**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**АРХАНГЕЛЬСКИЙ КОЛЛЕДЖ ТЕЛЕКОММУНИКАЦИЙ**

**ИМ. Б.Л. РОЗИНГА (ФИЛИАЛ) СПбГУТ**

**(АКТ (ф) СПбГУТ)**

СОГЛАСОВАНО

Рук. предприятия

\_\_\_\_\_\_\_\_\_\_\_ А.В. Кудрявцев

(Подпись) (И.О. Фамилия)

«30» мая 2025г.

**ТЕХНИЧЕСКИЙ ОТЧЁТ**

**по ПМ.11, ПМ.01**

|  |
| --- |
| ООО ПКП «ТИТАН» |
| Информационные системы и программирование |
| 09.02.07. 25ТО01. 015 ПЗ |
| (Обозначение документа) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИСПП-21 | |  | 30.05.25 | А.А. Маратканов |
|  | (Группа) | | (Подпись) | (Дата) | (И.О. Фамилия) |
| Рук. практики от предприятия | | |  | 30.05.25 | А.Л. Аникиев |
|  | |  | (Подпись) | (Дата) | (И.О. Фамилия) |

Архангельск 2025

Содержание

[Перечень сокращений и обозначений 3](#_Toc201306122)

[Введение 4](#_Toc201306123)

[1 Охрана труда и техника безопасности при работе на ПК 6](#_Toc201306124)

[1.1 Требования к рабочему месту 6](#_Toc201306125)

[1.2 Безопасность труда при работе за ПК 6](#_Toc201306126)

[1.3 Ответственность 6](#_Toc201306127)

[2 Выполнение работ по ПМ.11 7](#_Toc201306128)

[2.1 Проектирование базы данных 7](#_Toc201306129)

[2.2 Разработка базы данных и объектов базы данных 9](#_Toc201306130)

[2.3 Администрирование и защита базы данных 10](#_Toc201306131)

[3 Выполнение работ по ПМ.01 12](#_Toc201306132)

[3.1 Проектирование программного обеспечения 12](#_Toc201306133)

[3.2 Разработка программных модулей 12](#_Toc201306134)

[3.3 Разработка мобильного приложения 14](#_Toc201306135)

[3.4 Отладка и тестирование программных модулей 15](#_Toc201306136)

[3.5 Оптимизация и рефакторинг программного кода 17](#_Toc201306137)

[Заключение 19](#_Toc201306138)

[Список использованных источников 20](#_Toc201306139)

Перечень сокращений и обозначений

В настоящем техническом отчете применяются следующие сокращения и обозначения:

БД – база данных

ПК – персональный компьютер

ПО – программное обеспечение

СУБД — система управления базами данных

ТЗ – техническое задание

API – программный интерфейс приложения

ERD – диаграмма «сущность-связь»

HTML – язык разметки гипертекста

IDE –интегрированная среда разработки

SQL – язык структурированных запросов

UML – унифицированный язык моделирования

Введение

Базой производственной практики является ООО ПКП «Титан». Предприятие занимается:

* лесозаготовка – основное направление деятельности холдинга;
* услуги гостеприимства;
* сфера управления коммерческой недвижимостью;
* закупки – одно из самых важных направлений работы группы компаний «Титан», которое обеспечивает материальную базу для всей хозяйственной деятельности холдинга.

Цели производственной практики является:

* получение практического опыта по выполнению работ по ПМ.11 «Разработка, администрирование и защита баз данных» и развитие общих и профессиональных компетенций;
* получение практического опыта по выполнению работ по ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем» и развитие общих и профессиональных компетенций.

Задачами производственной практики являются:

* формирование алгоритма разработки программных модулей с ТЗ;
* разработка программного модуля в соответствии с ТЗ;
* выполнение отладки программных модулей с использованием специализированных программных средств;
* выполнение тестирование программных модулей;
* осуществление рефакторинга и оптимизации программного кода;
* разработка модулей ПО для мобильных платформ;
* осуществление сбора, обработки, анализа информации для проектирования БД;
* проектирование БД на основе анализа предметной области;
* разработка объектов БД в соответствии с результатами анализа предметной области;
* реализовать БД в конкретной СУБД;
* администрирование БД;
* защита информации в БД с использованием технологии защиты информации.

