOT NULL,
                FOREIGN KEY (customer_id) REFERENCES customers(id)
```

```
            )
```

```
        """)
```

```

conn.execute("""
    CREATE TABLE IF NOT EXISTS order_items (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        order_id INTEGER NOT NULL,
        product_id INTEGER NOT NULL,
        quantity INTEGER NOT NULL,
        price_at_time REAL NOT NULL,
        FOREIGN KEY (order_id) REFERENCES orders(id),
        FOREIGN KEY (product_id) REFERENCES products(id)
    )
""")

```

Методы для работы с товарами

```

def add_product(self, product: Product) -> Optional[int]:
    if not Validator.validate_price(product.price):
        return None
    if not Validator.validate_stock(product.stock):
        return None

    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.execute("""
            INSERT INTO products (name, description, price, stock, category)
            VALUES (?, ?, ?, ?, ?)
            """, (product.name, product.description, product.price, product.stock,
product.category))
        return cursor.lastrowid

def get_product(self, product_id: int) -> Optional[Product]:
    with sqlite3.connect(self.db_path) as conn:
        row = conn.execute('SELECT * FROM products WHERE id = ?',
(product_id,)).fetchone()
        return Product(*row) if row else None

def update_stock(self, product_id: int, quantity: int) -> bool:

```

```

with sqlite3.connect(self.db_path) as conn:
    cursor = conn.execute('UPDATE products SET stock = stock - ? WHERE id = ? AND
stock >= ?',
                            (quantity, product_id, quantity))
    return cursor.rowcount > 0

```

Метод создания заказа

```

def create_order(self, customer_id: int, items: List[dict]) -> Optional[int]:
    """Создание заказа со списком товаров: [{'product_id': 1, 'quantity': 2}]"""
    total = 0
    order_items = []

```

Проверка наличия товаров и расчет суммы

```

for item in items:
    product = self.get_product(item['product_id'])
    if not product or product.stock < item['quantity']:
        return None
    total += product.price * item['quantity']
    order_items.append({
        'product_id': product.id,
        'quantity': item['quantity'],
        'price_at_time': product.price
    })

```

```

with sqlite3.connect(self.db_path) as conn:

```

Создание заказа

```

    cursor = conn.execute("""
        INSERT INTO orders (customer_id, order_date, status, total_amount)
        VALUES (?, ?, ?, ?)
    """, (customer_id, datetime.now().isoformat(), 'новый', total))
    order_id = cursor.lastrowid

```

Добавление позиций заказа

```

for item in order_items:

```

```

conn.execute("""
    INSERT INTO order_items (order_id, product_id, quantity, price_at_time)
    VALUES (?, ?, ?, ?)
    """, (order_id, item['product_id'], item['quantity'], item['price_at_time']))

# Обновление остатков
conn.execute('UPDATE products SET stock = stock - ? WHERE id = ?',
              (item['quantity'], item['product_id']))

conn.commit()
return order_id

def update_order_status(self, order_id: int, status: str) -> bool:
    valid_statuses = ['новый', 'в обработке', 'отправлен', 'доставлен', 'отменен']
    if status not in valid_statuses:
        return False

    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.execute('UPDATE orders SET status = ? WHERE id = ?', (status,
order_id))
        return cursor.rowcount > 0

def get_orders_by_customer(self, customer_id: int) -> List[Order]:
    with sqlite3.connect(self.db_path) as conn:
        rows = conn.execute('SELECT * FROM orders WHERE customer_id = ? ORDER
BY order_date DESC',
                             (customer_id,)).fetchall()
        return [Order(*row) for row in rows]

def get_order_details(self, order_id: int) -> dict:
    """Получение полной информации о заказе"""
    with sqlite3.connect(self.db_path) as conn:
        # Информация о заказе

```

```

        order = conn.execute('SELECT * FROM orders WHERE id = ?',
(order_id,)).fetchone()

        if not order:
            return None

        # Позиции заказа
        items = conn.execute("""
            SELECT oi.*, p.name, p.category
            FROM order_items oi
            JOIN products p ON oi.product_id = p.id
            WHERE oi.order_id = ?
        """, (order_id,)).fetchall()

        # Информация о клиенте
        customer = conn.execute('SELECT * FROM customers WHERE id = ?',
(order[1],)).fetchone()

        return {
            'order': Order(*order),
            'customer': Customer(*customer),
            'items': [dict(row) for row in items],
            'total': order[4]
        }

```

Форма выполнения: Исходный код модуля, скриншоты работы, примеры использования.

4.1.2. Разработка REST API для модуля

Создайте REST API для управления заказами с использованием Flask:

api.py

