**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

\_\_\_\_\_\_\_\_\_\_\_ С.С. Лихванчук

(Подпись) (И.О. Фамилия)

«31» мая 2025г.

**ТЕХНИЧЕСКИЙ ОТЧЁТ**

**по ПМ.11, ПМ.01**

|  |
| --- |
| НАИМЕНОВАНИЕ ОРГАНИЗАЦИИ |
| Информационные системы и программирование |
| 09.02.07. 25ТО01. 001 ПЗ |
| (Обозначение документа) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИСПП-21 | |  | 31.05.2025 | А.М. Вепрёв |
|  | (Группа) | | (Подпись) | (Дата) | (И.О. Фамилия) |
| Рук. практики от предприятия | | |  | 31.05.2025 | М.В. Вепрёв |
|  | |  | (Подпись) | (Дата) | (И.О. Фамилия) |

Архангельск 2025

Содержание

[Перечень сокращений и обозначений 3](#_Toc199288309)

[Введение 4](#_Toc199288310)

[1 Организация охраны труда на предприятии 6](#_Toc199288311)

[1.1 Общие положения о службе охраны труда 6](#_Toc199288312)

[1.2 Требования охраны труда для программиста 7](#_Toc199288313)

[2 Выполнение работ по ПМ.11 9](#_Toc199288314)

[2.1 Проектирование базы данных 9](#_Toc199288315)

[2.2 Разработка базы данных и объектов базы данных 11](#_Toc199288316)

[2.3 Администрирование и защита базы данных 15](#_Toc199288317)

[3 Выполнение работ по ПМ.01 17](#_Toc199288318)

[3.1 Проектирование программного обеспечения 17](#_Toc199288319)

[3.2 Разработка программных модулей 18](#_Toc199288320)

[3.3 Разработка мобильного приложения 19](#_Toc199288321)

[3.4 Отладка и тестирование программных модулей 21](#_Toc199288322)

[3.5 Оптимизация и рефакторинг программного кода 22](#_Toc199288323)

[Заключение 26](#_Toc199288324)

[Список использованных источников 27](#_Toc199288325)

Перечень сокращений и обозначений

В настоящем техническом отчете применяются следующие сокращения и обозначения:

ПК – персональный компьютер

ПО – программное обеспечение

…

UML – унифицированный язык моделирования

WPF – Windows Presentation Foundation

…

Введение

Прохождение производственной практики осуществляется в военном комиссариате Архангельской области. Военный комиссариат выполняет следующие функции: организует постановку граждан на воинский учет, призыв на военную службу, ведет учет военнообязанных, проводит подготовку призывников и предоставляет консультации по вопросам военной службы и социальной защиты.

Цели производственной практики:

* 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:

* осуществление сбора, обработки и анализа информации для проектирования БД;
* проектирование БД на основе анализа предметной области;
* разработка БД в соответствии с результатами анализа предметной области;
* реализация БД в конкретной СУБД;
* администрирование БД;
* защита информации в БД с использованием технологий защиты информации;
* формирование алгоритмов разработки программных модулей и их разработка в соответствии с техническим заданием;
* выполнение отладки программных модулей с использованием специализированных программных средств;
* выполнение тестирования программных модулей;
* осуществление рефакторинга и оптимизации программного кода;
* разработка программных модулей для мобильных платформ.

Для выполнения поставленных задач предоставляется ПК со следующим аппаратным и программным обеспечением:

* процессор – Intel Core i5-7400;
* материнская плата – Asus Prime B250-Plus;
* блок питания – Aerocool VX 700W;
* ОЗУ – Kingston HyperX FURY 16 ГБ;
* SSD – Sandisk SSD Plus 120 ГБ;
* HDD – 1 ТБ Seagate SV35;
* ОС – Microsoft Windows 10 Pro;
* IDE – Visual Studio 2022, Android Studio;
* СУБД – MySql.

# Организация охраны труда на предприятии

## Общие положения о службе охраны труда

Служба охраны труда (далее - служба) является структурным подразделением организации.