Для практикантов предоставляется рабочее место с персональным компьютером и всем необходимым для работы аппаратным и программным обеспечением:

* процессор: Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHz;
* системная плата: VivoBook Flip 14\_ASUS Flip TP412UA;
* видеокарта: встроенная;
* оперативная память – 12 ГБ;
* операционная система: Microsoft Windows 10 Pro;
* прикладное ПО: пакет Microsoft Office, Google chrome, Visual Studio 2022, Visual Studio Code, Microsoft SQL Server Management Studio 18, draw.io.

# Охрана труда и техника безопасности при работе на ПК

## **Требования к рабочему месту**

Рабочее место сотрудника, работающего за ПК, должно соответствовать следующим требованиям:

* эргономичное размещение стола, кресла и монитора (верхняя граница экрана – на уровне глаз);
* расстояние от глаз до экрана – от 50 до 70 см;
* наличие естественного или регулируемого искусственного освещения;
* кресло с регулируемой высотой и поддержкой спины;
* организация перерывов – каждые 1 час работы за ПК должен сопровождаться перерывом от 5 до 10 минут.

## **Безопасность труда при работе за ПК**

Сотрудники обязаны:

* проходить вводный и повторный инструктаж по охране труда;
* соблюдать режим труда и отдыха;
* не перегружать зрение, чередовать работу с ПК с другими задачами;
* использовать сертифицированное оборудование и ПО;
* немедленно сообщать руководству о неисправностях оборудования.

## **Ответственность**

Сотрудники несут дисциплинарную и административную ответственность за несоблюдение правил охраны труда. Руководители подразделений обязаны контролировать соблюдение норм охраны труда и техники безопасности.

# Выполнение работ по ПМ.11

## Проектирование базы данных

ООО ПКП «ТИТАН» требуется добавить возможность заказа номенклатур заказчиком и сопоставление номенклатур с поставщиком.

В БД требуется хранить информацию о заказах, оформленных заказчиками. Каждая номенклатура имеет свой уникальный номер, наименование, наименование для печати, вид, единицы измерения.

Список поставщиков содержит их код поставщика и название (уникальны), список заказчиков содержит уникальный код раздела и его название (уникально).

На рисунке 1 показана концептуальная модель предметной области в виде ERD [1], созданная с помощью средства проектирования Draw.io.



Рисунок 1 – Концептуальная модель

На рисунке 2 показана логическая модель [2] предметной области, созданная с помощью средства проектирования Draw.io.

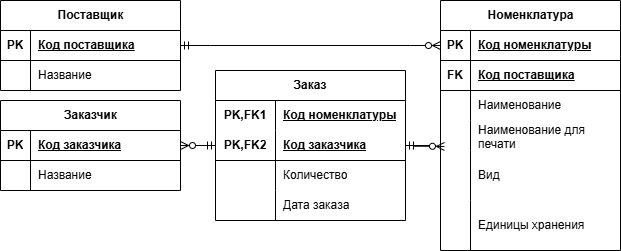


Рисунок 2 – Логическая модель

На рисунке 3 показана физическая модель предметной области, созданная с помощью средства проектирования MySQL WorkBench.

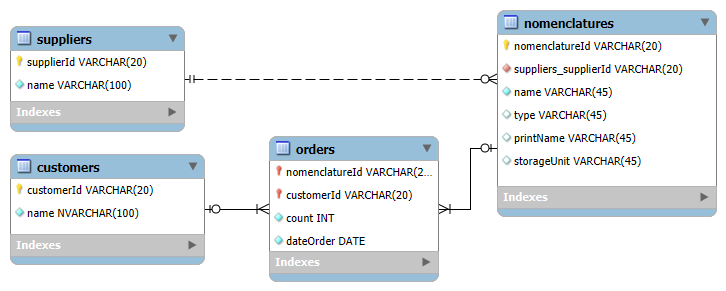


Рисунок 3 – Физическая модель

## Разработка базы данных и объектов базы данных

Описание словаря данных и ограничений целостности [2] для таблицы «nomenclatures» представлен в таблице 1.

