МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка программного обеспечения мониторинга показателей эффективности работы персонала компании»

Обучающийся Д.Е. Алтунин
(Инициалы Фамилия) (личная подпись)

Руководитель к.т.н., доцент, О.В. Аникина
ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

С. 76, рис. 20, лит. источников 20

Тема «Разработка программного обеспечения мониторинга показателей эффективности работы персонала компании».

Работа состоит из введения, трех глав, заключения, списка использованной литературы и приложений.

Во введении раскрывается актуальность выбранной темы, описываются цель и задачи работы.

В первой главе проводится сравнительный анализ имеющихся на рынке программных средств, для осуществления мониторинга показателей эффективности персонала компании. Проводится анализ бизнес процессов Филиала ФКУ «Налог-Сервис» ФНС России в Сахалинской области, а так же сформированы требования к разрабатываемому программному обеспечению.

Во второй главе ведется проектирование программного обеспечения, включающее в себя проектирование архитектуры «клиент-сервер», проектирование и моделирование базы данных, а так же проектирование пользовательского интерфейса, с описанием алгоритмов тестирования программного обеспечения.

В третьей главе выполняется разработка программного обеспечения. В главе обосновывается выбор языка программирования, веб-фреймворка, СУБД и других дополнительных компонентов, а так же описывается реализация БД и серверной части веб-приложения, разработка клиентской и административной части веб-приложения.

В заключении обобщается проделанная работа по достижению цели и решению поставленных в введении задач, а так же описывается функционал разработанного программного обеспечения.

Содержание

Введение

В условиях динамичного развития информационных технологий и предоставления услуг технической поддержки, особое значение приобретает эффективное управление человеческими ресурсами. Персонал является ключевым фактором успеха любой компании, поэтому мониторинг и анализ показателей эффективности работы сотрудников становится неотъемлемой частью системы управления организацией. Своевременное получение объективной информации о результатах деятельности персонала позволяет принимать обоснованные управленческие решения, направленные на повышение качества услуг, оптимизацию бизнес-процессов и достижение стратегических целей компании [18],[6].

ФКУ “Налог-Сервис” ФНС России уделяет большое внимание вопросам управления персоналом и повышения эффективности работы сотрудников. Однако, несмотря на важность данной задачи, компания сталкивается с проблемой отсутствия эффективных инструментов для сбора, обработки и анализа данных о работе персонала. Традиционные методы оценки, такие как аттестация и ежегодная оценка результатов деятельности, не всегда обеспечивают достаточную оперативность и полноту информации. В связи с этим возникает потребность в разработке специализированного программного обеспечения, которое позволит автоматизировать процесс мониторинга показателей эффективности персонала и предоставит руководству ФКУ “Налог-Сервис” ФНС России области удобные инструменты для анализа и принятия решений.

Целью данной выпускной квалификационной работы является разработка веб-приложения для мониторинга показателей эффективности работы

персонала Филиала ФКУ “Налог-Сервис” ФНС России в Сахалинской области. Для достижения поставленной цели необходимо решить следующие задачи:

- провести обзор и анализ существующих решений для мониторинга показателей эффективности персонала;

- проанализировать бизнес-процессы ФКУ “Налог-Сервис” ФНС России в Сахалинской области, связанные с оценкой эффективности персонала, и сформировать требования к разрабатываемому ПО;

- спроектировать архитектуру веб-приложения, базу данных и пользовательский интерфейс с учетом специфики деятельности компании;

- разработать серверную и клиентскую части приложения, реализовать алгоритмы расчета показателей эффективности, актуальных для отрасли;

- провести тестирование и отладку разработанного программного обеспечения.

Объектом исследования является процесс мониторинга показателей эффективности работы персонала Филиала ФКУ “Налог-Сервис” ФНС России в Сахалинской области.

Предметом исследования выступают методы и инструменты разработки веб-приложений для автоматизации процесса мониторинга показателей эффективности персонала.

При выполнении работы использовались следующие методы исследования:

– анализ литературных источников и существующих решений в области мониторинга эффективности персонала;

– объектно-ориентированный анализ и проектирование (ООАП);

– методы концептуального и логического моделирования баз данных;

– методы разработки веб-приложений с использованием языка Python и фреймворка Flask;

– методы тестирования и отладки программного обеспечения.

Практическая значимость работы заключается в том, что разработанное веб-приложение может быть внедрено в ФКУ “Налог-Сервис” ФНС России для автоматизации процесса мониторинга показателей эффективности работы персонала. Это позволит:

– повысить оперативность и достоверность информации о результатах деятельности сотрудников, занятых в различных сферах деятельности (ИТ, кадры, бухгалтерия и др.);

– сократить трудозатраты на сбор и обработку данных;

– обеспечить руководство компании удобными инструментами для анализа и принятия управленческих решений, направленных на повышение качества работы и предоставляемых услуг технической поддержки;

– повысить мотивацию и вовлеченность персонала за счет прозрачности и объективности оценки результатов труда.

Разработанное ПО может быть адаптировано под специфику деятельности других компаний и масштабировано в случае увеличения количества сотрудников и объема обрабатываемых данных [6],[16].

1 Аналитическая часть

1.1 Обзор и анализ существующих решений для мониторинга показателей эффективности персонала

В настоящее время существует ряд программных продуктов и сервисов, предназначенных для мониторинга и оценки эффективности работы персонала в различных отраслях [1],[2],[18]. Рассмотрим наиболее известные и широко используемые решения.

1.1.1 SAP SuccessFactors Performance & Goals

SAP SuccessFactors Performance & Goals - это облачное решение для управления эффективностью персонала, которое позволяет устанавливать цели, отслеживать прогресс и проводить оценку результатов работы сотрудников [20].

Система предоставляет инструменты для постановки SMART-целей, выстраивания иерархии целей в соответствии со стратегией компании, а также для проведения промежуточных и годовых оценок эффективности. На рисунке 1 показан интерфейс системы.

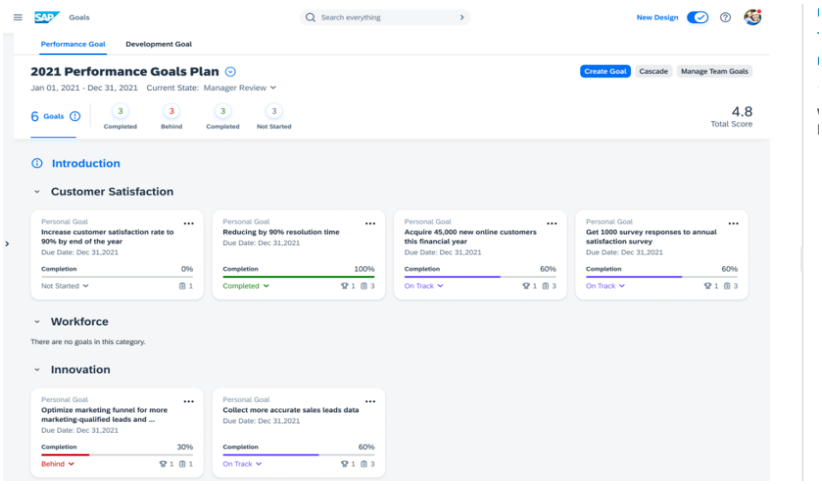

Рисунок 1 – Интерфейс SAP SuccessFactors

Ключевые преимущества SAP SuccessFactors Performance & Goals следующие:

- интеграция с другими модулями SAP SuccessFactors (обучение, развитие талантов, компенсации и льготы);
- пПоддержка непрерывного процесса управления эффективностью (цели, обратная связь, коучинг);
- мобильное приложение для iOS и Android;
- библиотека готовых целей и компетенций для различных отраслей и должностей.

Недостатки:

- высокая стоимость лицензий и внедрения;
- ограниченные возможности кастомизации и расширения функционала.

Стоимость: от $8 в месяц за пользователя (при покупке пакета из нескольких модулей SAP SuccessFactors)

1.1.2 Oracle Taleo Performance Management

Oracle Taleo Performance Management - это решение для управления эффективностью персонала, которое являются частью комплексной HCM-системы Oracle Taleo Cloud Service [19]. Продукт позволяет создавать цели, проводить оценку компетенций и результатов работы, предоставлять обратную связь и развивать таланты сотрудников.

Ключевые преимущества Oracle Taleo Performance Management:

- интеграция с другими модулями Oracle Taleo Cloud Service (подбор, обучение, управление талантами);
- поддержка различных методов оценки (по целям, компетенциям, OKR, 360 градусов)

– инструменты для калибровки оценок и анализа результатов;

– возможность настройки процессов оценки под специфику компании;

Недостатки:

– сложность внедрения и настройки системы;

– высокая стоимость лицензий и консалтинговых услуг.

Стоимость: от $15 в месяц за пользователя (при покупке пакета модулей Oracle Taleo Cloud Service). Интерфейс данной системы представлен на рисунке 2.

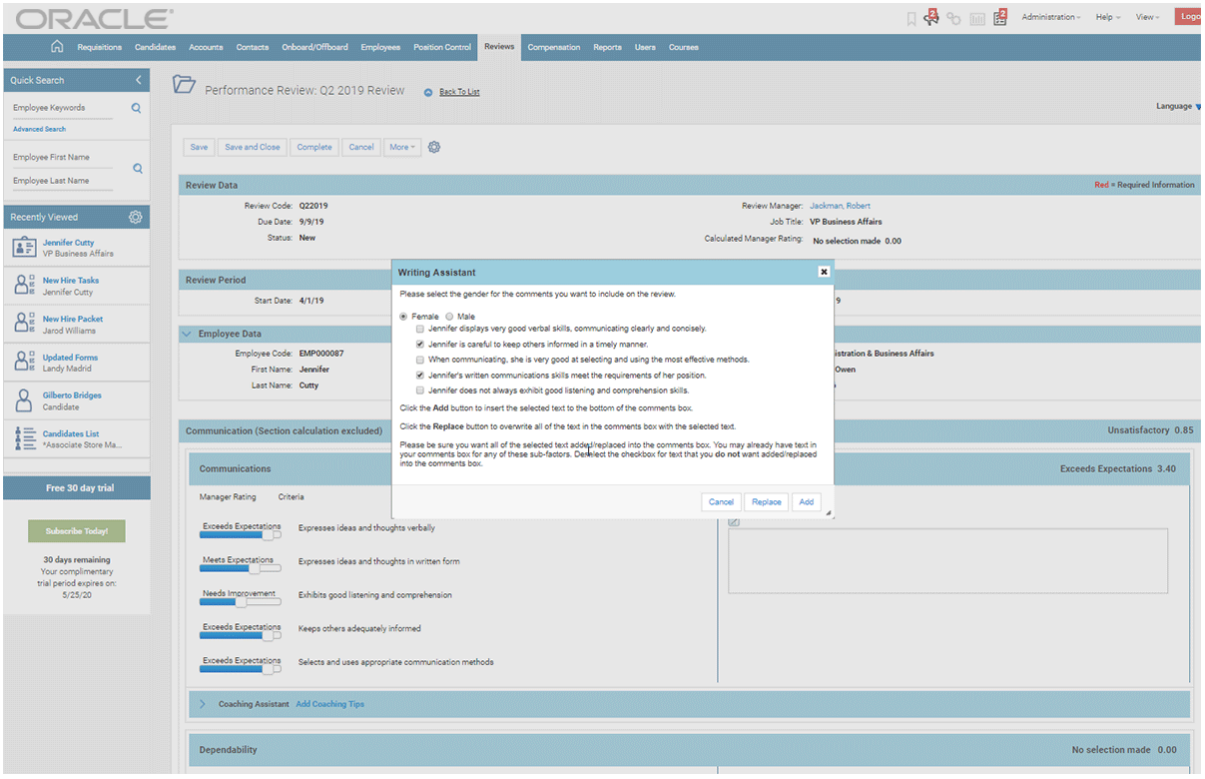

Рисунок 2 – Интерфейс Oracle Taleo

1.1.3 WebTutor

WebTutor - российская HRM-система, которая включает в себя модуль для управления эффективностью персонала [7]. Решение позволяет формировать цели сотрудников, проводить оценку по компетенциям, рассчитывать KPI (ключевые показатели эффективности) и формировать индивидуальные планы развития. Интерфейс данной системы показан на рисунке 3.

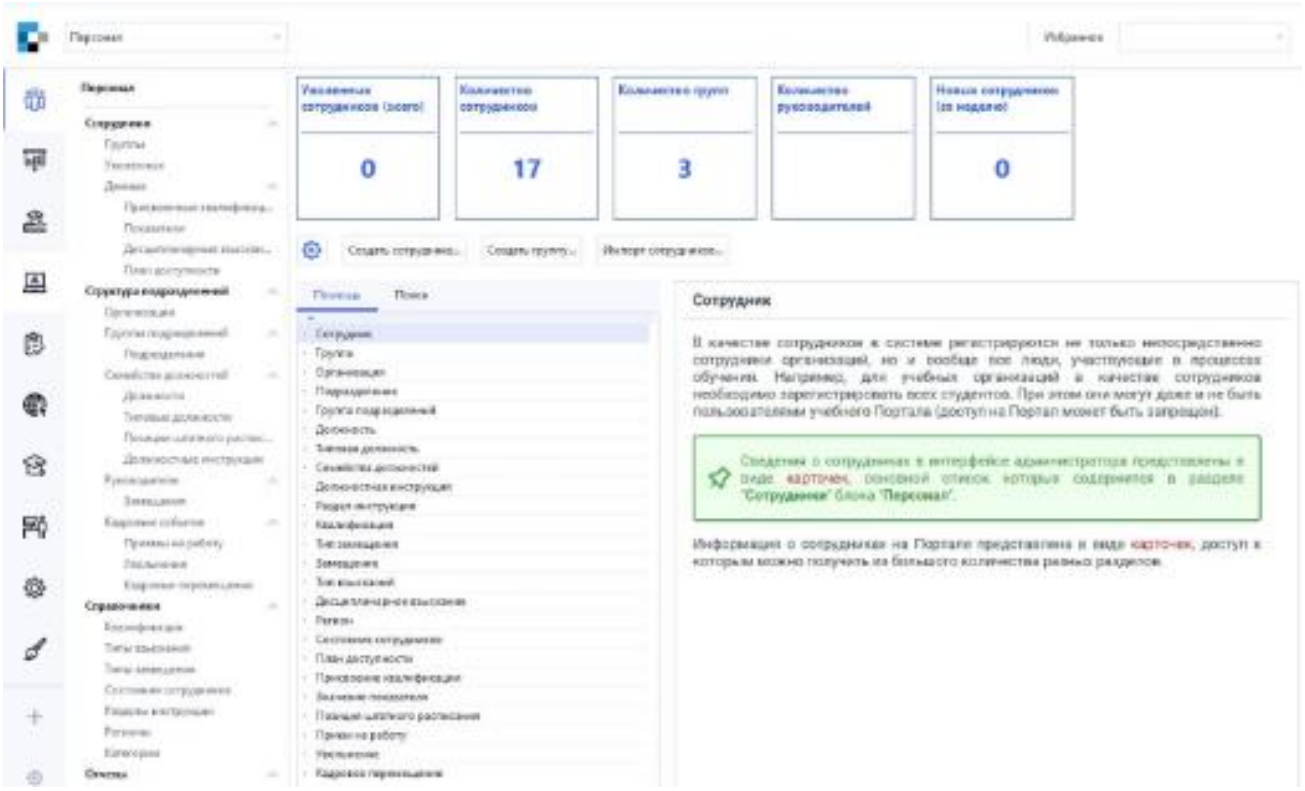

Рисунок 3 – Интерфейс WebTutore

Ключевые преимущества WebTutor:

- поддержка различных методик оценки (KPI, компетенции, управление по целям);
- интеграция с другими модулями WebTutor (подбор, адаптация, обучение, кадровый резерв);

– возможность гибкой настройки под специфику компании;

– относительно невысокая стоимость по сравнению с зарубежными аналогами.