Специалисты службы должны знать и руководствоваться в своей работе следующим:

* конституцией Российской Федерации, законами и иными нормативными правовыми актами в сфере охраны труда;
* государственными нормативными требованиями охраны труда;
* международными договорами в области охраны труда, ратифицированными Российской Федерацией;
* национальными и межгосударственными стандартами в области безопасности и охраны труда;
* требованиями охраны труда, установленными правилами и инструкциями по охране труда;
* делопроизводством и методическими документами по вопросам охраны труда;
* методами выявления, оценки и управления профессиональными рисками;
* производственной и организационной структурой организации;
* основными технологическими процессами и режимами производства;
* видами применяемого оборудования и правилами его эксплуатации;
* методами изучения условий труда на рабочих местах;
* психофизиологическими требованиями к работникам;
* правилами и средствами контроля соответствия технического состояния оборудования требованиям безопасного ведения работ;
* порядком проведения расследования несчастных случаев;
* передовым отечественным и зарубежным опытом в области охраны труда;
* порядком и сроками составления отчетности о выполнении мероприятий по охране труда;
* локальными актами работодателя.

Все специалисты службы должны пройти специальное обучение по охране труда.

## Требования охраны труда для программиста

Программист должен соблюдать следующие требования охраны труда во время работы:

* персональный компьютер следует включать в работу в той последовательности, которая определена инструкцией по эксплуатации;
* для подключения персонального компьютера к электрической сети программист должен использовать шнур питания, поставляемый в комплекте с персональным компьютером; не следует использовать самодельные электрические шнуры для подключения к сети персонального компьютера и различных его устройств;
* программист должен знать, что рациональная рабочая поза способствует уменьшению утомляемости;
* при помощи поворотной площадки видеомонитор должен быть отрегулирован в соответствии с рабочей позой программиста;
* конструкция рабочего стула (кресла) должна обеспечивать поддержание рабочей позы программиста при работе с персональным компьютером, позволять изменять позу с целью снижения статического напряжения мышц шейно-плечевой области и спины для предупреждения развития утомления;
* тип рабочего стула (кресла) должен выбираться в зависимости от характера и продолжительности работы с персональным компьютером с учетом роста программиста;
* рабочий стул (кресло) должен быть подъемно-поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также расстоянию спинки от переднего края сиденья; при этом регулировка каждого параметра должна быть независимой, легко осуществляемой и иметь надежную фиксацию;
* поверхность сиденья, спинки и других элементов стула (кресла) должна быть полумягкой, с нескользящим, неэлектризуемым и воздухопроницаемым покрытием, обеспечивающим легкую очистку от загрязнений;
* плоскость рабочего стола должна быть регулируемой по высоте в пределах 680–800 мм с учетом индивидуальных особенностей программиста; при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм;
* экран видеомонитора должен находиться от глаз программиста на оптимальном расстоянии 600–700 мм, но не ближе 500 мм с учетом размеров алфавитно-цифровых знаков и символов;
* клавиатуру следует располагать на поверхности стола на расстоянии 100–300 мм от края, обращенного к пользователю, или на специальной, регулируемой по высоте рабочей поверхности, отделенной от основной столешницы;
* для снятия зрительного и позотонического напряжения программисту в процессе работы следует устраивать микропаузы продолжительностью 1–3 минуты.

# Выполнение работ по ПМ.11

## Проектирование базы данных

Для военного комиссариата требуется разработать телеграм-бота, информирующего призывников по определенным вопросам и уведомляющего их о значимых датах нашей страны.

В БД требуется хранить следующие данные о вопросах: заголовок вопроса, имя файла, данные файла, имя фотографии, данные фотографии, дата обновления; о пользователях: имя пользователя, дата создания, время последней активности; о категориях: название категории; об уведомлениях: (название уведомления), текст уведомления, дата отправки, статус, фотография уведомления (, дата обновления).

На рисунке 1 показана концептуальная модель предметной области в виде ERD.