Таблица 1 – Словарь данных для таблицы «nomenclatures»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ключ** | **Поле** | **Тип данных** | **Обязательное** | **Примечание** |
| nomenclatures | | | | |
| PK | nomenclatureId | varchar(20) | + |  |
| FK | supplierId | varchar(20) | + |  |
|  | name | nvarchar(100) | + | Уникально |
|  | printName | nvarchar(100) |  |  |
|  | type | nvarchar(20) |  |  |
|  | storageUnit | nvarchar(10) |  | По умолчанию (Шт) |

SQL-запрос для создания таблицы nomenclatures представлен в листинге 1.

Листинг 1 – SQL-запрос для создания таблицы nomenclatures

--Создание таблицы nomenclatures с полями и первичным ключом

CREATE TABLE IF NOT EXISTS `mydb`.`nomenclatures` (

`nomenclatureId` VARCHAR(20) NOT NULL,

`suppliers\_supplierId` VARCHAR(20) NOT NULL,

`name` VARCHAR(45) NOT NULL,

`type` VARCHAR(45) NULL,

`printName` VARCHAR(45) NULL,

`storageUnit` VARCHAR(45) NULL,

PRIMARY KEY (`nomenclatureId`),

UNIQUE INDEX `name\_UNIQUE` (`name` ASC) VISIBLE,

INDEX `fk\_nomenclatures\_suppliers\_idx` (`suppliers\_supplierId` ASC) VISIBLE,

CONSTRAINT `fk\_nomenclatures\_suppliers`

FOREIGN KEY (`suppliers\_supplierId`)

REFERENCES `mydb`.`suppliers` (`supplierId`)

ON DELETE NO ACTION

ON UPDATE CASCADE)

ENGINE = InnoDB

SQL-запрос для создания представления orderDetails с отображением деталей заказа представлен в листинге 2.

Листинг 2 – SQL-запрос для создания представления orderDetails

--Создание представления orderDetails с деталями заказа

CREATE VIEW `orderDetails` AS

SELECT

n.name AS nomenclature,

c.name AS customer,

nc.count,

nc.dateOrder

FROM nomenclatures n

JOIN nomenclatures\_has\_customers nc

ON n.nomenclatureId = nc.nomenclatures\_nomenclatureId

JOIN customers c

ON c.customerId = nc.customers\_customerId

WHERE nc.dateOrder >= '2025-01-01';

## Администрирование и защита базы данных

Во избежание прецедентов с БД используется разграничения доступа к командам БД с использованием создания ролей представленного в листинге 3.

Листинг 3 – SQL-запрос для создания ролей

--Создание ролей в бД

CREATE ROLE 'role\_reader';

GRANT SELECT ON mydb.\* TO 'role\_reader';

CREATE ROLE 'role\_editor';

GRANT SELECT, INSERT, UPDATE, DELETE ON mydb.\* TO 'role\_editor';

CREATE ROLE 'role\_admin';

GRANT ALL PRIVILEGES ON mydb.\* TO 'role\_admin';

Для обеспечения централизованного и безопасного управления правами доступа, участия в администрировании и обеспечения соблюдения политики безопасности информации в информационной системе. Для каждого пользователя задаются уникальные логин и пароль, после чего осуществляется привязка к нужной роли представлено в листинге 4.

Листинг 4 – SQL-запрос для создания пользователей и назначение ролей

--Создание пользователей в бД и назначение ролей

CREATE USER 'reader'@'%' IDENTIFIED BY 'reader\_password';

GRANT 'role\_reader' TO 'reader'@'%';

SET DEFAULT ROLE 'role\_reader' TO 'reader'@'%';

CREATE USER 'editor'@'%' IDENTIFIED BY 'editor\_password';

GRANT 'role\_editor' TO 'editor'@'%';

SET DEFAULT ROLE 'role\_editor' TO 'editor'@'%';

CREATE USER 'admin'@'%' IDENTIFIED BY 'admin\_password';

GRANT 'role\_admin' TO 'admin'@'%';

SET DEFAULT ROLE 'role\_admin' TO 'admin'@'%';

Во избежание потери данных необходимо периодически выполнять полное резервное копирование БД. Резервные копии позволяют восстановить данные после сбоя и других непредвиденных проблем. Для выполнения резервного копирования БД требуется выполнить SQL-скрипт, представленный листингом 5.

