ИТ-документация проекта «Планирование отпусков»

Проект: Веб-приложение "Планирование отпусков"

Версия: 2.0

Дата: 19.12.2025

Содержание

1. Общие сведения о проекте

В рамках производственной практики разработан учебно-практический прототип информационной системы для планирования и согласования отпусков сотрудников компании среднего размера. Проект выполнен в формате командной разработки с применением современных технологий веб-разработки, контейнеризации и реляционных баз данных.

Разрабатываемая система предназначена для автоматизации процессов подачи, согласования и учёта отпусков сотрудников, а также для повышения прозрачности планирования отсутствий внутри подразделений организации. Проект ориентирован на решение типовой бизнес-задачи, актуальной для организаций, использующих разрозненные или неформализованные способы учёта отпусков.

2. Назначение и цели системы

Назначением системы является создание единого программного решения для автоматизации процессов планирования, согласования и учёта отпусков сотрудников организации.

Основной целью проекта является устранение проблем, возникающих при неформализованном управлении отпусками, таких как использование бумажных заявлений, переписка по электронной почте и ведение учёта в электронных таблицах. Подобные подходы приводят к потере данных, отсутствию прозрачности, конфликтам в расписании и одновременному отсутствию ключевых сотрудников.

Разрабатываемая система решает следующие задачи:

- централизует процесс подачи и согласования заявок на отпуск;
- обеспечивает прозрачность информации о запланированных отпусках внутри отделов;
- позволяет учитывать баланс отпускных дней каждого сотрудника;
- снижает нагрузку на HR-отдел за счёт автоматизации отчётности;
- предоставляет руководителям инструмент контроля загрузки подразделений.

Система ориентирована на использование в реальных условиях и может рассматриваться как прототип коммерческого программного продукта.

3. Область применения системы

Разрабатываемая система предназначена для использования в компаниях малого и среднего размера, в которых отсутствуют специализированные корпоративные HR-системы либо процессы управления отпусками реализованы с применением электронных таблиц и неформальных средств коммуникации (электронная почта, бумажные заявления).

Система ориентирована на автоматизацию процесса планирования отпусков сотрудников, согласования заявок с учётом загрузки отделов и ведения централизованного учёта отпускных дней. Решение может применяться в организациях с иерархической структурой управления и несколькими подразделениями.

Благодаря использованию реляционной базы данных и модульной архитектуры система может быть масштабирована при увеличении количества пользователей и подразделений без изменения логики работы.

4. Архитектура системы

Проект реализован с использованием клиент-серверной архитектуры, что является типовым и широко применяемым подходом при разработке современных веб-приложений. Данная архитектура предполагает логическое разделение системы на серверную и клиентскую части, каждая из которых выполняет строго определённые функции.

Серверная часть приложения (backend) отвечает за реализацию бизнес-логики системы. В её задачи входит обработка запросов от клиентской части, проверка прав доступа пользователей, выполнение операций с базой данных, а также контроль корректности вводимых данных. Backend предоставляет набор HTTP-эндпоинтов, через которые осуществляется взаимодействие с системой, и выступает центральным звеном, обеспечивающим целостность и согласованность данных.

Реляционная база данных PostgreSQL используется для централизованного хранения информации о пользователях, отделах, заявках на отпуск, балансах отпускных дней и истории согласований. Выбор PostgreSQL обусловлен поддержкой транзакций, механизмов обеспечения целостности данных и возможностью масштабирования при увеличении объёма информации.

Клиентская часть приложения (frontend) реализована в виде веб-интерфейса и предназначена для взаимодействия пользователей с системой. Пользовательский интерфейс обеспечивает ввод данных, отображение информации о заявках и балансах отпусков, а также визуализацию календарей отпусков подразделений. Клиентская часть не содержит бизнес-логики и взаимодействует с сервером исключительно посредством HTTP-запросов.