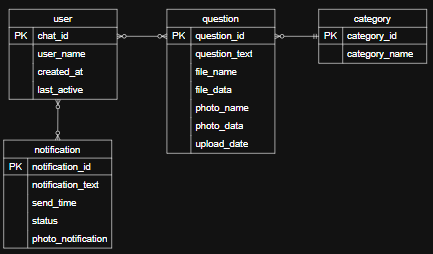


Рисунок 1 – Концептуальная модель

Для хранения данных выбрана реляционная СУБД MySQL, поэтому требуется преобразовать разработанную концептуальную модель в логическую с учетом правил преобразования ER-модели в реляционную модель данных:

* связь между сущностями question и category 1:M и необязательная для сущности category, поэтому формируется два отношения, соответствующих исходным сущностям, внешний ключ добавляется в отношение question;
* связь между сущностями question и user М:М и необязательная для обеих сущностей, поэтому формируется три отношения, два соответствуют исходным сущностям, третье, history, является связующим и содержит два внешних ключа и дату запроса, являющихся частью первичного ключа;
* связь между сущностями user и notification М:М и необязательная для обеих сущностей, поэтому формируется три отношения, два соответствуют исходным сущностям, третье, recipient, является связующим и содержит два внешних ключа, являющихся частью первичного ключа.

На рисунке 2 представлена логическая модель предметной области, преобразованная из концептуальной модели.

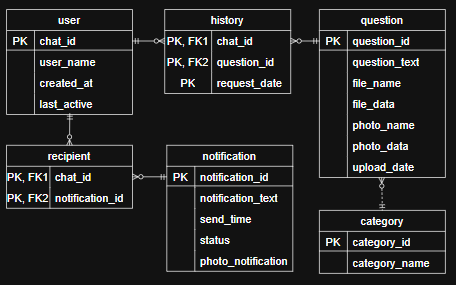


Рисунок 2 – Логическая модель

Согласно разработанной схеме БД, все значения в атрибутах являются атомарными, все значения функционально зависят от первичного ключа и отсутствуют транзитивные зависимости, что соответствует требованиям третьей нормальной формы.

На рисунке 3 представлена физическая модель предметной области, разработанная на основе логической модели для СУБД MySQL. Для проектирования БД использовано CASE-средство проектирования БД MySQL Workbench.

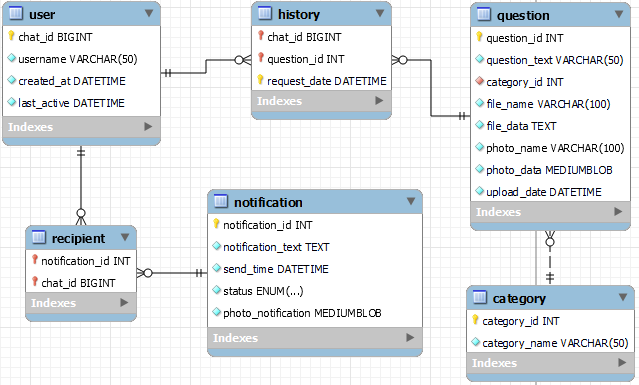


Рисунок 3 – Физическая модель

## Разработка базы данных и объектов базы данных

При создании таблиц необходимо указывать тип данных и ограничения целостности. В таблице 1 в виде словаря данных представлено описание созданных таблиц и ограничений целостности БД.

Таблица 1 – Словарь данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ключ** | **Поле** | **Тип данных** | **Обязательное** | **Примечание** |
| question | | | | |
| PK | question\_id | int | + | Автоинкрементный |
| FK | category\_id | int | + |  |
|  | question\_text | text | + |  |
|  | file\_name | varchar | + | Размер: 100, уникальное |
|  | file\_data | text | + |  |
|  | photo\_name | varchar | + | Размер: 100, уникальное |
|  | photo\_data | mediumblob | + |  |
|  | upload\_date | datetime | + | По умолчанию: CURRENT\_TIMESTAMP |
| category | | | | |
| PK | category\_id | int | + | Автоинкрементный |
|  | category\_name | varchar | + | Размер: 50 |
| user | | | | |
| PK | chat\_id | int | + | Автоинкрементный |
|  | username | varchar | + | Размер: 50 |
|  | created\_at | datetime | + | По умолчанию: CURRENT\_TIMESTAMP |
|  | last\_active | datetime | + |  |
| notification | | | | |
| PK | notification\_id | int | + | Автоинкрементный |
|  | notification\_text | text | + |  |
|  | status | enum | + | Допустимые значения типа: 'ожидает', 'отправлено' |
|  | send\_time | datetime | + |  |
|  | photo\_notification | mediumblob | + |  |
| history | | | | |
| PK,  FK1 | chat\_id | int | + |  |
| PK, FK2 | notification\_id | int | + |  |
| PK | request\_date | datetime | + | По умолчанию: CURRENT\_TIMESTAMP |
|  |  |  |  |  |
|  |  |  |  |  |