Листинг 5 – Код для создания резервной копии БД

--Выполнение резервного копирования

mysqldump -u root -p mydb > "D:/Backups/mydb\_backup.sql

Для восстановления данных из созданной резервной копии требуется выполнить SQL-скрипт, представленный листингом 6.

Листинг 6 – Код для восстановления из резервной копии БД

--Восстановление из резервной копии

mysql -u root -p mydb < "D:/Backups/mydb\_backup.sql"

# Выполнение работ по ПМ.01

## Проектирование программного обеспечения

Предприятием поставлена задача по разработке системы поиска и создание заказа, для закупок.

Систему требуется реализовывать как автономный сервис.

Создать программу для поиска – программа преобразует запрос в эмбеддинг, передает его в API, которое находит подходящие номенклатуры, передает в программу, программа создает заказ.

Действия доступные пользователю отображены на диаграмме прецедентов, предоставленной на рисунке 4.

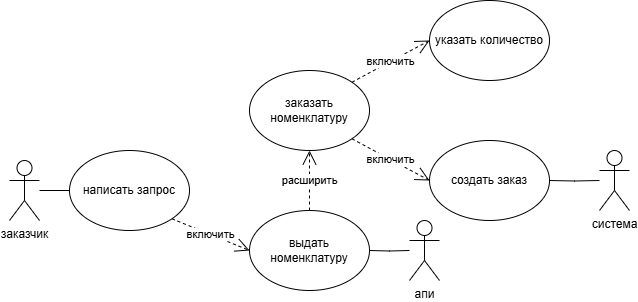


Рисунок 4– Диаграмма прецедентов

## Разработка программных модулей

Во время производственной практики для создания API был использован язык разработки python 3.10 [3], и библиотека flask [4].

В программе реализована возможность поиска номенклатур из БД, возможность указать количество заказываемой номенклатуры. Сопоставление запроса с индексом выполняет функция search\_similar, представленная в листинге 7.

Листинг 7 – Код функции search\_similar

# Основной маршрут программы

@app.route("/faiss/search", methods=["POST"])

def search\_similar():

# Получение JSON-данных

data = request.get\_json()

query = data.get("query", "")

top\_k = data.get("top\_k", 10)

# Проверка на пустой запрос

if not query or faiss\_index is None:

return jsonify({"error": "Ошибка запроса или индекс не найден"}), 400

# Нормализация запроса

normalize\_query = normalize(query)

print(f"Получен запрос на поиск: {normalize\_query}")

# Генерация эмбеддинга

embedding = model.encode([normalize\_query])[0].astype("float32")

with index\_lock:

D, I = faiss\_index.search(np.array([embedding]), top\_k)

results = []

for idx, dist in zip(I[0], D[0]):

print(f"Индекс: {idx}, Расстояние: {dist}")

idx\_str = str(idx)

meta = nomenclature\_map.get(idx\_str)

if meta:

print(f"Найдено соответствие для индекса {idx\_str}: {meta}")

results.append({

"nomenclatureId": meta["nomenclatureId"],

"nomenclature": meta["nomenclature"],

"distance": float(dist)

})

else:

print(f"Нет соответствия для индекса {idx\_str}")

if not results:

print("Не найдено ни одного соответствия")

# Возвращение результата

return jsonify({"results": results})