Недостатки:

– ограниченный функционал в базовой версии продукта;

– отсутствие встроенных инструментов для сравнительного анализа эффективности сотрудников.

Стоимость: от 200 руб. в месяц за пользователя (при покупке базового пакета)

1.1.4 1С:Предприятие 8. Управление по целям и KPI

"1С:Предприятие 8. Управление по целям и KPI" - это решение для автоматизации процессов целеполагания, мониторинга и оценки эффективности деятельности сотрудников [13].

Продукт разработан на платформе "1С:Предприятие 8" и может быть интегрирован с другими конфигурациями 1С. Интерфейс представлен на рисунке 4.

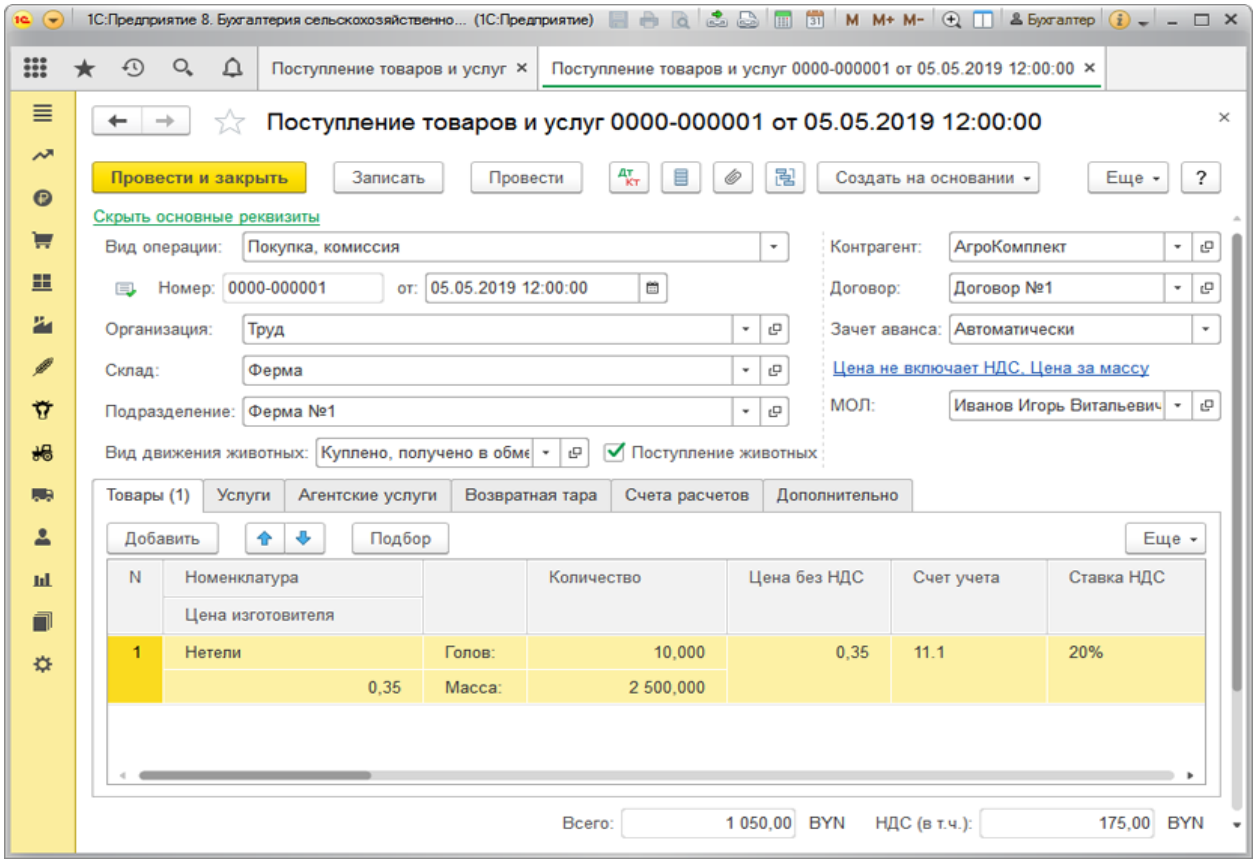

Рисунок 4 – Интерфейс 1С:Предприятие 8

Ключевые преимущества "1С:Предприятие 8. Управление по целям и KPI":

– поддержка различных методик управления эффективностью (BSC, KPI, управление по целям);

– интеграция с другими конфигурациями 1С (зарплата и управление персоналом, управление производственным предприятием);

– возможность гибкой настройки под специфику компании;

– относительно невысокая стоимость по сравнению с зарубежными аналогами.

Недостатки:

– ограниченный функционал в базовой версии продукта;

– отсутствие встроенных инструментов для сравнительного анализа эффективности сотрудников;

– сложность внедрения и поддержки системы силами внутренней ИТ-службы.

Стоимость: от 15 000 руб. за базовую версию продукта + стоимость лицензий на платформу 1С:Предприятие 8. Для сравнения систем мониторинга показателей эффективности персонала, рассмотренных в разделе 1.1, составим общую сравнительную таблицу.

Таблица 1 - Сравнение систем мониторинга показателей эффективности персонала

Критерий	SAP SuccessFactors Performance & Goals	Oracle Taleo Performance Management	WebTutor	1С:Предприятие 8. Управление по целям и KPI
Методики оценки	SMART-цели, иерархия целей	Цели, компетенции, OKR, 360 градусов	KPI, компетенции, управление по целям	BSC, KPI, управление по целям
Интеграция	SAP SuccessFactors модули	Oracle Taleo Cloud Service модули	WebTutor модули	1С конфигурации
Адаптация и кастомизация	Ограниченные возможности	Возможность настройки процессов	Гибкая настройка под специфику	Гибкая настройка под специфику
Мобильность	iOS и Android приложения	Не указано	Не указано	Не указано
Библиотеки целей	Есть	Не указано	Не указано	Не указано
Удобство внедрения	Высокая сложность	Высокая сложность	Требуется настройка и доработка под нужды	Сложность внедрения и поддержки
Стоимость	От $8/мес за пользователя	От $15/мес за пользователя	От 200 руб/мес за пользователя (базовый)	От 15000 руб (базовая версия) + лицензии 1С

Таким образом, каждая из рассмотренных систем имеет свои сильные и слабые стороны. SAP SuccessFactors и Oracle Taleo являются более широкофункциональными, но имеют высокую стоимость и сложность внедрения. WebTutor и 1С:Предприятие предлагают более доступные по цене решения с возможностью гибкой адаптации, но могут потребовать существенных усилий по настройке и расширению базового функционала.

Основными недостатками являются высокая стоимость внедрения и владения, необходимость существенной доработки под специфику бизнеса, а также ограниченные возможности для сравнительного анализа эффективности сотрудников в различных подразделениях и должностях. В связи с этим разработка специализированного решения для Филиала ФКУ “Налог-Сервис” ФНС России в Сахалинской области представляется целесообразной и экономически обоснованной.

1.2 Анализ бизнес-процессов Филиала ФКУ “Налог-Сервис” ФНС России в Сахалинской области, связанных с оценкой эффективности персонала

Филиал ФКУ “Налог-Сервис” ФНС России в Сахалинской области - подведомственное учреждение Федеральной налоговой службы, созданное в рамках проекта модернизации Налоговой службы как ИТ-учреждение, главной задачей которого является обеспечение деятельности цифровой технологической платформы ФНС России. С помощью Центров обработки данных реализуется непрерывная отказоустойчивая работа электронных сервисов Федеральной налоговой службы. Задачами ФКУ являются обеспечение информационно-вычислительной и телекоммуникационной инфраструктуры, анализ состояния и сопровождение внедренных программных

комплексов и технологических процессов ФНС России. Филиал имеет 5 отделений, которые располагаются на острове Сахалин – на расстоянии от 30 до 900 км от областного центра (г. Южно-Сахалинск) и на Курильских островах – на расстоянии от 500 до 1000 км.

Для обеспечения высокого качества услуг Филиал ФКУ “Налог-Сервис” ФНС России в Сахалинской области уделяет большое внимание управлению эффективностью персонала [3]. Рассмотрим основные бизнес-процессы филиала, связанные с оценкой эффективности сотрудников:

1.2.1 Годовая оценка эффективности

Раз в год (обычно в январе-феврале) в компании проводится годовая оценка эффективности сотрудников по итогам предыдущего года. Процесс включает в себя следующие этапы [5],[8],[12]:

– Самооценка сотрудника: каждый сотрудник заполняет форму самооценки, в которой указывает свои ключевые результаты и достижения за прошедший год, а также проводит оценку своих компетенций по установленной шкале.

– Оценка руководителя: непосредственный руководитель сотрудника проводит оценку его результатов и компетенций, основываясь на данных самооценки, а также на собственных наблюдениях и отзывах коллег.

– Калибровочная сессия: руководители подразделений проводят совместное обсуждение оценок сотрудников для обеспечения единых стандартов и снижения субъективности оценки.

– Обратная связь: руководитель проводит индивидуальную встречу с сотрудником, на которой обсуждает результаты оценки, дает развернутую обратную связь и определяет зоны развития на следующий год.

По итогам годовой оценки принимаются решения о пересмотре заработной платы, выплате бонусов, продвижении сотрудников, а также формируются индивидуальные планы развития.

Недостатки существующего процесса:

– большие временные затраты на сбор и обработку данных (заполнение форм самооценки, подготовка отчетов руководителей);

– субъективность оценок, особенно по компетенциям;

– сложность сравнения эффективности сотрудников из разных подразделений;

– отсутствие четкой связи между результатами оценки и стратегическими целями компании.

Для лучшего понимания данного процесса имеет смысл создать майндмеп (ментальную карту), что и было выполнено на рисунке 5.

Рисунок 5 – Ментальная карта рассматриваемого процесса

1.2.2 Оценка эффективности сотрудников ИТ

Для сотрудников, занятых информационными технологиями (начальники отделов, менеджеры, инженеры), в Филиале ФКУ “Налог-Сервис” ФНС России в Сахалинской области применяется специальная методика оценки эффективности, основанная на количественных показателях.

Основными метриками являются [9],[10],[17]:

- количество принятых/закрытых заявок за день;
- время принятия/закрытия одной заявки;
- точность выполнения заявок (%);
- количество ошибок при выполнении и закрытии заявки;
- соблюдение нормативов времени на выполнение операций (%);
- количество случаев нарушения регламента.

Руководители ежемесячно анализируют эффективность своих сотрудников и проводят рейтингование по каждому показателю.

На основе рейтингов принимаются решения о премировании лучших сотрудников, а также о необходимости дополнительного обучения или ротации для отстающих работников [5],[8],[12].

Недостатки существующего процесса:

- ориентация только на количественные показатели, без учета качественных аспектов работы (например, ответственное отношение к поручениям, взаимодействие с коллегами);
- отсутствие единой базы данных по эффективности сотрудников ИТ, затрудняющее сравнительный анализ и обмен лучшими практиками между отделами;
- недостаточная оперативность получения данных (раз в месяц), не позволяющая быстро реагировать на снижение эффективности отдельных сотрудников.

1.3 Формирование требований к разрабатываемому ПО на основе анализа предметной области

На основе анализа бизнес-процессов Филиала ФКУ “Налог-Сервис” ФНС России в Сахалинской области, связанных с оценкой эффективности персонала.

А также с учетом выявленных недостатков существующих решений на рынке, можно сформулировать следующие требования к разрабатываемому программному обеспечению для мониторинга показателей эффективности сотрудников [4],[11].

1.3.1 Функциональные требования

– Система должна обеспечивать возможность настройки показателей эффективности для различных категорий сотрудников (инженеры, бухгалтеры, водители и т.д.) с учетом специфики их деятельности.

– Система должна предоставлять возможность ручного ввода дополнительных данных для расчета показателей эффективности (например, результаты оценки компетенций, качественные показатели работы).

– Система должна обеспечивать хранение исторических данных по эффективности сотрудников и предоставлять инструменты для анализа динамики показателей.

– Система должна предоставлять возможность настройки целевых значений показателей эффективности и автоматически оценивать степень достижения целей сотрудниками.

– Система должна обеспечивать возможность сравнения эффективности сотрудников внутри подразделений и между подразделениями компании.

– Система должна предоставлять инструменты для визуализации данных по эффективности персонала (диаграммы, графики, таблицы).

– Система должна обеспечивать возможность формирования отчетов по эффективности сотрудников за различные периоды времени (месяц, квартал, год).

– Система должна предоставлять возможность настройки прав доступа для различных категорий пользователей (руководители, консультанты, сотрудники).

– Система должна обеспечивать интеграцию с корпоративным порталом и системой электронного документооборота компании.

1.3.2 Нефункциональные требования

– Система должна быть реализована в виде веб-приложения с возможностью доступа из корпоративной сети и через Интернет.

– Система должна обеспечивать высокую производительность и быстрый отклик при работе с большими объемами данных.

– Система должна обеспечивать сохранность и конфиденциальность данных в соответствии с требованиями законодательства РФ и внутренними политиками компании.

– Система должна иметь интуитивно понятный и удобный пользовательский интерфейс, адаптированный для работы на различных устройствах (ПК, планшеты, смартфоны).

– Система должна предусматривать возможность масштабирования и расширения функциональности для поддержки роста бизнеса и появления новых требований.

– Система должна иметь подробную документацию для пользователей и администраторов, а также обеспечивать возможность обучения сотрудников работе с системой.

– Система должна обеспечивать высокий уровень надежности и доступности (не менее 99,5% времени бесперебойной работы).

– Система должна предусматривать механизмы резервного копирования и восстановления данных в случае сбоев.

– Система должна обеспечивать возможность интеграции с другими информационными системами компании через стандартные протоколы и API.