*Продолжение таблицы 1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ключ** | **Поле** | **Тип данных** | **Обязательное** | **Примечание** |
|  | Name | nvarchar(100) | + | Уникально |
| … | … | … | … | … |
| … | … | … | … | … |
| … | … | … | … | … |

Для заполнения таблиц БД данными из файлов формата \*.csv был использован мастер импорта данных MySQL Workbench.

Для этого следует предпринять следующие действия:

* в панели «Navigator» нажать ПКМ по БД;
* в открывшемся меню выбрать пункт «Table Data Import Wizard»;
* в открывшемся мастере импорта выбрать файл формата \*.csv для соответствующей таблицы и, нажав на кнопку «Next», выполнить дополнительные настройки;
* после завершения всех настроек нажать на кнопку «Next».

Также необходимо создать запросы к БД и подпрограммы на языке SQL.

В БД требуется создать следующие представления:

* представление, возвращающее юридические вопросы: код вопроса и заголовок вопроса;
* представление, возвращающее информационные вопросы: код вопроса и заголовок вопроса;
* представление, возвращающее медицинские вопросы: код вопроса и заголовок вопроса;
* представление, возвращающее историю вопросов пользователей: заголовок вопроса, имя пользователя, дата обращения. Код представлен в листинге 1.

Листинг 1 – SQL-запрос для создания представления user\_question\_history

-- Создание представления для истории вопросов пользователей

CREATE VIEW user\_question\_history AS

SELECT

u.username,

q.question\_text,

h.request\_date

FROM

history h

JOIN

user u ON h.chat\_id = u.chat\_id

JOIN

question q ON h.question\_id = q.question\_id;

В БД требуется создать следующие хранимые процедуры:

* хранимую процедуру, которая возвращает уведомления на текущий момент;
* хранимую процедуру, которая обновляет статус уведомления на 'отправлено' по заданному коду уведомления. Код представлен в листинге 2;
* хранимую процедуру, которая добавляет запись в таблицу история по коду чата и коду вопроса;
* хранимую процедуру, которая добавляет нового пользователя или обновляет его, если он уже существует, по имени пользователя, коду чата и последней активности.

Листинг 2 – SQL-запрос для создания процедуры update\_notification\_status

-- Создание процедуры, которая обновляет статус уведомления на 'отправлено'

DELIMITER //

CREATE PROCEDURE update\_notification\_status(IN p\_notification\_id INT)

BEGIN

UPDATE notification

SET status = 'отправлено'

WHERE notification\_id = p\_notification\_id;

END //

DELIMITER ;

В БД требуется создать следующие триггеры:

* триггер, который добавляет «подписку» пользователям на все уведомления в таблице получатель при добавлении нового уведомления;
* триггер, который «подписывает» пользователя на уведомления в таблице получатель. Код представлен в листинге 3;
* триггер для обновления даты загрузки вопроса;
* триггер на вставку, который корректирует дату и время отправки уведомления, чтобы она не была в прошедшем времени, добавляя год при необходимости, если месяц и день уже прошли в этом году, и устанавливает статус "ожидает";
* триггер на обновление, который устанавливает статус "ожидает", если статус изменился на "отправлено" и корректирует дату отправки, если дата отправки уже прошла.

Листинг 3 – SQL-запрос для создания триггера after\_user\_insert

-- Создание триггера, который "подписывает" нового пользователя на уведомления

DELIMITER //

CREATE TRIGGER after\_user\_insert

AFTER INSERT ON user

FOR EACH ROW

BEGIN

INSERT INTO recipient (chat\_id, notification\_id)

SELECT NEW.chat\_id, notification\_id

FROM notification;

END; //

DELIMITER ;

## Администрирование и защита базы данных

Для разграничения прав доступа к БД необходимо создать роли, соответствующие следующим категориям пользователей: призывник – права на чтение информации о вопросах; военный комиссар – права на чтение и запись данных в БД.