Инфраструктура контейнеризации на базе Docker и Docker Compose используется для объединения всех компонентов системы в единое

изолированное окружение. Контейнеризация позволяет обеспечить воспроизводимость запуска проекта, упростить развёртывание и исключить зависимость работы системы от особенностей конкретной операционной среды. Взаимодействие между контейнерами осуществляется по внутренней сети Docker, что повышает безопасность и стабильность работы приложения.

5. Технологический стек

В процессе разработки проекта был использован современный и широко применяемый технологический стек, обеспечивающий надёжность, расширяемость и удобство сопровождения системы.

Язык программирования Python выбран в качестве основного языка серверной части благодаря своей читаемости, большому количеству библиотек и активному сообществу. Использование Python позволяет ускорить разработку бизнес-логики и упростить дальнейшую поддержку кода.

Веб-фреймворк Flask или FastAPI применяется для реализации серверного приложения. Данные фреймворки предоставляют инструменты для обработки HTTP-запросов, маршрутизации, валидации данных и реализации REST API. FastAPI дополнительно обеспечивает высокую производительность и автоматическую генерацию документации API, что упрощает разработку и тестирование.

Система управления базами данных PostgreSQL используется для хранения всех данных системы. PostgreSQL поддерживает транзакции, внешние ключи и ограничения целостности, что позволяет гарантировать корректность данных и реализовать сложные связи между сущностями предметной области.

HTML и CSS применяются для построения структуры и оформления пользовательского интерфейса. **Bootstrap** используется в качестве CSS-фреймворка для создания адаптивного и унифицированного интерфейса, обеспечивающего корректное отображение системы на различных устройствах.

Библиотека FullCalendar.js используется для визуализации отпусков в виде календаря. Данная библиотека позволяет наглядно отображать периоды отсутствия сотрудников, реализовывать временные шкалы и повышает удобство восприятия информации пользователями.

Docker и Docker Compose применяются для контейнеризации приложения и автоматизации процесса развёртывания. Контейнеризация позволяет запускать проект в идентичном окружении на различных системах, упрощает настройку инфраструктуры и снижает вероятность ошибок, связанных с различиями в среде выполнения.

Система контроля версий Git используется для организации командной разработки. Git обеспечивает отслеживание изменений в коде, совместную работу нескольких разработчиков, а также возможность возврата к предыдущим версиям проекта.

Система автоматизации GitHub Actions используется для организации непрерывной интеграции и доставки проекта. Она позволяет автоматически запускать тесты, проверять качество кода, собирать и развёртывать приложение при каждом изменении в репозитории. Использование GitHub Actions упрощает процесс поддержки и масштабирования проекта, сокращает количество ручных операций и снижает вероятность ошибок при релизе.

Использование данного технологического стека обеспечивает переносимость проекта, воспроизводимость окружения разработки и тестирования, а также удобство командной работы и дальнейшего развития системы.

6. Структура базы данных

Для хранения и обработки данных в проекте используется реляционная база данных PostgreSQL. Выбор реляционной модели обусловлен необходимостью строгого соблюдения целостности данных, наличием чётко определённых связей между сущностями и поддержкой транзакций, что особенно важно для систем, работающих с кадровой информацией.

Структура базы данных разработана на основе анализа предметной области и отражает основные бизнес-сущности, участвующие в процессе планирования и согласования отпусков. При проектировании учитывались роли пользователей, иерархия управления, необходимость учёта отпускных дней и фиксации всех действий, связанных с изменением статусов заявок.

В базе данных используются следующие основные таблицы:

- **departments** — предназначена для хранения информации об отделах компании и параметров, влияющих на бизнес-логику согласования отпусков;
- **users** — содержит данные обо всех пользователях системы, включая сотрудников, руководителей и HR-специалистов;
- **vacation_balances** — используется для учёта доступных и использованных отпускных дней сотрудников по календарным годам;
- **vacation_requests** — хранит информацию о заявках на отпуск, поданных сотрудниками, а также их текущий статус;
- **request_history** — обеспечивает хранение истории всех действий по заявкам, что позволяет отслеживать процесс согласования и обеспечивает прозрачность принимаемых решений.