Страница для пользования API представлена на рисунке 5

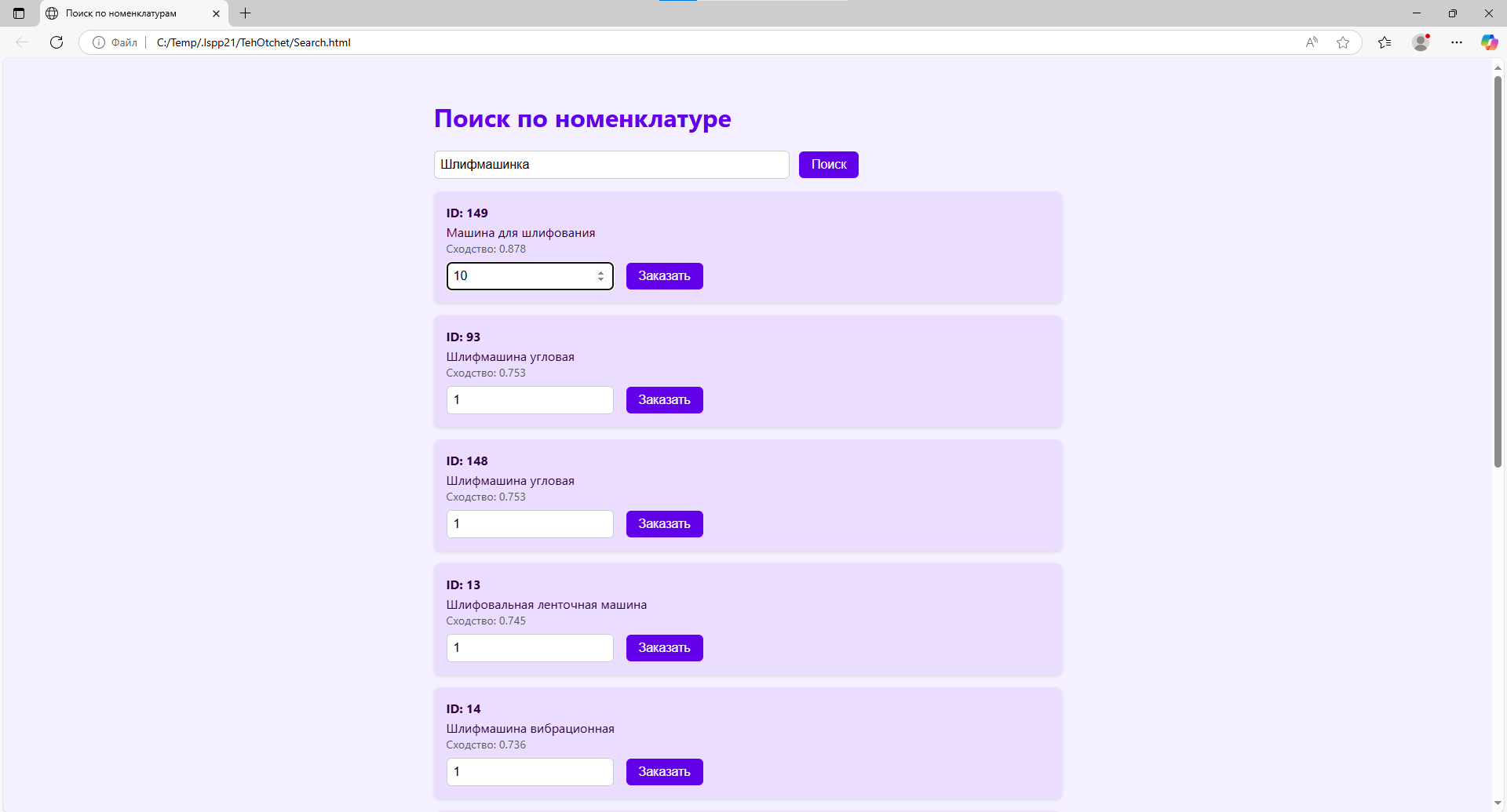


Рисунок 5 – Поиск номенклатур. Вид главной формы

## Разработка мобильного приложения

Предприятием поставлена задача разработать мобильное приложение для использования, разработанного API, с целью упрощения взаимодействия сотрудников с системой учета номенклатуры и оформления заказов. Приложение должно обеспечивать возможность быстрого поиска номенклатур по наименованию, получая актуальную информацию с сервера, а также оформлять заказ с указанием необходимого количества.

Во время производственной практики для разработки мобильного приложения использовался язык программирования Kotlin [5] в среде разработки Android Studio.