Создание роли и пользователя приведены в листинге 4.

Листинг 4 – SQL-запрос для создания роли и пользователя

-- Создание роли и присвоение ей прав

CREATE ROLE military\_recruitment\_officer;

GRANT SELECT, INSERT, UPDATE, DELETE ON voenkom.\* TO military\_recruitment\_officer;

-- Создание пользователя и назначение ему роли

CREATE USER 'commissar'@'%' IDENTIFIED BY '19450509';

GRANT military\_recruitment\_officer TO 'commissar'@'%';

SET DEFAULT ROLE military\_recruitment\_officer TO 'commissar'@'%';

В ходе практики был использован локальный сервер OpenServer для хранения БД и PhpMyAdmin для администрирования СУБД. Резервная копия БД была создана при помощи инструмента PhpMyAdmin – экспорт, в виде SQL скрипта. Для того, чтобы создать резервную копию БД в PhpMyAdminer, необходимо:

* выбрать нужную БД и в верхней части меню выбрать пункт «Экспорт»;
* после откроется окно для настройки экспорта;
* в этом окне можно выбрать формат файла для экспорта и метод экспорта: «Быстрый» – отображает минимум настроек, и «Обычный» – отображает все возможные настройки;
* после всех махинаций нажать на кнопку «Вперёд» для скачивания SQL скрипта с резервной копией БД.

Для восстановления БД из резервной копии необходимо запустить SQL скрипт, полученный при экспорте.

# Выполнение работ по ПМ.01

## Проектирование программного обеспечения

Требуется разработать систему, благодаря которой пользователь (призывник) сможет ознакомиться с информацией, касающейся службы в армии.

Назначение ПО – информирование пользователей (призывников) по различному роду вопросов по поводу службы в армии и вещами, связанными с ней.

ПО предполагает, что пользователь (призывник) выбирает категорию вопроса и интересующий его вопрос, после чего получает соответствующую информацию о вопросе, а также возможность либо скачать этот вопрос документом или фото-памятку к нему, либо скачать все вопросы документом или все фото-памятки архивом; получение уведомлений.

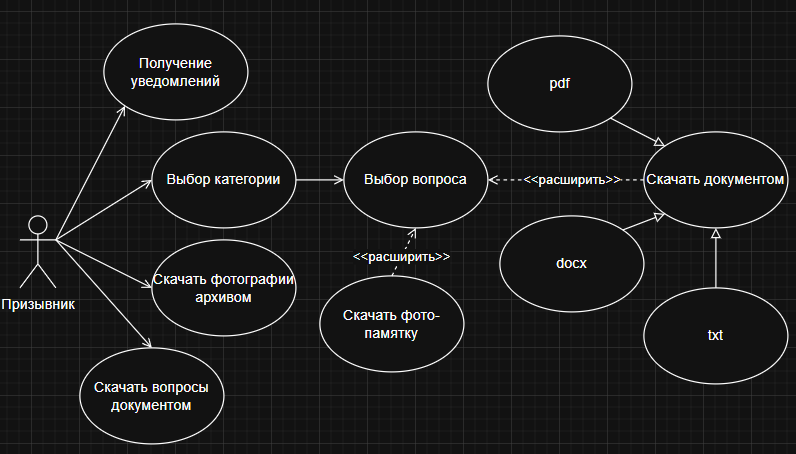


Рисунок 4 – Диаграмма прецедентов

## Разработка программных модулей

В течение производственной практики был разработан телеграм-бот при помощи библиотеки Telegram.Bot на языке C# в среде разработки Visual Studio 2022. Доступ к БД осуществлялся через API-приложение, которое в свою очередь взаимодействовало с БД благодаря ORM Entity Framework Core и библиотеке MySql.Data, при помощи библиотеки System.Net.Http.

Для отображения категорий вопросов был реализован метод GetCategoriesAsync. Код метода приведён в листинге 5. Отображение категорий вопросов в чате с телеграм-ботом представлено рисунком 5.