– Система должна соответствовать корпоративным стандартам разработки и оформления пользовательского интерфейса.

1.3.3 Ограничения и допущения

– Разработка системы должна вестись с учетом существующей ИТ-инфраструктуры Филиала ФКУ “Налог-Сервис” ФНС России в Сахалинской области и обеспечивать совместимость с используемыми аппаратными и программными платформами.

– При разработке системы необходимо учитывать планируемые сроки, а также обеспечивать возможность поэтапного внедрения и тестирования отдельных модулей.

– Разработка системы должна вестись в соответствии с действующими в компании политиками и регламентами в области ИТ, информационной безопасности и управления проектами.

– При расчете показателей эффективности система будет использовать данные из существующих учетных систем компании, которые могут содержать неполную или некорректную информацию. Требуется предусмотреть механизмы валидации и очистки данных.

– Разработка системы будет вестись силами внутренней ИТ-службы Филиала ФКУ “Налог-Сервис” ФНС России в Сахалинской области с привлечением внешних консультантов по мере необходимости.

Выводы разделу 1

Проведен анализ существующих решений для мониторинга показателей эффективности персонала.

Приведена оценка эффективности сотрудников, занятых информационными технологиями.

Сформулированные требования к разрабатываемому программному обеспечению для мониторинга показателей эффективности персонала учитывают специфику бизнес-процессов Филиала ФКУ “Налог-Сервис” ФНС России в Сахалинской области.

Масштаб компании и объемы обрабатываемых данных, а также ожидания заказчика в части функциональности, производительности, удобства использования и безопасности системы.

Данные требования будут использованы в качестве основы для проектирования архитектуры и пользовательского интерфейса системы, выбора технологического стека и планирования этапов разработки.

2 Проектирование программного обеспечения

2.1 Разработка архитектуры веб-приложения «клиент-сервер»

При проектировании архитектуры веб-приложения "клиент-сервер" для Филиала ФКУ «Налог-Сервис» ФНС России в Сахалинской области планируется использовать фреймворк Flask на языке Python для серверной части и базу данных Microsoft SQL Server для хранения данных. Такой выбор технологий обусловлен их широкой распространенностью, мощностью и гибкостью, что позволит создать масштабируемое и надежное приложение.

Архитектура будет строиться на основе модели "клиент-сервер", где клиентская часть (веб-браузер) будет отвечать за отображение пользовательского интерфейса и взаимодействие с пользователем, а серверная часть будет обрабатывать запросы, выполнять бизнес-логику и обмениваться данными с базой данных. Для реализации клиентской части будут использованы языки HTML, CSS и JavaScript. HTML позволит создать структуру страниц, CSS - визуальное оформление, а JavaScript добавит интерактивность и динамику в пользовательский интерфейс. Взаимодействие с сервером будет осуществляться посредством HTTP-запросов [14].

Серверная часть будет реализована на фреймворке Flask, который обеспечит маршрутизацию URL-адресов, обработку запросов и формирование ответов. Структура проекта будет модульной с разделением функциональности на отдельные компоненты, такие как авторизация (auth), работа с базой данных (db), обработка ошибок (error_handlers), экспорт данных (export_options) и прочие. Это позволит улучшить читаемость, тестируемость и поддерживаемость кода [15].

Для взаимодействия с базой данных MS SQL Server будет использована ORM-библиотека SQLAlchemy, которая предоставляет высокоуровневый интерфейс для выполнения запросов и управления данными. SQLAlchemy позволяет описывать структуру БД с помощью моделей Python-классов и выполнять операции над данными, абстрагируясь от особенностей конкретной СУБД.

Аутентификация и авторизация пользователей будет реализована с использованием сессий Flask. Это позволит идентифицировать пользователей, контролировать доступ к защищенным ресурсам и персонализировать опыт взаимодействия с приложением. Для обеспечения безопасности и целостности данных планируется реализовать валидацию и санитизацию пользовательского ввода, использовать параметризованные SQL-запросы для предотвращения SQL-инъекций, а также применять шифрование для хранения чувствительных данных, таких как пароли пользователей. В качестве дополнительных компонентов рассматривается использование библиотеки Celery для выполнения фоновых и ресурсоемких задач, таких как генерация отчетов или отправка email-уведомлений. Для кэширования и хранения временных данных может быть использовано хранилище ключ-значение Redis.

После, имеет смысл спроектировать непосредственно систему в целом. Потому ниже будут представлены 2 диаграммы: диаграмма компонентов, представленная на рисунке 6, и диаграмма развертывания – рисунок 7.

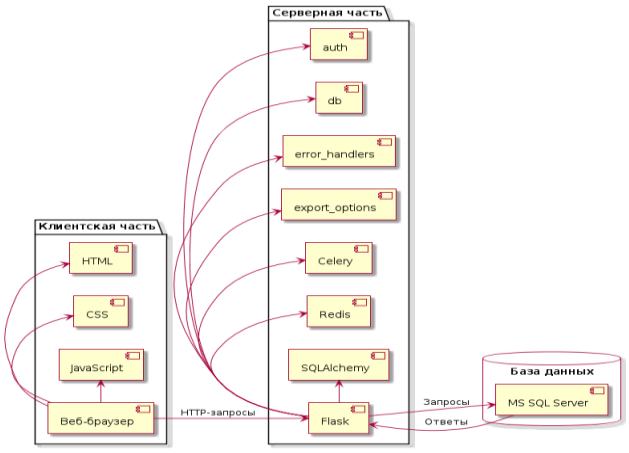

Рисунок 6 – Диаграмма компонентов

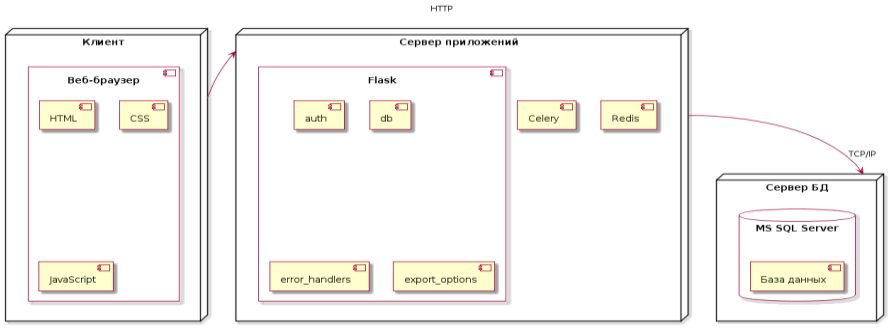

Рисунок 7 – Диаграмма развертывания

Диаграмма компонентов показывает взаимодействие между основными компонентами системы - клиентской частью (веб-браузер), серверной частью на Flask и базой данных MS SQL Server. Также отражены дополнительные компоненты, такие как Celery для фоновых задач и Redis для кэширования.

Диаграмма развертывания иллюстрирует физическое расположение компонентов системы на различных узлах (серверах). Клиентская часть выполняется на устройстве пользователя, серверная часть и дополнительные компоненты - на сервере приложений, а база данных располагается на выделенном сервере БД.

Представленная архитектура веб-приложения "клиент-сервер" на основе Flask, Python и MS SQL Server обеспечит надежность, производительность и масштабируемость решения для управления документооборотом в филиале ФКУ «Налог-Сервис» ФНС России в Сахалинской области. Модульный подход и использование проверенных библиотек и фреймворков позволят упростить разработку, тестирование и сопровождение приложения, а также заложат основу для дальнейшего развития и интеграции с другими системами при необходимости.

2.2 Концептуальное моделирование базы данных

Концептуальное моделирование является ключевым этапом в разработке базы данных для системы мониторинга показателей эффективности работы персонала компании. Оно позволяет определить основные сущности, их атрибуты и взаимосвязи, создавая высокоуровневое представление структуры данных.

В рамках концептуального моделирования были выделены следующие основные сущности:

– Сотрудник (Employee): представляет информацию о сотрудниках компании, включая их личные данные, должность, отдел и контактную информацию;

– Должность (Position): содержит информацию о должностях в компании, такую как название должности, описание обязанностей и требуемую квалификацию;

– Отдел (Department): хранит информацию об отделах компании, включая название отдела, руководителя и бюджет;

– Показатель эффективности (KPI - Key Performance Indicator): представляет различные показатели, используемые для оценки эффективности работы сотрудников, такие как количество выполненных задач, качество работы, соблюдение сроков и удовлетворенность клиентов;

– Оценка (Evaluation): содержит результаты оценки эффективности работы сотрудников по каждому показателю за определенный период времени;

– Период (Period): представляет временные периоды, за которые производится оценка эффективности, например, месяц, квартал или год.

На рисунке 8 приведена ER-диаграмма, иллюстрирующая связи между сущностями.

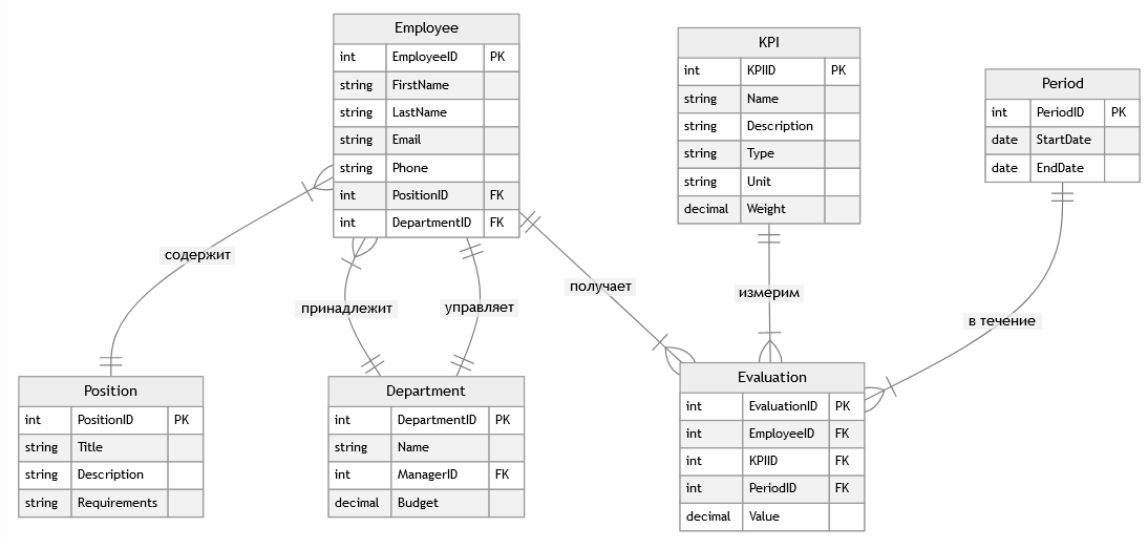

Рисунок 8 – ER-диаграмма

Данная ER-диаграмма показывает связи между сущностями:

– каждый сотрудник (Employee) занимает определенную должность (Position) и принадлежит к определенному отделу (Department);

– каждый отдел (Department) имеет руководителя (Employee);

– каждый сотрудник (Employee) получает оценки (Evaluation) по различным показателям эффективности (KPI) за определенные периоды времени (Period).

Физическая диаграмма базы данных представлена на рисунке 9.

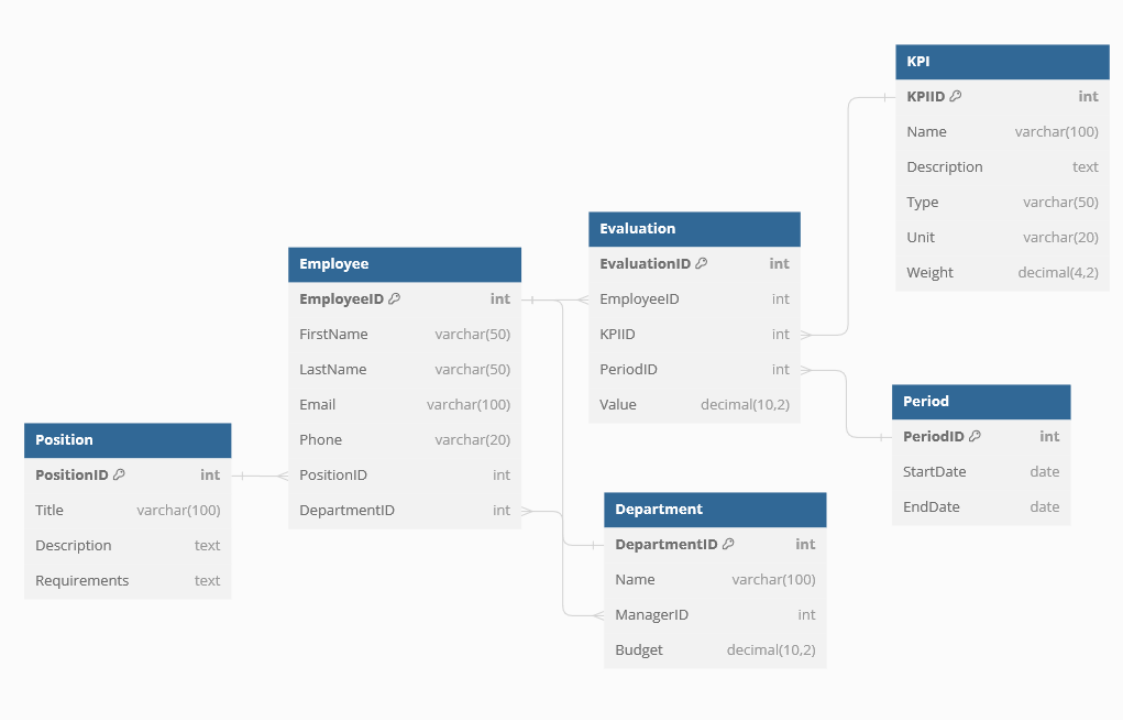

Рисунок 9 – Физическая диаграмма БД

Физическая диаграмма базы данных отображает структуру таблиц и связи между ними на уровне реляционной базы данных. Каждая сущность представлена отдельной таблицей с соответствующими атрибутами и первичными ключами (PK). Связи между таблицами обозначены внешними ключами (FK).

Таблица "Employee" содержит информацию о сотрудниках, включая их личные данные, должность (PositionID) и отдел (DepartmentID). Таблица

"Position" хранит информацию о должностях, а таблица "Department" - об отделах компании.

Таблица "KPI" содержит список показателей эффективности, используемых для оценки работы сотрудников. Каждый показатель имеет свое название, описание, тип, единицу измерения и вес.

Таблица "Evaluation" хранит результаты оценки эффективности работы сотрудников. Каждая запись связывает сотрудника (EmployeeID), показатель эффективности (KPIID) и период оценки (PeriodID) с соответствующим значением оценки (Value).

Таблица "Period" представляет периоды времени, за которые производится оценка эффективности, с указанием даты начала (StartDate) и даты окончания (EndDate) периода.