Интерфейс мобильного приложения включает в себя поля для ввода поискового запроса, отображение списка найденных номенклатур и форму для оформления заказа. Пример интерфейса представлен на рисунке 6.

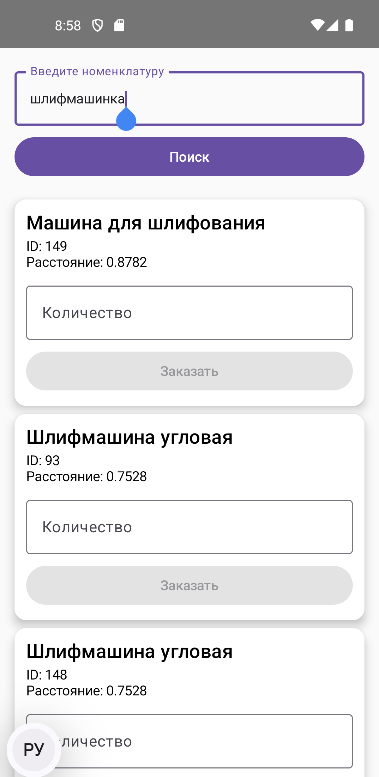


Рисунок 6 – Поиск номенклатур. Вид главной формы

## Отладка и тестирование программных модулей

Для проверки работы приложения необходимо провести тестирование.

Во время работы требовалось провести 3 теста по методу «черного ящика» представлены в таблице 2.

Таблица 2 – Набор тестов приложения

|  |  |  |
| --- | --- | --- |
| Действие | Ожидаемый результат | Полученный результат |
| Ввести в поле поиска номенклатуру и нажать кнопку поиска | Выводятся список похожих по смыслу номенклатур | Соответствует ожидаемому |
| Изменить количество заказываемой номенклатуры с помощью поля количества | Количество успешно изменилось | Соответствует ожидаемому |
| Нажать на кнопку заказать номенклатуру | Номенклатура заказана | Соответствует ожидаемому |

Помимо тестирования по методологии «чёрного ящика» требуется провести автоматизированное тестирование с помощью библиотеки python unittest.

В результате тестирования выполнена проверка на корректность работы поиска с запросом «болгарка» количество совпадений «5». Код автоматизированного теста приведён в листинге 8.

Листинг 8 – Модульный тест

import unittest

import requests

BASE\_URL = "http://127.0.0.1:5005"

class TestFaissSearchAPI(unittest.TestCase):

# Тест поиска по запросу

def test\_successful\_search(self):

response = requests.post(f"{BASE\_URL}/faiss/search", json={

"query": "болгарка",

"top\_k": 5

})

self.assertEqual(response.status\_code, 200)

data = response.json()

self.assertIn("results", data)

self.assertGreater(len(data["results"]), 0)

for item in data["results"]:

self.assertIn("nomenclatureId", item)

self.assertIn("nomenclature", item)

self.assertIn("distance", item)

# Тест обработки пустого запроса

def test\_empty\_query(self):

response = requests.post(f"{BASE\_URL}/faiss/search", json={

"query": ""

})

self.assertEqual(response.status\_code, 400)

# Тест отсутствия поля query

def test\_missing\_query\_field(self):

response = requests.post(f"{BASE\_URL}/faiss/search", json={})

self.assertEqual(response.status\_code, 400)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

Для отладки приложения необходимо использовать пошаговое выполнение кода с остановками на точках останова. Представлен на рисунке 7.



Рисунок 7 – Visual studio code. Фрагмент кода с точками остановки

## Оптимизация и рефакторинг программного кода

В ходе работы над практической практикой над API выполнен рефакторинг и оптимизация функции search\_similar, отвечающей за поиск номенклатуры по смыслу с использованием FAISS. Основными изменениями цели стали повышение читаемости кода, улучшение производительности и обеспечение устойчивости к возможным ошибкам при выполнении запроса.

Добавлена проверка входных данных – теперь функция корректно обрабатывает случаи запроса или незагруженного FAISS-индекса, возвращающего понятное сообщение с соответствующим HTTP-статусом.