Листинг 5 – Метод получения категорий

public async Task<List<Category>> GetCategoriesAsync()

{

// Отправляем асинхронный GET-запрос к API

var response = await \_httpClient.GetAsync("/api/Category");

// Проверяем успешность ответа

if (!response.IsSuccessStatusCode)

// В случае ошибки возвращаем пустой список

return new List<Category>();

// Читаем содержимое ответа как строку

var json = await response.Content.ReadAsStringAsync();

// Десериализуем JSON в список категорий

var categories = JsonConvert.DeserializeObject<List<Category>>(json);

// Объявляем цикл для обработки каждой категории

foreach (var c in categories)

{

// Если название категории не пустое

if (!string.IsNullOrEmpty(c.CategoryName))

// Преобразуем первую букву в верхний регистр

c.CategoryName = char.ToUpper(c.CategoryName[0]) + c.CategoryName.Substring(1);

}

// Возвращаем список категорий

return categories;

}

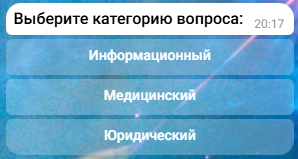


Рисунок 5 – Отображение списка категорий вопросов

## Разработка мобильного приложения

В течение производственной практики было разработано мобильное приложение для администрирования БД и отслеживания информации о пользователях на языке Kotlin в среде разработки Android Studio. Доступ к БД осуществлялся через API-приложение при помощи библиотеки Retrofit.

Для получения списка вопросов и соответствующей вопросу категории были разработаны два метода: getCategoryNameById() в CategoryViewModel и getQuestionList() в QuestionViewModel. Первый метод представлен в листинге 6, а второй в листинге 7. Экран с вопросами представлен рисунком 6.

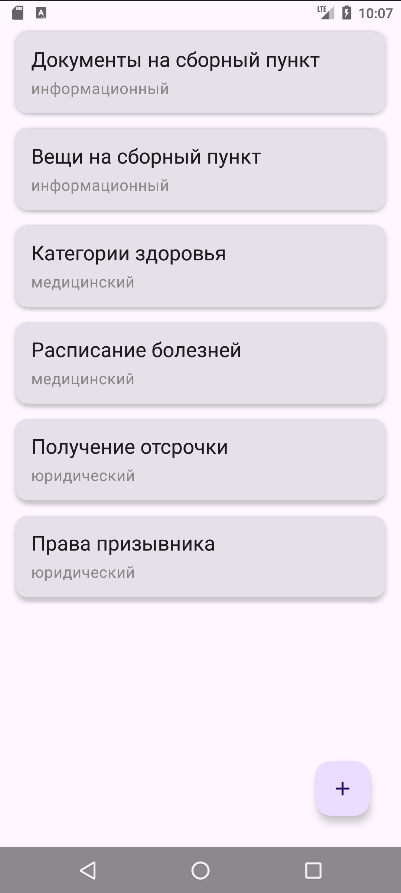


Рисунок 6 – TgBotAdmin. Вид экрана с вопросами

Листинг 6 – CategoryViewModel. Метод получения имени категории по идентификатору

fun getCategoryNameById(categoryId: Int): String? {

// Пробуем получить данные из состояния, проверяя, что оно является успешным

return (\_categoryListState.value as? UiState.Success<List<Category>>)

?.data // Получаем список категорий из состояния успеха

?.find { it.categoryId == categoryId } // Ищем категорию по заданному идентификатору

?.categoryName // Возвращаем имя категории, если она найдена

}

Листинг 7 – QuestionViewModel. Метод получения списка вопросов

fun getQuestionList() {

// Запускаем корутину в области видимости ViewModel

viewModelScope.launch {

// Устанавливаем состояние загрузки перед началом запроса

\_questionListState.value = UiState.Loading

try {

// Выполняем запрос на получение списка вопросов через сервис

val response = questionService.getQuestions()

// Проверяем, успешен ли ответ

if(response.isSuccessful) {

// Если ответ успешен, обрабатываем тело ответа

response.body()?.let { question ->

// Устанавливаем состояние успеха с полученными вопросами

\_questionListState.value = UiState.Success(question)

} ?: run {

// Если тело ответа пустое, устанавливаем состояние ошибки с кодом ответа

\_questionListState.value = UiState.Error("HTTP error: ${response.code()}")

}

}

} catch (e: Exception) {

// В случае исключения устанавливаем состояние ошибки с сообщением об ошибке

\_questionListState.value = UiState.Error("Неопределённая ошибка: ${e.message}")

}

}

}

## Отладка и тестирование программных модулей

Описание возможностей отладчика/процесса отладки

Тестирование приложения: метод тестирования, наборы тестов/тест-кейсы, для всего наиболее значимого функционала, unit-тесты определенных методов.