В целом, концептуальное моделирование и физическая реализация базы данных для системы мониторинга показателей эффективности работы персонала компании обеспечивают структурированное хранение данных, позволяющее эффективно оценивать и анализировать производительность сотрудников. ER-диаграмма дает наглядное представление о взаимосвязях между сущностями, а физическая диаграмма (в нотации DBML) показывает детальную структуру таблиц и связей между ними. Такая модель базы данных позволяет:

– хранить информацию о сотрудниках, их должностях и отделах;

– определять показатели эффективности (KPI) для оценки работы сотрудников;

– регистрировать результаты оценки эффективности за определенные периоды времени;

– анализировать данные для выявления тенденций, сравнения производительности сотрудников и принятия управленческих решений.

Дальнейшие шаги в разработке программного обеспечения мониторинга показателей эффективности работы персонала будут включать реализацию базы данных на выбранной СУБД, создание интерфейса для ввода и отображения данных, а также разработку алгоритмов анализа и формирования отчетов.

2.3 Проектирование пользовательского интерфейса

Пользовательский интерфейс является важнейшим компонентом программного обеспечения мониторинга показателей эффективности работы персонала компании. Хорошо продуманный и удобный интерфейс обеспечивает эффективное взаимодействие пользователей с системой, позволяет легко вводить, просматривать и анализировать данные о производительности сотрудников.

При проектировании пользовательского интерфейса руководствовались следующими принципами:

– простота и интуитивность: интерфейс должен быть понятным и легким в использовании даже для неопытных пользователей. Элементы управления и навигация должны быть логично организованы и соответствовать ожиданиям пользователей;

– консистентность: дизайн интерфейса должен быть единообразным во всех разделах приложения. Схожие действия и элементы должны выглядеть и работать одинаково, чтобы пользователи могли быстро освоиться;

– информативность: интерфейс должен предоставлять пользователям всю необходимую информацию для принятия решений и выполнения задач. Важные данные и показатели должны быть легко доступны и наглядно представлены;

– адаптивность: интерфейс должен корректно отображаться и работать на различных устройствах и разрешениях экрана. Необходимо обеспечить удобство использования как на десктопных компьютерах, так и на мобильных устройствах;

– эстетичность: визуальный дизайн интерфейса должен быть привлекательным, современным и соответствовать корпоративному стилю компании. Использование приятных цветов, иконок и шрифтов повышает удовольствие от работы с приложением.

Рассмотрим Основные компоненты пользовательского интерфейса.

Шапка (header): содержит логотип компании, название приложения и основное меню навигации. Меню состоит из пунктов "Главная", "Сотрудники", "Отделы", "KPI", "Отчеты" и др.

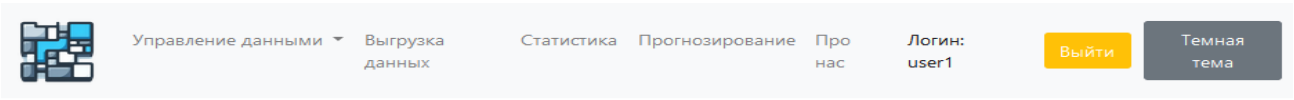

Рисунок 10 – Шапка

Основная рабочая область (main content area): в этой части отображается основной контент в зависимости от выбранного раздела.

Например, список сотрудников, детали отдела, форма для ввода результатов KPI и т.п.

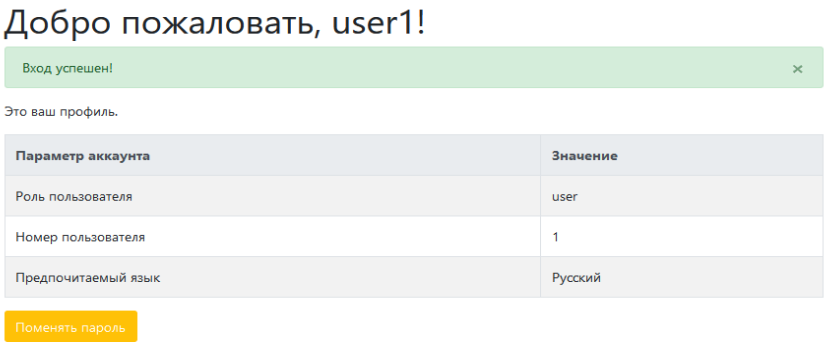

Рисунок 11 – Пример рабочей области (аккаунт пользователя)

Футер (footer): содержит дополнительную информацию, такую как копирайт, ссылки на условия использования и политику конфиденциальности.

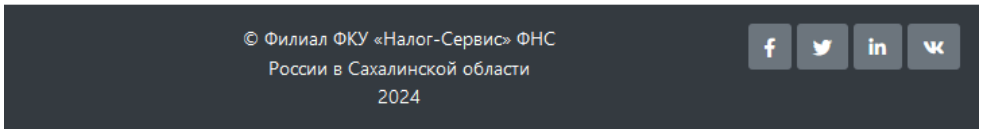

Рисунок 12 – Пример футера

Ниже представлены эскизы некоторых ключевых страниц интерфейса:

Страница задач:

- таблица со списком задач, содержащая даты и др.;
- поиск задач по различным критериям;
- возможность добавления, редактирования и удаления.

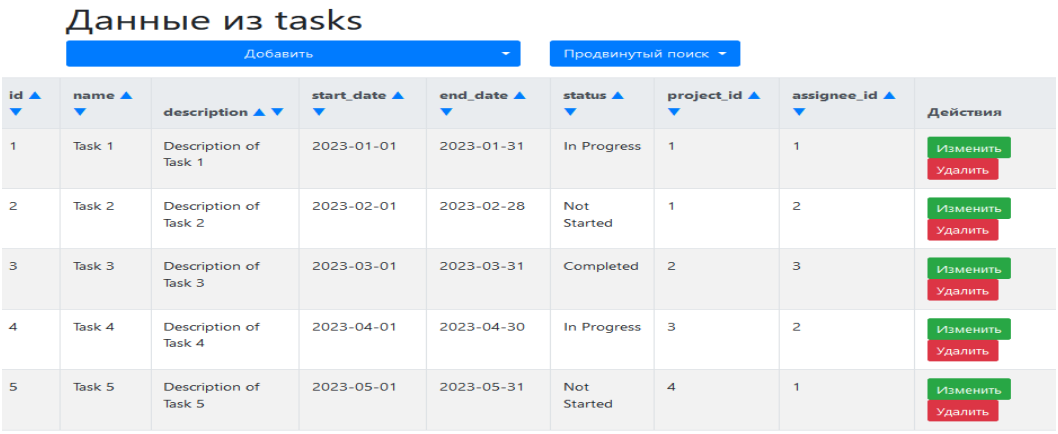

Рисунок 13 – Страница задач

Страница профиля сотрудника:

– фотография и основная информация о сотруднике (ФИО, должность, отдел, контакты и т.п.);

– возможность добавления новой оценки и просмотра детальных комментариев по каждой оценке.

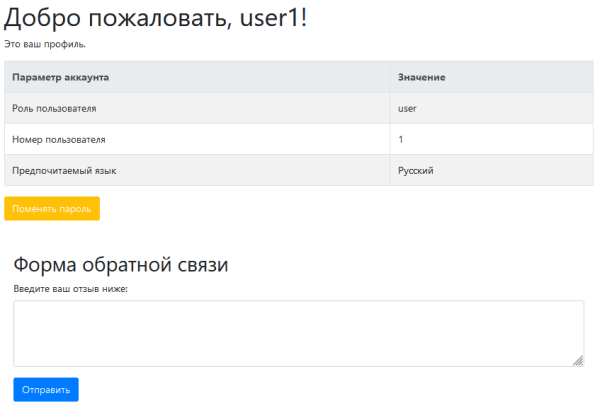

Рисунок 14 – Профиль сотрудника

Страница отчетов:

– выбор типа отчета (по сотрудникам, отделам, KPI и т.д.);

– возможность экспорта отчетов в различных форматах (PDF, Excel, CSV и т.п.).

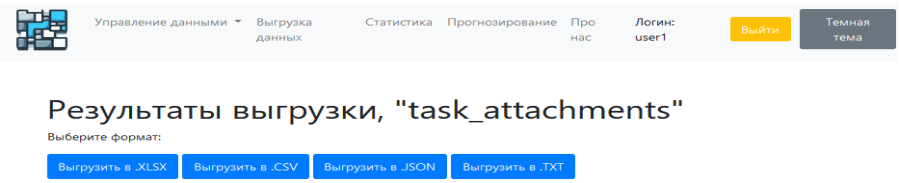

Рисунок 15 – Страница отчетов (варианты экспорта)

В процессе разработки интерфейса использовались современные технологии и фреймворки, такие как HTML, CSS, JavaScript, Bootstrap и др. Это позволило создать отзывчивый, интерактивный и визуально привлекательный интерфейс, обеспечивающий удобство работы пользователей. Особое внимание уделялось оптимизации производительности и скорости загрузки страниц. Применялись техники кэширования, минификации кода и ресурсов, ленивой загрузки изображений и др. Для обеспечения доступности интерфейса были реализованы функции локализации и поддержки различных языков. Пользователи могут выбрать предпочитаемый язык интерфейса, и все текстовые элементы, сообщения и подсказки будут отображаться на выбранном языке. Кроме того, были реализованы функции настройки внешнего вида интерфейса, такие как выбор цветовой схемы (светлая/темная тема), изменение размера шрифта и др. Это позволяет пользователям адаптировать интерфейс под свои предпочтения и особенности восприятия.

В целом, спроектированный пользовательский интерфейс отвечает всем требованиям и принципам удобства использования, обеспечивает эффективное взаимодействие пользователей с системой мониторинга показателей эффективности работы персонала и способствует повышению производительности труда в компании.

2.4 Описание алгоритмов тестирования ПО

Тестирование является важнейшей частью разработки программного обеспечения, позволяющей обеспечить высокое качество конечного продукта, его соответствие требованиям и ожиданиям пользователей. В данном разделе будут описаны основные алгоритмы и методики тестирования, применяемые для веб-приложения мониторинга эффективности персонала.

Для начала – модульное (юнит) тестирование предполагает проверку отдельных функций, классов и модулей программы на соответствие ожидаемому поведению. Для реализации модульного тестирования в Python широко используется фреймворк unittest, входящий в стандартную библиотеку языка. Рассмотрим пример модульного теста для функции проверки валидности email-адреса:

```
import unittest

def is_valid_email(email):
    # Простая регулярка для проверки email
    regex = r'^[\w\.-]+@[\w\.-]+\.\w+$'
    return True if re.match(regex, email) else False

class TestEmailValidator(unittest.TestCase):

    def test_valid_emails(self):
        valid_emails = ['user@example.com',
'name.surname@company.org', 'test123@mail.ru']
        for email in valid_emails:
            self.assertTrue(is_valid_email(email),
f"{email} should be a valid email address")

    def test_invalid_emails(self):
        invalid_emails = ['user@example',
'name@company', 'test@mail.']
        for email in invalid_emails:
            self.assertFalse(is_valid_email(email),
f"{email} should be an invalid email address")

if __name__ == '__main__':
    unittest.main()
```

В этом примере определен класс TestEmailValidator, наследующий от unittest.TestCase. Он содержит два метода, начинающихся с test_, в которых мы описываем тестовые сценарии для функции is_valid_email. Метод test_valid_emails проверяет, что функция корректно обрабатывает допустимые email-адреса, используя метод assertTrue для проверки условия. Метод test_invalid_emails проверяет, что функция корректно обрабатывает недопустимые email-адреса, используя метод assertFalse. Запуск тестов осуществляется через вызов unittest.main(). Фреймворк unittest предоставляет богатый набор методов для проверки различных условий, таких как равенство значений, принадлежность множеству, истинность/ложность выражений и т.д.

После модульного тестирования имеет смысл рассмотреть интеграционное. Интеграционное тестирование направлено на проверку взаимодействия между различными компонентами и модулями системы. Это позволяет выявить ошибки, связанные с некорректной передачей данных, несовместимостью интерфейсов или нарушением целостности данных. Для веб-приложения интеграционные тесты могут включать в себя:

- проверку маршрутизации и обработки HTTP-запросов;
- тестирование взаимодействия с базой данных (создание, чтение, обновление, удаление записей);
- проверку работы механизмов аутентификации и авторизации;
- тестирование интеграции с внешними сервисами и API (например, отправка email-уведомлений);
- проверку целостности данных при одновременном доступе нескольких пользователей.

Для реализации интеграционных тестов в Python часто используются библиотеки, такие как pytest и requests. Ниже приведен пример интеграционного

теста для проверки маршрута регистрации нового пользователя с использованием requests:

```
import requests

def test_user_registration():
    # Подготовка тестовых данных
    payload = {
        'username': 'testuser',
        'email': 'test@example.com',
        'password': 'testpassword'
    }

    # Отправка POST-запроса на регистрацию
    response =
requests.post('http://localhost:5000/register',
json=payload)

    # Проверка успешной регистрации
    assert response.status_code == 200

    # Проверка, что пользователь добавлен в базу данных
    # ...
```

В этом примере был отправлен POST-запрос на маршрут /register с помощью библиотеки requests, передавая данные для регистрации нового пользователя. Затем было проверено, что запрос был успешно обработан (статус-код 200), и выполняем дополнительные проверки, например, что пользователь действительно был добавлен в базу данных.

Также стоит упомянуть про функциональное тестирование. Функциональное тестирование направлено на проверку соответствия приложения функциональным требованиям и ожиданиям пользователей. Оно включает в себя тестирование различных сценариев использования, проверку корректности обработки пользовательского ввода, проверку валидации данных и обработки ошибок. Для реализации функционального тестирования веб-приложений часто используются инструменты для автоматизации тестирования,

такие как Selenium. Selenium позволяет имитировать действия пользователя в браузере, взаимодействовать с веб-интерфейсом и проверять результаты.

