Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский биотехнологический университет»

Кафедра *\_\_\_ПЦК «Информационные системы\_\_\_\_\_\_\_*

**ОТЧЕТ**

**по \_\_\_\_\_\_\_\_\_производственной \_\_\_\_\_\_\_\_\_\_\_\_ практике**

*(указать вид или тип практики)*

Место прохождения практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Срок прохождения практики с «24»\_\_\_03\_\_\_\_\_2025г. по «13»\_\_\_04\_\_\_2025г.

Руководитель практики от выпускающей кафедры Воронкова Я.А., преподаватель СПО

 *(ФИО, должность)*

Руководитель практики от организации\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 *(ФИО, должность)*

Выполнил:

студент \_\_\_21-ИСо-1\_\_\_группы \_\_4\_\_ курса

института (колледжа) \_\_МТК\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_Лавричнчук К. Д.\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_

 *(ФИО, подпись)*

«\_13\_\_» \_\_\_\_04\_\_\_\_\_\_\_\_ 20\_25\_г.

Москва, 2025 г.

1. **Диаграммы UML**

Диаграмма прецедентов (Use Case Diagram)

Показывает, какие действия могут выполнять разные роли (например, администратор, оператор):

Просмотр событий

Получение уведомлений

Просмотр логов

Настройка порогов

Вход в систему

Диаграмма классов (Class Diagram)

Отразим основные сущности:

User

CriticalEvent

Notification

EventLog

Threshold Связи: ассоциации, зависимости, композиция (например, Notification связан с User и CriticalEvent).

Диаграмма последовательностей (Sequence Diagram) или сотрудничества (Collaboration Diagram)

Описывает взаимодействие объектов при, например:

Создании события → триггер → формирование уведомлений

Входе пользователя → получение списка событий

2. **Проектирование пользовательского интерфейса**

Простой макет интерфейса с формами:

Вход в систему

Таблица критических событий

Просмотр логов

Управление пользователями (для администратора)

Настройки порогов

Можно сделать как набросок (вручную или в FigJam/Draw.io), либо я могу помочь с генерацией иллюстрации.

3. **Описание архитектуры**

Опиши клиент-серверную модель:

**Клиент:** веб-приложение или десктоп (напр. WinForms/Web)

**Сервер:** SQL Server + логика оповещения через триггер

**Интерфейс обмена:** SQL-запросы, возможно, API (если есть фронт)

4. **Программная реализация и тестирование**

Опиши:

Язык/технологии (SQL Server, возможно C#/Python/PHP, если есть фронт)

Реализация логики: создание триггера, генерация событий, автоматизация

Тест-кейсы:

Проверка генерации уведомлений при Severity = High

Проверка прав доступа пользователей

Проверка логирования

5. **Анализ рисков**

**Технические:** перебои в БД, утечки данных

**Организационные:** недостаточный контроль над пользователями

**Экономические:** стоимость внедрения и поддержки

**Безопасность:** несанкционированный доступ

Диаграмма прецедентов (Use Case Diagram)

Cущности:

**Администратор**

**Оператор**

Прецеденты (возможные действия):

| **№** | **Действие** | **Кто выполняет** |
| --- | --- | --- |
| 1 | Войти в систему | Администратор, Оператор |
| 2 | Просматривать критические события | Администратор, Оператор |
| 3 | Получать уведомления о событиях | Администратор |
| 4 | Просматривать логи событий | Администратор |
| 5 | Настраивать пороговые значения | Администратор |
| 6 | Управлять пользователями | Администратор |

Основные классы:

***User***

**Атрибуты:**

userId: int

username: string

role: string

email: string

**Методы:**

login()

logout()

***CriticalEvent***

**Атрибуты:**

eventId: int

eventType: string

description: string

severity: string

timestamp: datetime

**Методы:**

createEvent()

isCritical()

***Notification***

**Атрибуты:**

notificationId: int

userId: int

eventId: int

status: string

**Методы:**

send()

markAsRead()

***Log***

**Атрибуты:**

logId: int

eventId: int

action: string

timestamp: datetime

**Методы:**

writeLog()

***Threshold***

**Атрибуты:**

thresholdId: int

eventType: string

severityLevel: string

**Методы:**

setThreshold()

getThreshold()

Связи:

User — 1 ко многим → Notification

CriticalEvent — 1 ко многим → Notification

CriticalEvent — 1 ко многим → Log

Threshold связан с CriticalEvent по типу события

Сценарий: Обработка критического события и отправка уведомления

Участники:

**User** — администратор, получатель уведомлений

**Database** — база данных, где возникает событие

**Trigger** — триггер, обрабатывающий критические события

**NotificationService** — логика отправки уведомлений

**LogService** — модуль логирования

Шаги взаимодействия:

**Database** фиксирует новое событие (insert в таблицу CriticalEvent)

**Trigger** срабатывает при вставке записи с уровнем High

**Trigger** вызывает **NotificationService** для формирования уведомлений

**NotificationService** определяет получателей (из Users)

Уведомления записываются в таблицу Notification

**LogService** записывает факт создания уведомления в Logs

**User** получает уведомление при следующем входе или через интерфейс



Проектирование пользовательского интерфейса

**Цели интерфейса:**

**Удобство и интуитивность.** Пользовательский интерфейс (UI) должен быть разработан таким образом, чтобы как администраторы, так и операторы могли легко ориентироваться в системе.