## Оптимизация и рефакторинг программного кода

В процессе рефакторинга программного кода была оптимизирована логика обработки уведомлений. Основная функциональность, связанная с проверкой и отправкой уведомлений, была вынесена в отдельный асинхронный метод NotificationCheckerAsync. Это улучшение способствует более четкой структуре кода и повышает его читаемость.

Для повышения эффективности обработки уведомлений была реализована параллельная отправка уведомлений с использованием Task.WhenAll, что позволяет одновременно обрабатывать несколько запросов.

Также было добавлено логирование ошибок при помощи библиотек Serilog и Serilog.Sinks.File, что обеспечивает более надежный мониторинг состояния приложения.

Код до изменений представлен в листинге N, а после – листингом N.

Листинг N – Отправка уведомлений до рефакторинга и оптимизации кода

private async Task NotificationCheckerAsync(CancellationToken cancellationToken)

{

while (!cancellationToken.IsCancellationRequested)

{

try

{

var notifications = await \_apiService.GetCurrentNotificationsAsync();

if (notifications == null || notifications.Count == 0)

{

await Task.Delay(TimeSpan.FromMinutes(1), cancellationToken);

continue;

}

foreach (var notification in notifications)

{

var userChatIds = await \_apiService.GetUsersForNotificationAsync(notification.NotificationId);

if (userChatIds == null) continue;

foreach (var chatId in userChatIds)

{

try

{

if (notification.PhotoNotification != null && notification.PhotoNotification.Length > 0)

{

using var ms = new MemoryStream(notification.PhotoNotification);

var inputPhoto = new InputFileStream(ms);

await \_botClient.SendPhoto(chatId, inputPhoto, notification.NotificationText);

}

else

{

await \_botClient.SendMessage(chatId, notification.NotificationText);

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка отправки уведомления пользователю: {chatId}: {ex.Message}");

}

}

var marked = await \_apiService.MarkNotificationAssentAsync(notification.NotificationId);

if (!marked)

{

Console.WriteLine($"Не удалось отметить уведомление {notification.NotificationId} как 'отправленное'.");

}

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка в методе NotificationChecker: {ex.Message}");

}

await Task.Delay(TimeSpan.FromMinutes(1), cancellationToken);

}

}

Листинг N – Отправка уведомлений после рефакторинга и оптимизации кода

private async Task NotificationCheckerAsync(CancellationToken cancellationToken)

{

// Цикл, который будет выполняться, пока не будет отменен через cancellationToken

while (!cancellationToken.IsCancellationRequested)

{

try

{

// Получаем текущие уведомления из API

var notifications = await \_apiService.GetCurrentNotificationsAsync();

// Если уведомлений нет, ждем 1 минуту и продолжаем цикл

if (notifications == null || notifications.Count == 0)

{

await Task.Delay(TimeSpan.FromMinutes(1), cancellationToken);

continue;

}

var tasks = new List<Task>();

// Для каждого уведомления получаем список пользователей, которым оно должно быть отправлено

foreach (var notification in notifications)

{

var userChatIds = await \_apiService.GetUsersForNotificationAsync(notification.NotificationId);

// Если нет пользователей для уведомления, продолжаем к следующему уведомлению

if (userChatIds == null) continue;

// Для каждого пользователя создаем задачу на отправку уведомления

foreach (var chatId in userChatIds)

{

tasks.Add(SendNotificationAsync(chatId, notification));

}

}

// Ждем завершения всех задач по отправке уведомлений

await Task.WhenAll(tasks);