Ниже приведен пример функционального теста для проверки процесса авторизации с использованием Selenium:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

def test_login():
    # Создание экземпляра веб-драйвера
    driver = webdriver.Chrome()

    # Открытие страницы входа
    driver.get("http://localhost:5000/login")

    # Ввод учетных данных
    username_field =
driver.find_element_by_id("username")
    password_field =
driver.find_element_by_id("password")
    username_field.send_keys("testuser")
    password_field.send_keys("testpassword")

    # Отправка формы входа
    password_field.send_keys(Keys.RETURN)

    # Проверка успешного входа
    assert "Успешный вход" in driver.page_source

    # Завершение работы веб-драйвера
    driver.quit()
```

В этом примере был создан экземпляр веб-драйвера Chrome, открыто страницу входа, введено учетные данные пользователя и отправлено форму входа. Затем проверено, что на странице после успешного входа присутствует соответствующий текст. Selenium позволяет реализовывать сложные сценарии тестирования, включающие взаимодействие с различными элементами интерфейса, проверку визуального отображения и другие действия, имитирующие реальное использование приложения.

Также важным типом тестирования является нагрузочное. Нагрузочное тестирование предназначено для проверки производительности и масштабируемости приложения под высокой нагрузкой. Оно помогает выявить узкие места в системе, определить максимальную пропускную способность и оценить поведение приложения при одновременной работе большого количества пользователей. Для проведения нагрузочного тестирования веб-приложений часто используются инструменты, такие как Apache JMeter, Locust или k6. Эти инструменты позволяют имитировать нагрузку на систему, генерируя большое количество запросов и отслеживая показатели производительности, такие как время отклика, количество обработанных запросов в секунду и использование ресурсов сервера. Ниже приведен пример сценария нагрузочного тестирования с использованием Locust:

```
    from locust import HttpUser, task, between

    class WebsiteUser(HttpUser):
        wait_time = between(1, 5)  # Интервал между
запросами от 1 до 5 секунд

        @task
        def index(self):
            self.client.get("/")

        @task
        def login(self):
            self.client.post("/login", json={"username":
"testuser", "password": "testpassword"})

        @task
        def view_employees(self):
            self.client.get("/employees")

        # Другие задачи, имитирующие действия пользователей
```

В этом примере мы определяем класс WebsiteUser, наследующий от HttpUser из библиотеки Locust. Внутри класса определены задачи (@task), имитирующие различные действия пользователей, такие как просмотр главной

страницы, вход в систему и просмотр списка сотрудников. Параметр wait_time определяет интервал между выполнением задач для каждого экземпляра пользователя.

Запуск нагрузочного тестирования с помощью Locust позволит имитировать большое количество одновременных пользователей и оценить производительность приложения в условиях высокой нагрузки.

Результаты нагрузочного тестирования помогут выявить проблемы масштабируемости, узкие места в системе и определить оптимальную конфигурацию серверов для обеспечения требуемой производительности.

Выводы по разделу 2

Описанные в этом разделе алгоритмы и методики тестирования обеспечивают всестороннюю проверку веб-приложения мониторинга эффективности персонала на соответствие функциональным и нефункциональным требованиям.

Комплексное применение модульного, интеграционного, функционального и нагрузочного тестирования позволяет гарантировать высокое качество, надежность и производительность разработанного программного обеспечения.

3 Разработка программного обеспечения

3.1 Выбор и обоснование технологического стека (Python, Flask, MS SQL Server)

При разработке веб-приложения для мониторинга эффективности персонала в ФКУ "Налог-Сервис" ФНС России в Сахалинской области был выбран следующий технологический стек:

Язык программирования - Python

Python является одним из самых популярных языков программирования в мире благодаря своей простоте, читаемости кода, обширной экосистеме библиотек и фреймворков. Он позволяет быстро разрабатывать качественные приложения различной сложности. Основные преимущества Python:

- высокая производительность разработки за счет лаконичного синтаксиса и большого количества готовых модулей и библиотек;
- кроссплатформенность - Python-приложения можно запускать на любой ОС (Windows, Linux, macOS);
- активное сообщество разработчиков, большое количество учебных материалов и opensource-проектов;
- наличие развитых веб-фреймворков (Django, Flask), позволяющих быстро создавать полнофункциональные сайты и сервисы;
- широкие возможности для анализа и визуализации данных, машинного обучения (библиотеки NumPy, Pandas, Matplotlib и др.);
- хорошая интеграция с другими языками и технологиями (C/C++, Java, .NET и т.д.).

Немаловажно, что Python входит в число наиболее востребованных навыков у работодателей. По данным рейтинга TIOBE за июнь 2023 года,

Python занимает 1-е место по популярности среди языков программирования. Это позволит в дальнейшем, при необходимости, привлекать к поддержке и развитию системы квалифицированных специалистов.

Веб-фреймворк - Flask

Для создания серверной части веб-приложения был выбран фреймворк Flask. Это легковесный и гибкий фреймворк для разработки веб-приложений на языке Python. В отличие от Django, Flask предоставляет минималистичный набор инструментов и библиотек, позволяя разработчику самому выбирать нужные компоненты. Основные преимущества Flask:

- простота изучения и использования. Базовое приложение Flask может быть написано в одном файле и нескольких строках кода;
- гибкость и расширяемость. Flask не навязывает разработчику жесткую структуру проекта и позволяет использовать только те модули, которые нужны;
- встроенный веб-сервер для разработки и тестирования;
- поддержка шаблонов Jinja2 для динамической генерации веб-страниц на основе данных;
- встроенная поддержка юнит-тестирования;
- хорошая документация и активное сообщество разработчиков.

Кроме того, Flask хорошо подходит для разработки API и микросервисов, что может быть полезно для интеграции системы мониторинга с другими информационными системами ФКУ "Налог-Сервис".

```
    # Роут для индексной страницы (после успешного входа)
@app.route('/index')
@login_required(message="Cabinet is available only for
authorized users")
def index():
    return render_template('cabinet.html', )
```

В приведенном фрагменте кода видно, как с помощью Flask легко определяются маршруты (routes) к различным URL-адресам веб-приложения с

помощью декоратора @app.route(). Например, @app.route('/index') означает, что функция index() будет вызываться при обращении к URL-адресу /index. А декоратор @login_required позволяет ограничить доступ к определенным страницам только для авторизованных пользователей.

СУБД - Microsoft SQL Server

Для хранения данных в системе мониторинга используется СУБД Microsoft SQL Server. Это решение выбрано исходя из следующих факторов:

- MS SQL Server является основной СУБД, используемой в инфраструктуре ФНС России и ФКУ "Налог-Сервис". Использование единой платформы БД упростит интеграцию и взаимодействие систем;
- MS SQL Server обладает высокой производительностью, надежностью и масштабируемостью, позволяя обрабатывать большие объемы данных;
- наличие удобных инструментов для администрирования (SQL Server Management Studio) и богатый функционал T-SQL для работы с данными;
- развитые механизмы защиты данных, парольная аутентификация, шифрование и т.д.;
- хорошая интеграция с ОС семейства Windows Server, на которых развернута серверная инфраструктура ФКУ;
- наличие коннекторов и библиотек для подключения к MS SQL Server из Python-приложений (pymssql, pyodbc и др.).

Конечно, MS SQL Server является проприетарным ПО, однако ФКУ "Налог-Сервис" уже имеет лицензии и развернутую инфраструктуру на базе продуктов Microsoft, поэтому использование "родной" СУБД не потребует дополнительных затрат.

Другие компоненты стека:

Помимо основных компонентов (Python, Flask, MSSQL) в разработке веб-приложения используются следующие технологии и библиотеки:

– SQLAlchemy - ORM-библиотека для работы с БД из Python кода. Позволяет описывать структуру БД с помощью классов Python и выполнять запросы к БД без написания чистого SQL. В приведенном коде видно использование SQLAlchemy для работы с таблицей Translation;

– Flask-Mail - почтовая библиотека, встраиваемая во Flask-приложения. Позволяет отправлять почтовые уведомления из кода (например, при регистрации нового пользователя, восстановлении пароля и т.д.);

– Flask-Limiter - модуль для ограничения количества запросов к веб-приложению (rate limiting). Позволяет защитить приложение от DoS-атак и чрезмерной нагрузки. В коде видно подключение limiter к некоторым маршрутам;

– PyYAML - библиотека для чтения и записи файлов в формате YAML из Python-кода. В приложении используется для чтения спецификации OpenAPI;

– logging - встроенный в Python модуль для ведения логов работы приложения. В коде настраивается логирование и создается logger для маршрутов в приложении.

На стороне фронтенда будут использованы традиционные веб-технологии - HTML/CSS/JavaScript. Для придания интерактивности интерфейсу планируется использование jQuery. Также рассматривается подключение какого-либо CSS-фреймворка (Bootstrap, Material Design) для создания адаптивного и привлекательного интерфейса. Такой набор технологий полностью покрывает все потребности разрабатываемой системы, обеспечивает высокую скорость и удобство разработки, хорошую производительность и масштабируемость. Немаловажно, что используемые инструменты являются

проверенными и имеют обширную документацию и комьюнити, что облегчит поддержку и развитие системы силами ИТ-службы ФКУ "Налог-Сервис".

3.2 Реализация БД и серверной части веб-приложения

Реализация базы данных в веб-приложении для мониторинга эффективности персонала выполнена с использованием библиотеки SQLAlchemy на языке Python. SQLAlchemy - это популярная ORM (Object-Relational Mapping) библиотека, позволяющая работать с реляционными базами данных через объектно-ориентированный интерфейс.

Первым шагом в настройке SQLAlchemy является создание объекта Engine, который представляет собой абстракцию соединения с базой данных. Параметры подключения берутся из конфигурационного файла приложения app_config:

```
from sqlalchemy import create_engine
from config import app_config

engine = create_engine(app_config.db_path)
```

Далее создается фабрика сессий Session с помощью функции sessionmaker, которая связывается с созданным ранее объектом engine. Сессия представляет собой контекст выполнения запросов к БД и управления транзакциями:

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)
```

Перед началом работы с базой данных проверяется существование БД и создается пустой файл, если он отсутствует:

```
db_file = app_config.db_path.split('///')[1]
if not os.path.exists(db_file):
    open(db_file, 'a').close()
```

Для описания структуры базы данных в терминах SQLAlchemy используется декларативный подход. Создается базовый класс `Base`, от которого будут наследоваться все модели:

```
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
```

Затем определяются классы-модели, представляющие таблицы базы данных. Каждый класс наследуется от Base и содержит атрибуты, соответствующие столбцам таблицы. Связи между таблицами задаются с помощью внешних ключей (ForeignKey) и объектов отношений (relationship).

Далее представлен фрагмент кода с описанием некоторых моделей:

```
class Employee(Base):
    __tablename__ = 'employees'

    id = Column(Integer, primary_key=True)
    first_name = Column(String(50))
    last_name = Column(String(50))
    email = Column(String(100))
    phone = Column(String(20))
    position_id = Column(Integer,
ForeignKey('positions.id'))
    department_id = Column(Integer,
ForeignKey('departments.id'))

    position = relationship('Position',
back_populates='employees')
    department = relationship('Department',
back_populates='employees', foreign_keys=[department_id])
    evaluations = relationship('Evaluation',
back_populates='employee')


class Position(Base):
    __tablename__ = 'positions'

    id = Column(Integer, primary_key=True)
    title = Column(String(100))
    description = Column(Text)
    requirements = Column(Text)

    employees = relationship('Employee',
back_populates='position',
foreign_keys=[Employee.position_id])


class Department(Base):
    __tablename__ = 'departments'
```

```
        id = Column(Integer, primary_key=True)
        name = Column(String(100))
        manager_id = Column(Integer,
ForeignKey('employees.id'))
        budget = Column(Float)

        manager = relationship('Employee',
foreign_keys=[manager_id])
        employees = relationship('Employee',
back_populates='department',
foreign_keys=[Employee.department_id])
```

В каждой модели атрибут __tablename__ задает имя таблицы в БД. Поля описываются с помощью экземпляров класса Column, в которых указывается тип данных и дополнительные ограничения (первичный ключ, обязательность заполнения, уникальность и т.д.). Связи с другими таблицами определяются через ForeignKey (с указанием целевой таблицы и поля) и relationship (для создания объектов-отношений, по которым можно обращаться из кода). Аналогичным образом описаны модели для других сущностей предметной области (KPI, Evaluation, Period), а также служебные таблицы для авторизации и локализации (User, UserRole, Translation).

После определения всех моделей выполняется создание соответствующих таблиц в базе данных путем вызова метода create_all() объекта метаданных Base.metadata:

```
Base.metadata.create_all(engine)
```

SQLAlchemy автоматически генерирует и выполняет необходимые SQL-запросы на создание таблиц, учитывая все заданные связи и ограничения. Для удобства работы с моделями из кода создается словарь table_name_dict, сопоставляющий имена таблиц и соответствующие им классы моделей:

```
table_name_dict = {
    'employees': Employee,
    'positions': Position,
    'departments': Department,
    ...
```

```
}
```

Затем в приложении реализованы функции для начального заполнения БД тестовыми данными. Функция `generate_random_data` принимает на вход класс модели и генерирует для нее 10 случайных записей (если в соответствующей таблице еще нет данных):

```
def generate_random_data(model):
    session = Session()

    if not session.query(model).first():
        for _ in range(10):
            row = {}
            for column in model.__table__.columns:
                if column.primary_key:
                    continue
                elif isinstance(column.type, Integer):
                    row[column.name] = random.randint(1,
10)
                elif isinstance(column.type, Float):
                    row[column.name] = random.uniform(0,
100)
                elif isinstance(column.type, String):
                    row[column.name] =
''.join(random.choices(string.ascii_letters, k=8))
                elif isinstance(column.type, Text):
                    row[column.name] =
''.join(random.choices(string.ascii_letters, k=20))
                elif isinstance(column.type, Date):
                    start_date = datetime.now().date()
                    end_date = start_date +
timedelta(days=30)
                    row[column.name] = start_date if
column.name == 'start_date' else end_date

            session.add(model(**row))

        session.commit()
    session.close()
```

Эта функция в цикле проходит по всем полям модели (за исключением первичного ключа) и в зависимости от типа столбца генерирует подходящее случайное значение:

- для полей типа Integer выбирается случайное целое число в диапазоне от 1 до 10;
- для Float - случайное число с плавающей запятой от 0 до 100;
- для String и Text - случайная строка заданной длины из букв латинского алфавита;
- для полей с типом Date берется текущая дата и генерируются две даты - текущая и смещенная на 30 дней вперед (для полей с названиями start_date и end_date).

Сгенерированные данные добавляются в сессию через метод add(), а затем сессия фиксируется (commit()) и закрывается. Функция generate_random_data вызывается в цикле для всех моделей из словаря table_name_dict:

```
for model in table_name_dict.values():
    generate_random_data(model)
```

Дополнительно реализована функция `initialize_user_roles()` для инициализации предопределенных пользовательских ролей в системе. Она очищает таблицу `user_roles` и добавляет две роли - 'admin' (с полным доступом) и 'user' (с доступом только к основным таблицам). Строка с разрешенными таблицами формируется программно на основе списка моделей:

```
def initialize_user_roles():
    session = Session()

    only_admin_tables = ['translations', 'users',
'user_roles', 'feedback']
    all_tables = ', '.join([table_name for table_name in
table_name_dict.keys() if table_name not in
only_admin_tables])

    session.query(UserRole).delete()

    admin_role = UserRole(
        id=1,
        role_type='admin',
        allowed_tables=f"{', '.join(only_admin_tables)},
{all_tables}",
```

```
        allowed_search=1,
        allowed_add=1,
        allowed_delete=1,
        allowed_edit=1
    )
    user_role = UserRole(
        id=2,
        role_type='user',
        allowed_tables=all_tables,
        allowed_search=1,
        allowed_add=1,
        allowed_delete=1,
        allowed_edit=1
    )

    session.add(admin_role)
    session.add(user_role)
    session.commit()
    session.close()
```

Таким образом, при первом запуске приложения автоматически создаются все необходимые таблицы, заполняются тестовыми данными и инициализируются пользовательские роли. В дальнейшем доступ к данным из кода осуществляется через сессии SQLAlchemy. Например, для получения списка всех сотрудников достаточно выполнить запрос:

```
session = Session()
employees = session.query(Employee).all()
```

Аналогично выполняется добавление, изменение и удаление записей через методы add(), update(), delete() сессии с последующей фиксацией изменений через commit(). SQLAlchemy сама транслирует операции над объектами Python в соответствующие SQL-запросы, что значительно упрощает работу с БД по сравнению с чистым SQL.