**Быстрый доступ к критическим уведомлениям.** Отображение событий высокой важности (например, «High» критические события) и их подробностей должно быть доступно в режиме реального времени.

**Отчетность и аудит.** Пользователю необходим доступ к логам событий, подробным отчётам об ошибках или отклонениях, а также возможность фильтрации данных по типу события, уровню серьёзности, дате и т.д.

**Основные компоненты UI:**

**Главная панель/дашборд.** На дашборде должна отображаться сводка системы: общее количество событий, количество событий высокой критичности, состояние отправленных уведомлений и краткая статистика (например, графики динамики по времени).

**Страница событий.** Здесь пользователи видят список критических событий с возможностью сортировки и фильтрации по различным параметрам (тип события, уровень серьёзности, временной интервал).

**Карточка события.** Детальное представление конкретного события, включающее описание, связанные логи и информацию об отправленных уведомлениях.

**Модуль управления пользователями.** Интерфейс для управления данными пользователей (администраторы, операторы), просмотра контактов, а также настройки параметров уведомлений.

**Модуль настроек системы.** Возможность настройки пороговых значений для различных типов событий, а также конфигурация каналов уведомлений (например, email, SMS и т.д.).

**Рекомендации по дизайну:**

**Простота и минимализм.** Избегайте избыточных деталей, чтобы не отвлекать пользователя от основной функциональности.

**Адаптивный дизайн.** Интерфейс должен корректно отображаться как на десктопных, так и на мобильных устройствах.

**Визуализация информации.** Используйте диаграммы и графики для иллюстрации динамики событий, что поможет в быстрой оценке ситуации.

**Обратная связь.** Предусмотрите информационные окна (pop-up уведомления, toast-сообщения) для подтверждения действий пользователя и информирования о статусе операций.

Описание архитектуры разрабатываемой системы

**Клиент-серверная модель:**

**Клиентская часть.** Это пользовательский интерфейс (веб-приложение, мобильное или десктопное приложение), который отправляет запросы на сервер и отображает полученные данные. Основной упор делается на удобство работы и быстродействие.

**Серверная часть.** Центральный сервер отвечает за обработку запросов, выполнение бизнес-логики и взаимодействие с базой данных. В данной работе реализована серверная логика (например, триггер для отправки уведомлений после вставки критического события).

**Компоненты архитектуры:**

**База данных.** Система использует базу данных «DataShieldDB», где находятся таблицы пользователей, критических событий, логов и уведомлений. В отчёте приведены SQL-скрипты создания и наполнения базы, а также механизм триггеров для автоматической генерации уведомлений.

**Сервер приложений.** Отвечает за бизнес-логику (например, процесс обработки вставляемых событий, рассылку уведомлений) и взаимодействует с БД через SQL-запросы. Здесь реализуются проверки уровня серьёзности событий и логика рассылки.

**Интерфейс API.** Между клиентом и сервером может быть RESTful API или GraphQL, что позволяет клиенту отправлять запросы (на получение данных, фильтрацию, управление пользователями) и получать ответы в структурированном виде (JSON/XML).

**Слой безопасности.** Реализуются механизмы аутентификации (например, через логин/пароль, JWT токены) и авторизации, а также шифрование данных (хранение хэшированных паролей, защита каналов связи).

**Преимущества выбранной архитектуры:**

**Масштабируемость.** Разделение на клиентскую и серверную часть позволяет масштабировать систему при увеличении нагрузки.

**Модульность.** Легкость внедрения новых функциональных модулей, таких как дополнительные каналы уведомлений или интеграция с внешними системами.

**Универсальность.** Возможность поддержки различных типов клиентов без изменения основной логики обработки.

Процесс программной реализации

**Этапы разработки:**

**Анализ и сбор требований.**

Определение функциональных и нефункциональных требований: какие события необходимо отслеживать, какие уровни критичности, какие каналы уведомлений.

Согласование требований с конечными пользователями и заинтересованными сторонами (например, администраторами и операторами).