```

from flask import Flask, request, jsonify
from models import OrderRepository, Customer, Product, Order
import logging

app = Flask(__name__)

```

```
repo = OrderRepository()
```

```
logging.basicConfig(level=logging.INFO)
```

```
@app.route('/api/products', methods=['GET'])
```

```
def get_products():
```

```
    """Получение списка товаров"""
```

```
    with sqlite3.connect('store.db') as conn:
```

```
        rows = conn.execute('SELECT * FROM products').fetchall()
```

```
        products = [{ 'id': r[0], 'name': r[1], 'description': r[2],  
                        'price': r[3], 'stock': r[4], 'category': r[5]} for r in rows]  
    return jsonify(products)
```

```
@app.route('/api/products', methods=['POST'])
```

```
def add_product():
```

```
    """Добавление товара"""
```

```
    data = request.json
```

```
    product = Product(
```

```
        name=data.get('name'),
```

```
        description=data.get('description', ''),
```

```
        price=float(data.get('price', 0)),
```

```
        stock=int(data.get('stock', 0)),
```

```
        category=data.get('category', ''))
```

```
    )
```

```
    product_id = repo.add_product(product)
```

```
    if product_id:
```

```
        return jsonify({'id': product_id, 'message': 'Товар добавлен'}), 201
```

```
    return jsonify({'error': 'Ошибка валидации'}), 400
```

```
@app.route('/api/orders', methods=['POST'])
```

```
def create_order():
```

```
    """Создание заказа"""
```

```
    data = request.json
```

```
    customer_id = data.get('customer_id')
```

```

items = data.get('items', [])

if not customer_id or not items:
    return jsonify({'error': 'Не указан customer_id или items'}), 400

order_id = repo.create_order(customer_id, items)
if order_id:
    return jsonify({'order_id': order_id, 'message': 'Заказ создан'}), 201
return jsonify({'error': 'Ошибка создания заказа (недостаточно товаров)'}), 400

@app.route('/api/orders/<int:order_id>', methods=['GET'])
def get_order(order_id):
    """Получение информации о заказе"""
    details = repo.get_order_details(order_id)
    if details:
        return jsonify({
            'order_id': details['order'].id,
            'customer': details['customer'].full_name,
            'status': details['order'].status,
            'total': details['total'],
            'items': [{
                'product': item['name'], 'quantity': item['quantity'],
                'price': item['price_at_time']} for item in details['items']]
        })
    return jsonify({'error': 'Заказ не найден'}), 404

@app.route('/api/orders/<int:order_id>/status', methods=['PUT'])
def update_order_status(order_id):
    """Обновление статуса заказа"""
    status = request.json.get('status')
    if repo.update_order_status(order_id, status):
        return jsonify({'message': f'Статус обновлен на {status}'})
    return jsonify({'error': 'Ошибка обновления статуса'}), 400

@app.route('/api/customers/<int:customer_id>/orders', methods=['GET'])

```

```
def get_customer_orders(customer_id):
    """Получение заказов клиента"""
    orders = repo.get_orders_by_customer(customer_id)
    return jsonify([
        {'id': o.id,
         'date': o.order_date,
         'status': o.status,
         'total': o.total_amount
        } for o in orders])

if __name__ == '__main__':
    app.run(debug=True)
```

Форма выполнения: Исходный код API, скриншоты запросов через Postman.

4.1.3. Тестирование и отладка API

Разработайте тесты для API с использованием PyTest:

```
# test_api.py

import pytest
import json
from api import app

@pytest.fixture
def client():
    app.config['TESTING'] = True
    with app.test_client() as client:
        yield client

def test_add_product(client):
    """Тест добавления товара"""
    response = client.post('/api/products', json={
        'name': 'Ноутбук',
        'description': 'Игровой ноутбук',
        'price': 150000,
        'stock': 10,
```

```

        'category': 'Электроника'
    })
    assert response.status_code == 201
    data = json.loads(response.data)
    assert 'id' in data

def test_create_order(client):
    """Тест создания заказа"""
    # Сначала добавим товар
    client.post('/api/products', json={
        'name': 'Телефон',
        'price': 50000,
        'stock': 5
    })

    response = client.post('/api/orders', json={
        'customer_id': 1,
        'items': [{'product_id': 1, 'quantity': 2}]
    })
    assert response.status_code == 201
    data = json.loads(response.data)
    assert 'order_id' in data

def test_get_order(client):
    """Тест получения заказа"""
    # Создаем заказ
    client.post('/api/products', json={'name': 'Планшет', 'price': 30000, 'stock': 3})
    create_resp = client.post('/api/orders', json={
        'customer_id': 1,
        'items': [{'product_id': 1, 'quantity': 1}]
    })
    order_id = json.loads(create_resp.data)['order_id']

    response = client.get(f'/api/orders/{order_id}')

```

```

assert response.status_code == 200
data = json.loads(response.data)
assert data['order_id'] == order_id

```