Итак, в данном проекте библиотека SQLAlchemy используется для реализации ORM-слоя между Python-кодом и реляционной базой данных. Это позволяет работать с данными в терминах привычных объектов и классов, а не на уровне SQL, что повышает скорость разработки и сопровождаемость кода.

Кроме того, SQLAlchemy предоставляет ряд дополнительных возможностей, таких как контроль транзакций, связи между моделями, миграции схемы данных и т.д. Вместе с фреймворком Flask и другими компонентами SQLAlchemy образует мощный и гибкий стек технологий для разработки веб-приложений с серверным хранением данных.

3.3 Разработка клиентской части для сотрудников (внесение данных о работе)

Клиентская часть веб-приложения для мониторинга эффективности персонала, предназначенная для использования сотрудниками, реализована с помощью HTML, CSS и JavaScript. Основная задача этой части - предоставить удобный и интуитивно понятный интерфейс для просмотра, добавления, редактирования и удаления данных о работе сотрудников. Центральным компонентом клиентской части является страница управления таблицей (шаблон table.html). Этот шаблон динамически генерируется на сервере с помощью Flask и Jinja2 и отображает данные из выбранной таблицы базы данных (например, employees, departments, kpis, evaluations и т.д.).

Рассмотрим ключевые элементы страницы для внесения данных о работе.

Заголовок страницы с указанием названия таблицы.

```
    <h1>{{ session['translations'].get('data_from', 'Data
from') }} {{ table_name }}</h1>
```

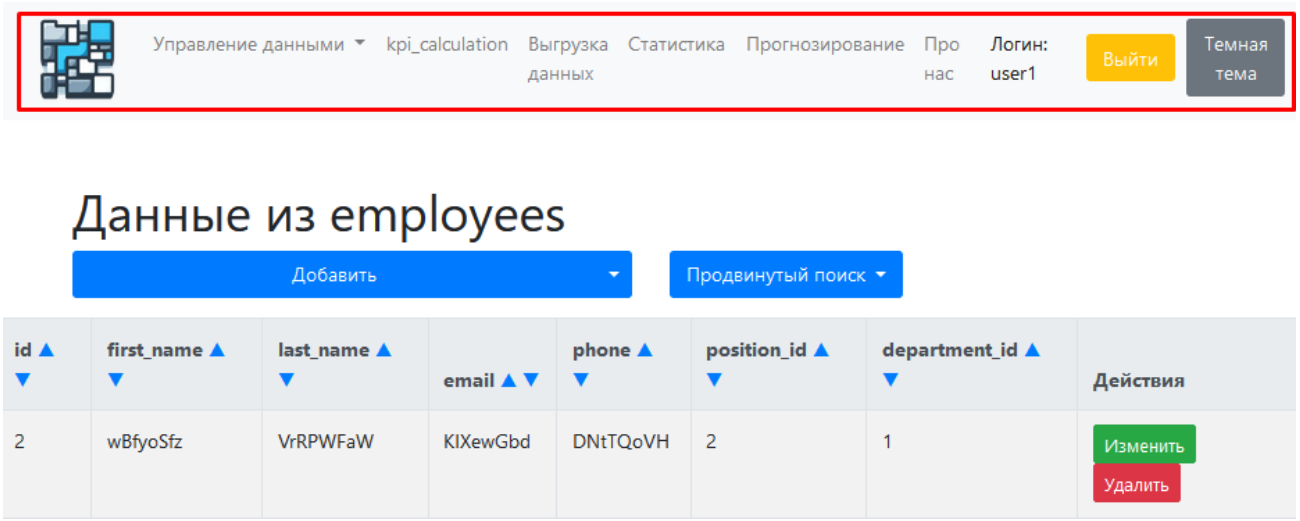

Рисунок 18 – Заголовок

Блок кнопок для добавления новых записей и импорта данных из файлов.

```
    <div class="btn-group btn-block">
        <a href="/add/{{ table_name }}" class="btn btn-
primary btn-block"
           {% if session.get('allowed_add') == 0
%}disabled{% endif %}>
            {{ session['translations'].get('add', 'Add') }}
        </a>
        <button type="button" class="btn btn-primary
dropdown-toggle dropdown-toggle-split"
                data-toggle="dropdown" aria-haspopup="true"
aria-expanded="false"
                {% if session.get('allowed_add') == 0
%}disabled{% endif %}>
            <span class="sr-only">Toggle Dropdown</span>
        </button>
        <div class="dropdown-menu">
            {% for format in app_config.import_formats %}
                <form action="/add/{{ table_name }}/{{
format }}" method="POST" enctype="multipart/form-data"
                      class="dropdown-item">
                    <a class="dropdown-item">{{
session['translations'].get('add_from', 'Add from') }}
                        .{{ format }}</a>
                    <input type="file" name="{{ format
}}File" accept=".{{ format }}" class="form-control-file">
                    <button type="submit" class="btn btn-
primary btn-sm mt-2">
```

```
                            {{
session['translations'].get('upload', 'Upload') }} .{{
format }}
                        </button>
                    </form>
                    {% if not loop.last %}<div class="dropdown-
divider"></div>{% endif %}
                {% endfor %}
            </div>
        </div>
```

Здесь используется компонент btn-group из Bootstrap для создания выпадающего меню с опциями импорта из разных форматов файлов (CSV, Excel). Кнопка "Добавить" и меню импорта доступны только для пользователей с соответствующими правами (allowed_add).

Блок расширенного поиска данных в таблице.

```
    <div class="dropdown">
        <button class="btn btn-primary dropdown-toggle"
type="button" id="dropdownMenuButton"
                data-toggle="dropdown" aria-haspopup="true"
aria-expanded="false">
    ...
                                <button type="submit" class="btn
btn-success mb-2">
                                    {{
session['translations'].get('search', 'Search') }}
                                </button>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
```

В этом фрагменте также используется выпадающее меню Bootstrap для отображения формы расширенного поиска. Поля критериев поиска генерируются динамически с помощью JavaScript при нажатии кнопки "Добавить критерий поиска". Пользователь может выбрать столбец для поиска, тип условия (точное значение или диапазон) и ввести значения для фильтрации.

Сформированные критерии поиска передаются на сервер через GET-параметры при отправке формы.

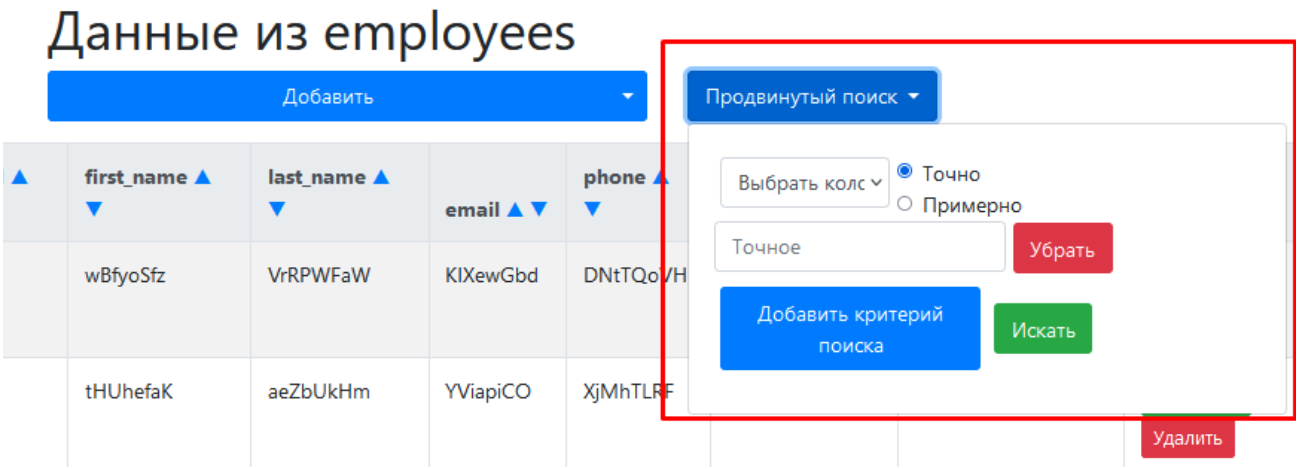

Рисунок 19 – Продвинутый поиск

Таблица формируется динамически на основе переданных с сервера данных (table_data) и списка имен столбцов (column_names). Для каждой записи отображаются значения полей с возможностью обрезки длинных текстовых значений (с помощью класса text-truncate) и всплывающей подсказкой при наведении (data-toggle="tooltip").

В последнем столбце таблицы размещены кнопки для редактирования и удаления записей. Их видимость зависит от прав пользователя (allowed_edit, allowed_delete).

```
<div class="d-flex justify-content-center">
    <div class="pagination">
        {{ pagination.links }}
    </div>
</div>
```

Для постраничного отображения данных используется объект pagination, сформированный на сервере с помощью библиотеки flask-paginate. Ссылки на страницы генерируются автоматически и выводятся в шаблоне.

Страница добавления/редактирования записи (add_record.html, edit_record.html) имеет схожую структуру и логику. Она содержит динамически сформированную форму на основе модели SQLAlchemy и позволяет пользователю вводить значения для создания новой или изменения существующей записи в базе данных. Вот фрагмент кода для страницы редактирования:

```
    <form method="POST">
        <div class="form-group row">
            {% for column_name in column_names %}
                <label for="{{ column_name }}" class="col-
sm-2 col-form-label">{{ column_name }}</label>
                <div class="col-sm-10">
                    {{ form[column_name](class='form-
control', id=column_name) }}
                </div>
            {% endfor %}
        </div>
        <div class="form-group row">
            <div class="col-sm-10">
                <button type="submit" class="btn btn-
primary">
                    {{
session['translations'].get('save_changes', 'Save changes')
}}
                </button>
                <a href="{{ url_for('manage_table',
table_name=table_name) }}" class="btn btn-secondary">
                    {{ session['translations'].get('cancel',
'Cancel') }}
                </a>
            </div>
        </div>
    </form>
```

Поля формы генерируются в цикле на основе переданного списка имен столбцов (column_names). Объект form представляет собой динамический класс

формы WTForms, созданный на сервере в соответствии с моделью таблицы. Каждое поле формы привязывается к соответствующему столбцу модели и имеет необходимые валидаторы.

Таким образом, клиентская часть веб-приложения для сотрудников предоставляет удобный интерфейс для работы с данными о показателях эффективности. Пользователи могут просматривать записи в виде таблицы с возможностью сортировки, пагинации и расширенного поиска по различным критериям. Для добавления и редактирования записей используются динамически сформированные формы с валидацией данных на стороне сервера.

3.4 Разработка административной части (управление сотрудниками, настройка показателей)

Административная часть веб-приложения для мониторинга эффективности персонала предоставляет функционал для управления сотрудниками, настройки показателей эффективности и работы с другими справочными данными. Доступ к этой части ограничен только для пользователей с ролью администратора.

Основным компонентом административной части является универсальная страница для управления таблицами базы данных (шаблон table.html). Эта страница динамически генерируется на основе имени таблицы, переданного в URL, и позволяет выполнять стандартные операции CRUD (Create, Read, Update, Delete) над записями таблицы.

Далее будут представлен ключевые возможности страницы управления таблицами.

Отображение списка записей таблицы с пагинацией и сортировкой.

```
    <table class="table table-responsive-sm table-striped
table-bordered table-hover">
```

```
        <thead class="thead-light">
            <tr>
                {% for column in column_names %}
                    <th>
                        {{ column }}
                        <a href="?sort={{ column
}}&direction=asc">&#9650;</a>
                        <a href="?sort={{ column
}}&direction=desc">&#9660;</a>
                    </th>
                {% endfor %}
                <th>{{
session['translations'].get('actions', 'Actions') }}</th>
            </tr>
        </thead>
        <tbody>
            {% for row in table_data %}
                <tr>
                    {% for column in column_names %}
                        <td>
                            <span class="text-truncate d-
inline-block" style="max-width: 150px;"
                                  data-toggle="tooltip"
title="{{ row.get(column) }}">
                                {{ row.get(column) }}
                            </span>
                        </td>
                    {% endfor %}
                    <td>
                        <!-- Кнопки для редактирования и
удаления записи -->
                    </td>
                </tr>
            {% endfor %}
        </tbody>
    </table>

    <div class="d-flex justify-content-center">
        <div class="pagination">
            {{ pagination.links }}
        </div>
    </div>
```

Записи таблицы выводятся в виде HTML-таблицы, где каждый столбец соответствует полю модели. Длинные текстовые значения обрезаются с помощью класса text-truncate и снабжаются всплывающей подсказкой при

наведении (data-toggle="tooltip"). Заголовки столбцов являются интерактивными и позволяют сортировать таблицу по возрастанию или убыванию соответствующего поля. Для реализации сортировки генерируются ссылки вида ?sort=<column>&direction=<asc|desc>, которые обрабатываются на сервере. Постраничная навигация реализована с помощью объекта pagination (библиотека flask-paginate), который генерирует ссылки на страницы в соответствии с текущими параметрами запроса.

Расширенный поиск записей по различным критериям.

```
    <div class="dropdown">
        <button class="btn btn-primary dropdown-toggle"
type="button" id="dropdownMenuButton"
                data-toggle="dropdown" aria-haspopup="true"
aria-expanded="false">
            {{ session['translations'].get('adv_search',
'Advanced search') }}
        </button>
        <div class="dropdown-menu p-4" aria-
labelledby="dropdownMenuButton">
            <div class="d-flex">
                <form action="" method="GET" class="form"
id="advancedSearchForm">
                    <div id="searchCriteriaContainer">
                        <!-- Динамически добавляемые
критерии поиска -->
                    </div>
                    <div class="form-row align-items-
center">
                        <div class="col">
                            <button type="button" class="btn
btn-primary mb-2" id="addSearchCriteria">
                                {{
session['translations'].get('add_search_criteria', 'Add
search criteria') }}
                            </button>
                        </div>
                        <div class="col">
                            <button type="submit" class="btn
btn-success mb-2">
                                {{
session['translations'].get('search', 'Search') }}
                            </button>
                        </div>
```

```
                    </div>
                </form>
            </div>
        </div>
    </div>
```

Для организации расширенного поиска используется выпадающее меню с формой, в которой администратор может динамически добавлять критерии поиска. Каждый критерий состоит из выбора поля, типа условия (точное значение или диапазон) и значений для фильтрации. Добавление новых критериев выполняется без перезагрузки страницы с помощью JavaScript. Сформированные критерии поиска передаются на сервер через GET-параметры вида searchCriteria[i][column], searchCriteria[i][type], searchCriteria[i][value] и т.д., где i - порядковый номер критерия. На сервере эти параметры обрабатываются и динамически встраиваются в SQL-запрос для фильтрации записей.