**Проектирование системы.**

**Проектирование базы данных.** Создание схемы таблиц (Users, CriticalEvents, EventLogs, Thresholds, Notifications) с описанием связей и триггеров, как это представлено в отчёте.

**Проектирование архитектуры.** Определение клиент-серверной модели, выбор технологий (например, ASP.NET, Java Spring, Node.js для серверной части, современные фреймворки для разработки клиентского приложения).

**Проектирование пользовательского интерфейса.** Разработка прототипов и макетов UI, утверждение прототипов с заказчиком.

**Разработка и интеграция.**

**Реализация серверной части.** Написание кода бизнес-логики, настройка сервера и взаимодействия с базой данных, реализация API.

**Создание базы данных.** Выполнение SQL-скриптов для создания и наполнения таблиц, настройка триггеров для автоматической отправки уведомлений.

**Разработка клиентской части.** Программирование интерфейса согласно утвержденному дизайну, интеграция с сервером через API.

**Модульное и интеграционное тестирование.** Проверка работы отдельных компонентов (например, тестирование работы триггеров, корректности уведомлений), а также общая интеграция системы.

**Тестирование и отладка.**

**Функциональное тестирование.** Проверка соответствия системы заявленным требованиям.

**Нагрузочное тестирование.** Оценка производительности базы данных и сервера, особенно при большом количестве событий.

**Безопасностное тестирование.** Анализ уязвимостей, проверка механизмов аутентификации и защиты данных.

**Внедрение и поддержка.**

**Пилотное внедрение.** Развертывание системы на тестовом сервере и получение обратной связи от пользователей.

**Корректировка и оптимизация.** На основе отзывов – внесение изменений и оптимизация бизнес-процессов.

**Запуск в промышленную эксплуатацию.** Финальное развертывание системы, настройка мониторинга и системы поддержки.

Анализ рисков реализации и внедрения ИС

**Основные категории рисков:**

**Технические риски:**

**Ошибки в бизнес-логике и синтаксисе SQL-кода.** Возможны ошибки при реализации триггеров и сложных SQL-запросов, что может привести к неверной обработке событий.

**Проблемы производительности.** При большом объёме данных и высокой частоте событий система может столкнуться с замедлением или перегрузкой, особенно если уведомления генерируются массово.

**Сбой связи между компонентами.** Нарушения во взаимодействии между клиентской и серверной частями могут привести к задержкам или недоступности данных.

**Риски безопасности:**

**Угрозы несанкционированного доступа.** Недостаточная защита данных может привести к утечке информации, особенно если база данных и коммуникационные каналы не защищены.

**Недостаточная защита уведомлений.** Отправка уведомлений без должной аутентификации и шифрования может создать угрозы для целостности системы.

**Операционные риски:**

**Ошибки пользователей.** Неправильное использование интерфейса или некорректное обновление данных может негативно сказаться на работе системы.

**Отсутствие резервного копирования.** Потеря данных при сбое оборудования или ошибках администрирования может привести к значительным убыткам.

**Неоптимизированные процессы обновления.** Внедрение новых версий системы без должного тестирования и планирования может вызвать сбои в работе.

**Риски внедрения:**

**Сопротивление изменениям.** Пользователи могут не принять новую систему, если она слишком сложна или отличается от привычных методов работы.

**Проблемы интеграции с существующими системами.** Внедрение нового решения может вызывать несовместимость с другими информационными системами организации.

**Срыв сроков проекта.** Непредвиденные технические сложности или задержки в разработке могут привести к нарушению графика реализации.

**Стратегии снижения рисков:**

**Ранняя диагностика и тестирование.** Применение модульного, интеграционного, нагрузочного и тестирования безопасности позволит выявлять и устранять ошибки на ранних этапах разработки.

**Планирование резервного копирования и аварийного восстановления.** Разработка схем резервного копирования базы данных и процедур восстановления после сбоев.

**Обучение пользователей.** Проведение обучающих сессий и создание подробных инструкций для пользователей, чтобы снизить вероятность ошибок при работе с системой.

**Пилотное тестирование и постепенное внедрение.** Запуск системы на тестовой базе с последующей обратной связью от конечных пользователей позволит скорректировать функциональность перед массовым развертыванием.

Заключение

Выполняя задание, основное внимание следует уделить созданию удобного и интуитивно понятного пользовательского интерфейса, который обеспечит оперативный доступ к информации о критических событиях. Архитектура системы построена по принципу разделения клиентской и серверной частей, что обеспечивает масштабируемость и модульность. Процесс программной реализации охватывает все стадии – от анализа требований до внедрения и поддержки, а детальный анализ рисков позволяет заблаговременно подготовиться к потенциальным проблемам. Такой комплексный подход позволит создать надёжную и безопасную информационную систему для оповещения администратора о критических событиях в базе данных.