```

def test_update_order_status(client):
    """Тест обновления статуса заказа"""
    client.post('/api/products', json={'name': 'Мышь', 'price': 2000, 'stock': 10})
    create_resp = client.post('/api/orders', json={
        'customer_id': 1,
        'items': [{'product_id': 1, 'quantity': 1}]
    })
    order_id = json.loads(create_resp.data)['order_id']

    response = client.put(f'/api/orders/{order_id}/status', json={'status': 'отправлен'})
    assert response.status_code == 200

```

Форма выполнения: Код тестов, отчет о результатах тестирования.

4.1.4. Интеграция с внешними сервисами

Реализуйте отправку уведомлений при изменении статуса заказа:

notifications.py

```

import smtplib
from email.mime.text import MIMEText
import logging

logger = logging.getLogger(__name__)

```

```

class NotificationService:
    @staticmethod
    def send_email(to_email: str, subject: str, body: str) -> bool:
        """Отправка email-уведомления"""
        try:
            msg = MIMEText(body, 'plain', 'utf-8')
            msg['Subject'] = subject
            msg['From'] = 'noreply@store.com'
            msg['To'] = to_email

```

```

# Пример настройки SMTP (для реального использования)
# server = smtplib.SMTP('smtp.gmail.com', 587)
# server.starttls()
# server.login('login', 'password')
# server.send_message(msg)
# server.quit()

logger.info(f"Уведомление отправлено на {to_email}: {subject}")
return True
except Exception as e:
    logger.error(f"Ошибка отправки уведомления: {e}")
    return False

# Добавление в метод обновления статуса
def update_order_status_with_notification(self, order_id: int, status: str) -> bool:
    """Обновление статуса с отправкой уведомления"""
    if not self.update_order_status(order_id, status):
        return False

    # Получение email клиента
    details = self.get_order_details(order_id)
    if details:
        NotificationService.send_email(
            details['customer'].email,
            f"Статус заказа #{order_id} изменен",
            f"Ваш заказ #{order_id} теперь имеет статус: {status}"
        )
    return True

```

Форма выполнения: Исходный код интеграции, логи отправленных уведомлений.

4.2. Оценочные средства, применяемые для промежуточной аттестации по итогам изучения дисциплины

Зачет проводится по завершении учебной практики на последнем аудиторном занятии.

Промежуточная аттестация в форме зачета осуществляется по результатам сдачи отчета по практике и с учетом текущего контроля успеваемости при выполнении всех видов текущего контроля.

Обучающиеся, не выполнившие виды работ, предусмотренные рабочей программой; пропустившие более 50% практики без уважительной причины, не допускаются к зачету.

Промежуточная аттестация таких лиц проводится только после прохождения ими всех видов текущего контроля.

V. ПОКАЗАТЕЛИ ОЦЕНКИ РЕЗУЛЬТАТОВ ОСВОЕНИЯ УЧЕБНОЙ ПРАКТИКИ

Уровень сформированности компетенций	Оценка	Критерии оценивания по видам работ	
		тестирование (процент правильных ответов)	прочие виды работ
Высокий	Отлично	90-100%	Обучающийся глубоко и прочно усвоил теоретический и освоил практический материал. Дает логичные и грамотные ответы. Демонстрирует знание не только основного, но и дополнительного материала, быстро ориентируется, отвечая на дополнительные вопросы. Свободно справляется с поставленными задачами, аргументировано и верно обосновывает принятые решения.
Повышенный	Хорошо	70-89%	Обучающийся твердо знает программный материал, грамотно и по существу излагает его. Не допускает существенных неточностей при ответах на вопросы, правильно применяет теоретические положения при решении практических задач, владеет навыками и приемами их выполнения.
Базовый	Удовлетворительно	50-69%	Обучающийся демонстрирует знания только основного материала, но не усвоил его детали, испытывает затруднения при решении практических задач. В ответах на поставленные вопросы допускает неточности. Дает определения понятий, искажающие их смысл. Нарушает последовательность изложения программного материала.
Не сформирована	Неудовлетворительно	0-49%	Обучающийся не знает, не выполняет или неправильно выполняет большую часть учебного материала. Допускает ошибки в формулировке определений, искажающие их смысл, беспорядочно и неуверенно излагает материал. Ответы на дополнительные вопросы отсутствуют. Не выполняет задания.

ЛИСТ РАССМОТРЕНИЙ И ОДОБРЕНИЙ
рабочей программы учебной практики
УП.02.01 Учебная практика
в составе ООП 09.02.11 Разработка и управление программным обеспечением

1) <u>Рассмотрена и одобрена:</u>
а) На заседании предметно-цикловой методической комиссии протокол № 6 от 10.03.2026 г. Председатель ПЦМК  С.М. Нурбаева
б) На заседании методического совета протокол №4 от 21.04.2026 г. Председатель методического совета  М.В. Иваницкая
2) <u>Рассмотрена и одобрена внешним экспертом</u>
а) директор ООО «Сатори Партнер» А.Б. Мальцев