Кнопка "Найти" инициирует отправку формы поиска и обновление таблицы с учетом указанных критериев.

Добавление новых записей в таблицу.

```
    <div class="btn-group btn-block">
        <a href="/add/{{ table_name }}" class="btn btn-
primary btn-block"
           {% if session.get('allowed_add') == 0
%}disabled{% endif %}>
            {{ session['translations'].get('add', 'Add') }}
        </a>
        <button type="button" class="btn btn-primary
dropdown-toggle dropdown-toggle-split"
                data-toggle="dropdown" aria-haspopup="true"
aria-expanded="false"
                {% if session.get('allowed_add') == 0
%}disabled{% endif %}>
            <span class="sr-only">Toggle Dropdown</span>
        </button>
        <div class="dropdown-menu">
            <!-- Опции импорта данных из файлов -->
        </div>
    </div>
```

Для добавления новой записи используется кнопка "Добавить", которая ведет на отдельную страницу с формой ввода данных. Шаблон формы (add_record.html) генерируется динамически на основе модели таблицы с помощью библиотеки WTForms. Поля формы и их типы определяются автоматически по типам столбцов в базе данных. Кроме ручного ввода, администратору доступна возможность импорта данных из файлов различных форматов (CSV, Excel). Для этого используется выпадающее меню с формами загрузки файлов. После выбора файла и нажатия кнопки "Загрузить" данные из файла парсятся и добавляются в таблицу.

Редактирование и удаление существующих записей.

```
    <td>
        {% if session.get('allowed_edit') == 0 %}
            <button class="btn btn-success btn-sm" disabled>
                {{ session['translations'].get('change',
'Change') }}
            </button>
        {% else %}
            <a href="/edit/{{ table_name }}/{{ row['id'] }}"
class="btn btn-success btn-sm">
                {{ session['translations'].get('change',
'Change') }}
            </a>
        {% endif %}

        {% if session.get('allowed_delete') == 0 %}
            <button class="btn btn-danger btn-sm" disabled>
                {{ session['translations'].get('delete',
'Delete') }}
            </button>
        {% else %}
            <a href="/delete/{{ table_name }}/{{ row['id']
}}" class="btn btn-danger btn-sm">
                {{ session['translations'].get('delete',
'Delete') }}
            </a>
        {% endif %}
    </td>
```

Для каждой записи в таблице выводятся кнопки "Изменить" и "Удалить". Кнопка "Изменить" ведет на страницу редактирования записи (edit_record.html), где данные загружаются в форму, аналогичную форме добавления. После внесения изменений и сохранения записи данные обновляются в базе данных. Кнопка "Удалить" ведет на страницу подтверждения удаления (delete_record.html), где администратор должен явно подтвердить операцию. После подтверждения запись безвозвратно удаляется из базы данных. Видимость и доступность кнопок редактирования и удаления зависит от прав текущего пользователя (allowed_edit, allowed_delete), которые определяются его ролью и настройками безопасности.

Перечисленные возможности доступны для всех основных таблиц приложения: employees (сотрудники), departments (отделы), positions (должности), kpis (показатели эффективности), periods (периоды оценки) и т.д. Административный интерфейс обеспечивает удобный доступ к этим данным и позволяет оперативно управлять ими. Кроме основных таблиц, в административной части также доступны страницы для управления пользователями системы (таблицы users и user_roles). Администратор может просматривать список пользователей, добавлять новых, редактировать данные и роли существующих пользователей, блокировать и удалять учетные записи. Для поддержки многоязычности интерфейса в системе предусмотрена таблица translations, которая хранит текстовые ресурсы на различных языках. Администраторы могут редактировать переводы через тот же табличный интерфейс.

Интерактивность и динамичность административного интерфейса достигается за счет использования JavaScript и библиотеки jQuery. Например, скрипт динамического добавления полей поиска:

```
$(document).ready(function() {
    let searchIndex = 0;
```

```
        function addSearchRow() {
            // Формирование HTML-кода для новой строки
критериев поиска
            const searchRow = $('<div class="form-row mb-
2"></div>');
            // ...

            // Добавление обработчиков событий для элементов
строки
            // ...

            // Добавление строки в контейнер
            $('#searchCriteriaContainer').append(searchRow);
            searchIndex++;
        }

        // Обработчик нажатия кнопки "Добавить критерий"
        $('#addSearchCriteria').click(addSearchRow);

        // Первоначальное добавление одной строки
        addSearchRow();
    });
```

Этот скрипт отслеживает нажатие кнопки "Добавить критерий", динамически формирует HTML-код для новой строки критериев, добавляет обработчики событий для элементов строки (переключение типа условия, удаление строки) и вставляет сформированный код в форму поиска. Серверная часть интерфейса управления реализована в виде отдельного модуля (modules/tables.py), который регистрирует обработчики маршрутов для различных операций над таблицами:

```
    def create_table_routes(app):
        # ...

        @app.route('/table/<table_name>')
        @login_required(message="Table management is
available only for authorized users")
        @table_access_required
        def manage_table(table_name):
            # Получение данных таблицы, обработка параметров
сортировки и поиска, пагинация
            # ...
```

```
        @app.route('/add/<table_name>', methods=['GET',
'POST'])
        @login_required(message="Adding records is available
only for authorized users")
        @table_access_required
        def add_record(table_name):
            # Генерация формы добавления записи, обработка
отправки формы
            # ...
```

Функция create_table_routes регистрирует обработчики для маршрутов /table/<table_name> (просмотр и поиск записей), /add/<table_name> (добавление записи), /edit/<table_name>/<record_id> (редактирование записи) и /delete/<table_name>/<record_id> (удаление записи). Все обработчики используют декораторы @login_required и @table_access_required для проверки авторизации пользователя и его прав доступа к соответствующей таблице.

Функция manage_table отвечает за отображение списка записей таблицы. Она извлекает из базы данных записи с учетом текущих параметров сортировки и поиска, выполняет пагинацию и передает данные в шаблон table.html. Функции add_record, edit_record и delete_record реализуют операции добавления, редактирования и удаления записей соответственно. Они динамически генерируют формы на основе модели таблицы (с помощью библиотеки WTForms), обрабатывают отправку форм и выполняют соответствующие изменения в базе данных.

Расчет же эффективности персонала выполнен так:

```
    @app.route('/kpi_calculation', methods=['GET', 'POST'])
        @login_required(message="KPI calculation is
available only for authorized users")
        def kpi_calculation():
            if request.method == 'POST':
                employee_id = request.form['employee_id']
                period_id = request.form['period_id']

                sessionx = Session()
                try:
```

```
                    employee =
sessionx.query(Employee).filter(Employee.id ==
employee_id).first(
                    period =
sessionx.query(Period).filter(Period.id ==
period_id).first()

                    kpis = sessionx.query(KPI).all()

                    kpi_results = {}
                    kpi_results['KPI1'] =
calculate_kpi1(kpi_values)
                    kpi_results['KPI2'] =
calculate_kpi2(kpi_values)
                    kpi_results['KPI3'] =
calculate_kpi3(kpi_values)
                    kpi_results['KPI4'] =
calculate_kpi4(kpi_values)
                    kpi_results['KPI5'] =
calculate_kpi5(kpi_values)
                    kpi_results['KPI6'] =
calculate_kpi6(kpi_values)
                    kpi_results['KPI7'] =
calculate_kpi7(kpi_values)

                    # Расчет общего показателя эффективности
                    ...
                    return
render_template('kpi_results.html',

employee=employee,
                                                period=period,

kpi_results=kpi_results)
```

И на стороне веб-сайта выглядит следующим образом:

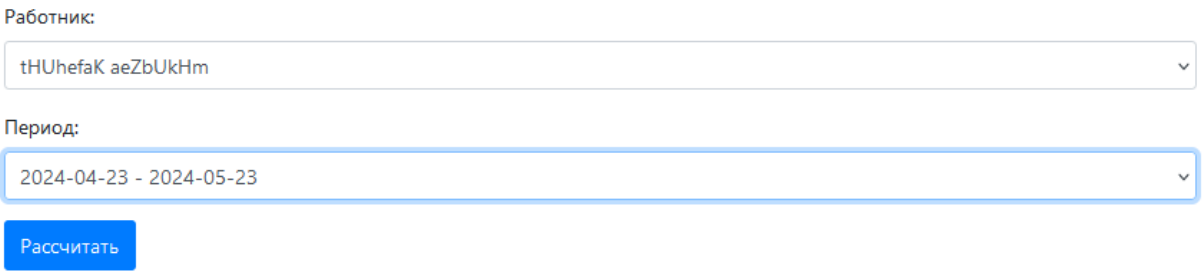

Рисунок 21 – Выбор работника и периода

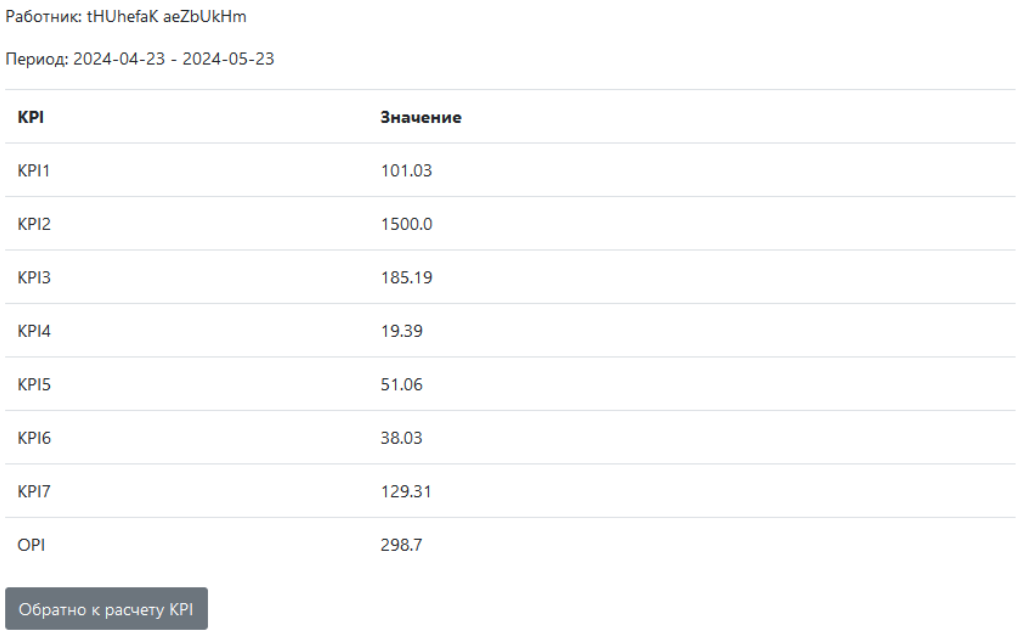

Рисунок 22 – Вывод KPI по сотруднику

В целом, разработанный административный интерфейс обеспечивает полный контроль над данными системы мониторинга эффективности персонала. Он предоставляет удобные инструменты для управления сотрудниками, отделами, должностями, показателями KPI и другой справочной информацией. За счет использования общего подхода на основе динамической генерации страниц и форм административная часть является гибкой и расширяемой.

3.5 Интеграция подсистем, тестирование и отладка ПО

После разработки отдельных компонентов системы мониторинга эффективности персонала (базы данных, серверной части, клиентских интерфейсов для сотрудников и администраторов) необходимо выполнить их интеграцию в единое работоспособное приложение. Интеграция подразумевает установление взаимодействия между подсистемами, проверку корректности передачи данных и управления, а также итоговое тестирование всего приложения на соответствие требованиям.

Первым шагом является развертывание приложения на целевой платформе. Для этого необходимо подготовить серверную инфраструктуру, установить и настроить необходимые компоненты (веб-сервер, СУБД, интерпретатор Python и т.д.), развернуть код приложения и выполнить начальную инициализацию базы данных. В случае использования контейнерной технологии (Docker) этот процесс может быть автоматизирован с помощью соответствующих конфигурационных файлов (Dockerfile, docker-compose.yml). После развертывания необходимо проверить базовую работоспособность приложения, запустив его и выполнив несколько простых сценариев использования (авторизация пользователя, переход по основным страницам, добавление и изменение записей в справочниках и т.п.). Эта проверка позволяет быстро выявить проблемы, связанные с некорректной конфигурацией окружения или ошибками в процессе сборки и развертывания.

Далее следует этап тщательного функционального тестирования всех подсистем и интерфейсов пользователя. Для каждого компонента системы подготавливается набор тестовых сценариев, охватывающих основные варианты использования и граничные случаи. Примеры тестовых сценариев:

а) Аутентификация пользователей:

1) вход в систему с корректными учетными данными;

2) вход в систему с некорректным паролем;

3) попытка доступа к защищенным страницам без авторизации;

4) проверка корректности выхода из системы;

б) Управление данными о сотрудниках:

1) обавление нового сотрудника с корректными данными;

2) попытка добавления сотрудника с некорректными данными (пустые обязательные поля, недопустимые значения);

3) редактирование данных существующего сотрудника;

4) удаление сотрудника и проверка, что связанные с ним данные также удалены;

в) Расчет показателей эффективности:

1) запуск расчета показателей для сотрудника за указанный период при наличии всех необходимых исходных данных;

2) попытка расчета показателей при отсутствии или некорректности исходных данных;

3) проверка корректности вычисления показателей по каждой из формул;

4) проверка возможности выгрузки результатов расчета в различных форматах (Excel, PDF и т.п.);

г) Импорт данных из внешних файлов:

1) загрузка файла с корректной структурой и данными;

2) попытка загрузки файла в неподдерживаемом формате;

3) попытка загрузки файла с нарушением структуры данных;

4) обработка ситуации, когда при загрузке возникают ошибки в части строк (некорректные значения, нарушение ссылочной целостности и т.п.);

д) Разграничение доступа:

1) проверка, что пользователь с ролью сотрудника не имеет доступа к административным функциям;

2) проверка, что пользователь не может просматривать и изменять данные, не относящиеся к его зоне ответственности;

3) проверка механизма делегирования полномочий.

Тестовые сценарии выполняются вручную или автоматизируются с помощью инструментов и фреймворков для тестирования (например, Selenium, pytest и т.п.). Автоматизация тестирования позволяет многократно запускать тесты после каждого изменения в коде, что снижает трудоемкость регрессионного тестирования.