// После отправки уведомлений помечаем их как отправленные в API

foreach (var notification in notifications)

{

var marked = await \_apiService.MarkNotificationAssentAsync(notification.NotificationId);

// Если не удалось отметить уведомление как отправленное, логируем предупреждение

if (!marked)

{

Log.Warning($"Не удалось отметить уведомление {notification.NotificationId} как 'отправленное'.");

}

}

}

catch (Exception ex)

{

// Логируем ошибки, возникающие во время проверки уведомлений

Log.Error(ex, "Ошибка в методе NotificationChecker");

}

// Ждем 1 минуту перед следующей проверкой уведомлений

await Task.Delay(TimeSpan.FromMinutes(1), cancellationToken);

}

}

private async Task SendNotificationAsync(long chatId, Notification notification)

{

try

{

// Проверяем, есть ли у уведомления фотография

if (notification.PhotoNotification != null && notification.PhotoNotification.Length > 0)

{

// Если фотография есть, создаем поток из байтового массива и отправляем фото пользователю

using var ms = new MemoryStream(notification.PhotoNotification);

var inputPhoto = new InputFileStream(ms);

await \_botClient.SendPhoto(chatId, inputPhoto, notification.NotificationText);

}

else

{

// Если фотографии нет, просто отправляем текст уведомления

await \_botClient.SendMessage(chatId, notification.NotificationText);

}

}

catch (Exception ex)

{

// Логируем ошибки, возникающие при отправке уведомления пользователю

Log.Error(ex, $"Ошибка отправки уведомления пользователю: {chatId}");

}

}

Заключение

По прохождении производственной практики в ВКАО был получен практический опыт выполнения работ по ПМ.01 «Разработка модулей программного обеспечения для компьютерных систем» и ПМ.11 «Разработка, администрирование и защита баз данных». В результате были закреплены теоретические знания, приобретены необходимые практические умения и навыки работы в соответствии с установленными компетенциями.

Поставленные цели достигнуты и в процессе их достижения были решены следующие задачи:

* осуществлен сбор, обработка и анализ информации для проектирования БД;
* спроектирована БД на основе анализа предметной области;
* разработана БД в соответствии с результатами анализа предметной области;
* была выполнена реализация БД в конкретной СУБД;
* было осуществлено администрирование БД;
* было осуществлена защита информации в БД с использованием технологий защиты информации;
* сформированы алгоритмы разработки программных модулей и осуществлена их разработка в соответствии с техническим заданием;
* выполнена отладка программных модулей с использованием специализированных программных средств;
* выполнено тестирование программных модулей;
* осуществлён рефакторинг и оптимизация программного кода;

Список использованных источников

1. Telegram.Bot. Документация / Telegram.Bot. – Текст : электронный // GitHub : [сайт]. – 2023. – URL: [https://github.com/TelegramBots/Telegram.Bot](https://github.com/TelegramBots/Telegram.Bot%20) (дата обращения: 18.05.2025).
2. Serilog. Logging for .NET Applications / Serilog. – Текст : электронный // serilog.net : [сайт]. – 2023. – URL: https://serilog.net/ (дата обращения: 18.05.2025).
3. Serilog.Sinks.File. File Sink for Serilog / Serilog. – Текст : электронный // GitHub : [сайт]. – 2023. – URL: https://github.com/serilog/serilog-sinks-file (дата обращения: 18.05.2025).
4. Microsoft. System.Net.Http / Microsoft Learn. – Текст : электронный // docs.microsoft.com : [сайт]. – 2023. – URL: <https://docs.microsoft.com/en-us/dotnet/api/system.net.http> (дата обращения: 18.05.2025).
5. Microsoft. Entity Framework Core / Microsoft Learn. – Текст : электронный // docs.microsoft.com : [сайт]. – 2023. – URL: [https://docs.microsoft.com/en-us/ef/core/](https://docs.microsoft.com/en-us/ef/core/%20) (дата обращения: 18.05.2025).
6. Retrofit. Type-safe HTTP client for Android and Java / Square. – Текст : электронный // square.github.io : [сайт]. – 2023. – URL: [https://square.github.io/retrofit/](https://square.github.io/retrofit/%20) (дата обращения: 21.05.2025).