Для повышения устойчивости была реализована обработка исключений при вызове кодирования модели, чтобы избежать аварийного выполнения процедуры при выполнении внутренних ошибок.

Таким образом, функция search\_similar стала более надежной, читаемой и устойчивой к ошибкам, что обеспечивает общее качество и стабильность работы API-сервиса при эксплуатации в среде многих пользователей.

Листинг 9 – Функция search\_similar выполняет поиск наиболее похожих номенклатур

# Основной маршрут программы

@app.route("/faiss/search", methods=["POST"])

def search\_similar():

# Получение JSON-данных

payload = request.get\_json()

query = payload.get("query")

top\_k = payload.get("top\_k", 10)

# Проверка на пустой запрос

if not query or faiss\_index is None:

logger.warning("Некорректный запрос или индекс не загружен")

return jsonify({"error": "Некорректный запрос или индекс не загружен"}), 400

# Нормализация запроса

normalized = normalize(query)

logger.info(f"Запрос: '{normalized}'")

# генерация эмбеддинга

try:

embedding = model.encode([normalized])[0].astype("float32")

except Exception as e:

logger.error(f"Ошибка при кодировании: {e}")

return jsonify({"error": "Ошибка при получении эмбеддинга"}), 500

# Поиск вектора в индексе

with index\_lock:

D, I = faiss\_index.search(np.array([embedding]), top\_k)

results = []

for idx, dist in zip(I[0], D[0]):

meta = nomenclature\_map.get(str(idx))

if meta:

results.append({

"nomenclatureId": meta["nomenclatureId"],

"nomenclature": meta["nomenclature"],

"distance": float(dist)

})

else:

logger.debug(f"Нет данных для индекса {idx}")

if not results:

logger.info("Совпадений не найдено")

return jsonify({"results": results})

Заключение

Производственная практика на предприятии ООО ПКП «ТИТАН» успешно завершена.

В ходе практики были достигнуты поставленные цели:

* получен практический опыт по выполнению работ по ПМ.11 «Разработка, администрирование и защита баз данных» и развиты общие и профессиональные компетенции;
* получен практический опыт по выполнению работ по ПМ.01 «Разработка модулей программного обеспечения» и развиты общие и профессиональные компетенции.

Для достижения целей практики выполнены следующие задачи:

* разработаны компоненты проектной и технической документации с использованием графических языков спецификаций;
* разработаны модули программного обеспечения для мобильных платформ;
* разработан код программного продукта на основе готовых спецификаций на уровне модуля;
* выполнение тестирования программных модулей;
* осуществлен рефакторинг и оптимизация программного кода;
* спроектирована БД;
* разработаны объекты БД;
* реализована БД в СУБД My SQL WorkBench;
* решены вопросы администрирования БД;
* реализованы методы и технологии защиты информации в БД.

По результатам производственной практики приобретены ценные практические навыки в разработке баз данных, создании и оптимизации API, HTML-страницами и мобильных приложений. Полученные знания и опыт будут полезны для дальнейшего профессионального роста и успешного выполнения задач в сфере информационных технологи.

Список использованных источников

1. Голицына, О. Л. Базы данных : учебное пособие / О. Л. Голицына, Н.В. Максимов, И. И. Попов. – Москва : ФОРУМ : ИНФРА-М, 2020. – 400 с.
2. Голицына, О. Л. Основы проектирования баз данных : учебное пособие / О. Л. Голицына, Т. Л. Партыка, И. И. Попов. – Москва : ФОРУМ : ИНФРА-М, 2021. – 416 с.
3. Документация Flask 2.2. Текст : электронный // Django: [сайт]. – 2025. – URL: https://django.fun/docs/flask/2.2/#api-reference (дата обращения: 30.04.2025).
4. Kotlin : официальная документация языка программирования Kotlin : сайт. – Прага. – 2025. – URL: https://kotlinlang.org (дата обращения 15.05.2025). – Текст : электронный
5. Python : официальная документация языка программирования Python : сайт. – Прага. – 2025. – URL: https://docs.python.org/3.10 (дата обращения 24.04.2025). – Текст : электронный