Далее представлен примеры модульных тестов с использованием фреймворка unittest для основных функций приложения:

```
class ModularTestCase(unittest.TestCase):

    def setUp(self):
        self.app = app.test_client()
        self.app.testing = True

        # Set up a test database session
        engine = create_engine(app_config.db_path)
        self.session = sessionmaker(bind=engine)()

        # Create necessary tables and seed data for
testing, if needed
        self.user_id = randint(1, 1000)
        self.test_user = User(id=self.user_id,
email='test@example.com', username='testuser',
password='testpassword')
        self.session.add(self.test_user)
        self.session.commit()

    def test_language_change_valid(self):
```

```
            response = self.app.get('/language/English',
follow_redirects=True)
            self.assertEqual(response.status_code, 200)

        def test_language_change_invalid(self):
            response = self.app.get('/language/unknownlang',
follow_redirects=True)
            self.assertEqual(response.status_code, 200)

        def test_index_route(self):
            response = self.app.get('/index',
follow_redirects=True)
            self.assertEqual(response.status_code, 200)
```

Эти тесты проверяют следующее:

- изменение языка интерфейса на допустимый и недопустимый язык;
- доступность основных страниц приложения (главная, о нас, статистика);
- корректность обработки несуществующих маршрутов;
- работу механизма ограничения частоты запросов для отправки обратной связи;
- возможность отправки и сохранения обратной связи в базе данных.

Методы setUp и tearDown выполняются перед каждым тестом и после него соответственно. Они отвечают за инициализацию тестового окружения (создание тестового клиента, сессии БД, добавление тестовых данных) и очистку после теста (удаление тестовых данных).

Запуск модульных тестов выполняется командой:

```
python -m unittest discover tests
```

Это запустит все тесты из файлов test_*.py в папке tests. Результаты запуска показаны ниже.

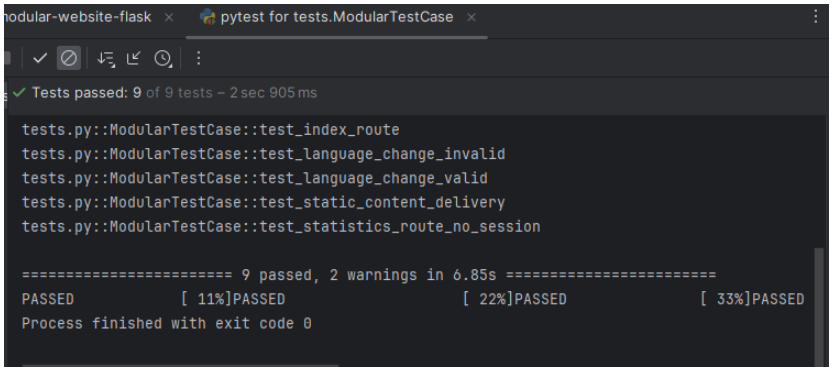

Рисунок 20 – Результат выполнения тестов

Как видим – все тесты успешно пройдены. Стоит однако заметить, что помимо функционального тестирования, было проведено также нефункциональное тестирование, которое включало следующие виды тестирования.

Тестирование производительности - проверка времени отклика системы, количества одновременно обслуживаемых пользователей, утилизации ресурсов сервера под нагрузкой.

Тестирование безопасности - проверка защищенности приложения от несанкционированного доступа, SQL-инъекций, XSS-атак и других угроз.

Тестирование удобства использования (usability) - оценка интерфейсов пользователя на интуитивность, удобство навигации, визуальное оформление.

По результатам всех видов тестирования был сформирован отчет, содержащий выявленные дефекты и несоответствия, их приоритет и рекомендации по исправлению. Исправление ошибок и доработка приложения по замечаниям было выполнено на этапе отладки.

При внесении исправлений были соблюдены принципы чистого кода, выполнен рефакторинг для улучшения читаемости и поддерживаемости кода.

Все изменения были сопровождены соответствующими модульными и интеграционными тестами.

Выводы по разделу 3

Таким образом, произведен выбор и обоснование технологического стека (Python, Flask, MS SQL Server). Реализация базы данных в веб-приложении для мониторинга эффективности персонала выполнена с использованием библиотеки SQLAlchemy на языке Python.

На этапе интеграции подсистем и тестирования разработанное веб-приложение прошло всестороннюю проверку на соответствие функциональным и нефункциональным требованиям, исправлены дефекты, подготовлены документация и план развертывания.

Это позволило обеспечить высокое качество конечного продукта и его готовность к вводу в эксплуатацию и использованию конечными пользователями.

Заключение

В рамках данной выпускной квалификационной работы была достигнута основная цель - разработано веб-приложение для мониторинга показателей эффективности работы персонала ФКУ "Налог-Сервис" ФНС России в Сахалинской области.

Приложение позволяет автоматизировать процесс сбора, обработки и анализа данных о результативности сотрудников, предоставляет руководству удобные инструменты для принятия управленческих решений и повышает прозрачность и объективность оценки персонала.

В ходе выполнения работы были решены следующие задачи:

– проведен обзор и анализ существующих решений для мониторинга показателей эффективности персонала в компаниях;

– проанализированы бизнес-процессы ФКУ "Налог-Сервис" ФНС России в Сахалинской области, связанные с оценкой эффективности персонала;

– спроектирована архитектура веб-приложения по модели "клиент-сервер" с использованием фреймворка Flask на языке Python для серверной части и СУБД Microsoft SQL Server для хранения данных;

– разработана серверная часть приложения на языке Python с использованием фреймворка Flask, ORM-библиотеки SQLAlchemy, а также ряда вспомогательных модулей и сервисов. Реализовано взаимодействие с СУБД MS SQL Server, организована обработка пользовательских запросов, управление сессиями и аутентификацией, формирование динамических веб-страниц;

– разработаны интерфейсы пользователей для сотрудников и администраторов системы;

– проведено комплексное тестирование разработанного приложения, включая модульное тестирование отдельных функций и компонентов (с

помощью фреймворка unittest), интеграционное тестирование взаимодействия подсистем, а также приемочное тестирование на соответствие требованиям заказчика.

Разработанное веб-приложение полностью соответствует сформулированным требованиям и реализует весь необходимый функционал для автоматизации мониторинга эффективности персонала ФКУ "Налог-Сервис".

Оно позволяет:

- проводить оценку работы сотрудников на основе количественных показателей, учитывающих специфику их деятельности;
- наглядно отображать динамику показателей с помощью графиков, диаграмм, сводных таблиц;
- формировать аналитические отчеты для руководства с различными разрезами и группировками данных;
- обеспечивать сбор данных от пользователей через веб-интерфейс с поддержкой импорта из внешних файлов;
- предоставлять администраторам гибкие возможности по управлению структурой данных, показателями, нормативами, пользователями;
- обеспечивать разграничение доступа на основе ролевой модели в соответствии с полномочиями сотрудников.

Использование системы мониторинга эффективности персонала позволит ФКУ "Налог-Сервис" получить ряд ощутимых преимуществ:

- повысить мотивацию и вовлеченность сотрудников за счет прозрачности и объективности оценки результатов их труда;
- повысить производительность и качество работы персонала за счет регулярной обратной связи и выявления зон развития;

– улучшить обмен информацией между сотрудниками и руководством, способствуя лучшему пониманию целей компании и повышению уровня сотрудничества;

– оптимизировать затраты на персонал за счет выявления неэффективных сотрудников и подразделений;

– обеспечить обоснованность кадровых решений (найм, ротация, обучение, сокращение) с учетом объективных данных о результативности;

– сократить временные и трудовые затраты на сбор и обработку данных по показателям эффективности;

– обеспечить руководство актуальной и достоверной информацией о состоянии трудовых ресурсов для целей стратегического управления.

Таким образом, поставленная цель ВКР достигнута, задачи полностью выполнены, а разработанное приложение готово к внедрению и использованию в ФКУ "Налог-Сервис".

Список используемой литературы

1. Аксенова Е. А. Управление персоналом : учебник / Е. А. Аксенова, П. В. Малиновский, Н. М. Малиновская : ЮНИТИ-ДАНА, 2017. — 560 с.
2. Анализ бизнес-процессов компании, связанных с оценкой эффективности персонала
3. Асалиев А. М. Оценка персонала в организации : учебное пособие / Г. Г. Вукович, О. Г. Кириллова, Е. А. Косарева, А. М. Асалиев : ИНФРА-М, 2023. — 171 с.
4. Вигерс К. Разработка требований к программному обеспечению / К. Вигерс, Д. Битти. - 3-е изд., доп. - Москва : Русская редакция, 2019. — 736 с.
5. Грекул В. И. Проектирование информационных систем: учебник и практикум для вузов / В. И. Грекул, Н. Л. Коровкина, Г. А. Левочкина. — Москва : Издательство Юрайт, 2023. — 385 с.
6. Дейнека А. В. Управление персоналом организации : учебник для вузов / А. В. Дейнека. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2022. — 423 с.
7. Долганова О. И. Моделирование бизнес-процессов : учебник и практикум для вузов / О. И. Долганова, Е. В. Виноградова, А. М. Лобанова. — Москва : Издательство Юрайт, 2023. — 289 с.
8. Заботина Н. Н. Проектирование информационных систем : учебное пособие / Н. Н. Заботина : ИНФРА-М, 2022. — 331 с.
9. Зараменских Е. П. Управление жизненным циклом информационных систем: учебник и практикум для вузов / Е. П. Зараменских. — 2-е изд. — Москва : Издательство Юрайт, 2021. — 497 с.

10. Золотухина Е. Б. Управление жизненным циклом информационных систем (продвинутый курс) : учебник и практикум для вузов / Е. Б. Золотухина, С. А. Красникова, А. С. Вишня : КУРС, 2017. — 119 с.

11. Кибанов А. Я. Управление персоналом организации : учебник / А.Я. Кибанов. — Москва : ИНФРА-М, 2022. — 695 с.

12. Коваленко В. В. Проектирование информационных систем : учебное пособие / В. В. Коваленко : Издательство ФОРУМ, 2023. — 357 с.

13. Михеева Е. В. Современные информационные системы управления бизнес-процессами организации : учебное пособие / Е.В. Михеева, О.М. Титова. — Москва : ИНФРА-М, 2021. — 376 с.

14. Назарова О. Б. Моделирование бизнес-процессов : учебно-методическая литература / О. Б. Назарова, О. Е. Масленникова : ФЛИНТА, 2023. — 261 с.

15. Проектирование информационных систем : учебник и практикум для вузов / Д. В. Чистов [и др.]. — Москва : Издательство Юрайт, 2023. — 258 с.

16. Суслов Г. В. Управление персоналом организации : учебное пособие / Г. В. Суслов : РИОР, 2020. — 154 с.

17. Формирование требований к разрабатываемому ПО на основе анализа предметной области

18. Хадасевич Н. Р. Оценка персонала в организации : учебник и практикум для вузов / Н. Р. Хадасевич. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2021. — 195 с.

19. Oracle Taleo Cloud for Dummies / Chris Lynch, Lois Elkins. — John Wiley & Sons Inc, 2022. — 320 p.

20. SAP SuccessFactors Learning: The Comprehensive Guide / Karishma S., Tauseef S. — 1st Ed. — Rheinwerk Publishing, 2020. — 460 p.

Приложение А

Исходный программный код

Рисунок 8, исходники

```
@startuml

skin rose
left to right direction


package "Клиентская часть" {
    [Веб-браузер] -> [HTML]
    [Веб-браузер] -> [CSS]
    [Веб-браузер] -> [JavaScript]
}

package "Серверная часть" {
    [Flask] -> [auth]
    [Flask] -> [db]
    [Flask] -> [error_handlers]
    [Flask] -> [export_options]
    [Flask] -> [Celery]
    [Flask] -> [Redis]
    [Flask] -> [SQLAlchemy]
}

database "База данных" {
    [MS SQL Server]
}

[Веб-браузер] --> [Flask] : HTTP-запросы
[Flask] --> [MS SQL Server] : Запросы
[Flask] <-- [MS SQL Server] : Ответы

@enduml
```

Рисунок 9, исходники

```
@startuml

skin rose

node "Клиент" {
    component [Веб-браузер] {
        [HTML]
        [CSS]
        [JavaScript]
    }
}

node "Сервер приложений" {
    component [Flask] {
        [auth]
        [db]
        [error_handlers]
        [export_options]
    }
    component [Celery]
    component [Redis]
}

node "Сервер БД" {
    database "MS SQL Server" {
        [База данных]
    }
}

[Клиент] -right-> [Сервер приложений] : HTTP
[Сервер приложений] --> [Сервер БД] : TCP/IP
@enduml
```

Рисунок 10, исходники

```
erDiagram
    Employee {
        int EmployeeID PK
        string FirstName
        string LastName
        string Email
        string Phone
        int PositionID FK
        int DepartmentID FK
    }

    Position {
        int PositionID PK
        string Title
        string Description
        string Requirements
    }

    Department {
        int DepartmentID PK
        string Name
        int ManagerID FK
        decimal Budget
    }

    KPI {
        int KPIID PK
        string Name
        string Description
        string Type
        string Unit
        decimal Weight
    }

    Evaluation {
        int EvaluationID PK
        int EmployeeID FK
        int KPIID FK
        int PeriodID FK
        decimal Value
```

```
}

Period {
    int PeriodID PK
    date StartDate
    date EndDate
}

Employee }|--|| Position : "содержит"
Employee }|--|| Department : "принадлежит"
Department ||--|| Employee : "управляет"
Employee ||--|{ Evaluation : "получает"
KPI ||--|{ Evaluation : "измерим"
Period ||--|{ Evaluation : "в течение"
```

Рисунок 11, исходники

```
Table Employee {
  EmployeeID int [pk]
  FirstName varchar(50)
  LastName varchar(50)
  Email varchar(100)
  Phone varchar(20)
  PositionID int [ref: > Position.PositionID]
  DepartmentID int [ref: > Department.DepartmentID]
}
Table Position {
  PositionID int [pk]
  Title varchar(100)
  Description text
  Requirements text
}
Table Department {
  DepartmentID int [pk]
  Name varchar(100)
  ManagerID int [ref: > Employee.EmployeeID]
  Budget decimal(10,2)
}
Table KPI {
  KPIID int [pk]
  Name varchar(100)
  Description text
  Type varchar(50)
```

```
  Unit varchar(20)
  Weight decimal(4,2)
}
Table Evaluation {
  EvaluationID int [pk]
  EmployeeID int [ref: > Employee.EmployeeID]
  KPIID int [ref: > KPI.KPIID]
  PeriodID int [ref: > Period.PeriodID]
  Value decimal(10,2)
}
Table Period {
  PeriodID int [pk]
  StartDate date
  EndDate date
}
```