Структура базы данных была предварительно спроектирована в виде логической схемы связей между сущностями. Использование ER-модели на

этапе проектирования позволило выявить и устранить возможные логические ошибки, определить корректные типы связей и обеспечить нормализацию данных до этапа физической реализации таблиц в PostgreSQL. Такой подход повышает надёжность системы и упрощает её дальнейшее сопровождение и развитие.

7. Описание таблиц базы данных

В данном разделе приведено подробное описание структуры базы данных информационной системы планирования отпусков. Каждая таблица отражает отдельную сущность предметной области и используется для реализации конкретных бизнес-процессов системы. Проектирование таблиц выполнено с соблюдением принципов реляционной модели данных, нормализации и обеспечения ссылочной целостности.

7.1 Таблица departments

Таблица departments предназначена для хранения справочной информации об организационной структуре компании. Данные, содержащиеся в данной таблице, используются для группировки сотрудников по подразделениям, а также для контроля допустимой нагрузки на отдел в период отпусков.

Каждая запись таблицы соответствует одному отделу организации. Наличие отдельной таблицы отделов позволяет централизованно управлять структурой компании и при необходимости изменять параметры подразделений без внесения изменений в данные пользователей.

Основные поля таблицы:

- id — уникальный идентификатор отдела. Используется в качестве первичного ключа и применяется для связи с таблицей пользователей;
- name — наименование отдела. Используется для отображения информации в пользовательском интерфейсе;
- max_simultaneous_vacations — максимально допустимое количество сотрудников данного отдела, которые могут находиться в отпуске одновременно. Данное поле используется при проверке заявок на отпуск и предотвращает превышение допустимой нагрузки на подразделение.

7.2 Таблица users

Таблица users является ключевой таблицей системы и предназначена для хранения информации обо всех пользователях, зарегистрированных в системе. В таблице хранятся данные сотрудников, руководителей подразделений и HR-специалистов.

Каждый пользователь имеет уникальную учётную запись, которая используется для аутентификации и авторизации в системе. Таблица users также отражает организационную иерархию компании и принадлежность сотрудников к конкретным отделам.

Основные поля таблицы:

- id — уникальный идентификатор пользователя, используемый в качестве первичного ключа;
- login — уникальный логин пользователя, применяемый при авторизации в системе;
- password — пароль пользователя, хранящийся в зашифрованном виде;
- role — роль пользователя в системе, определяющая набор доступных функций (сотрудник, руководитель, HR);
- full_name — полное имя пользователя, отображаемое в интерфейсе системы;
- department_id — внешний ключ, указывающий на отдел, к которому относится пользователь;
- manager_id — внешний ключ, ссылающийся на идентификатор руководителя пользователя. Поле используется для реализации иерархии подчинённости внутри организации.

7.3 Таблица vacation_balances

Таблица vacation_balances предназначена для хранения информации о балансе отпускных дней сотрудников. Учёт отпусков ведётся по годам, что позволяет корректно обрабатывать ежегодное начисление отпускных дней и формировать отчётность за разные периоды.

Для каждого пользователя в таблице может существовать несколько записей, соответствующих разным календарным годам. Это обеспечивает гибкость системы и возможность расширения логики расчёта отпусков в будущем.

Основные поля таблицы:

- user_id — идентификатор пользователя, для которого ведётся учёт отпускных дней;
- year — календарный год, за который учитывается баланс;
- total_days — общее количество отпускных дней, доступных пользователю в указанном году;
- used_days — количество дней отпуска, уже использованных сотрудником.

Комбинация полей user_id и year образует составной первичный ключ, что исключает дублирование данных.

7.4 Таблица vacation_requests

Таблица vacation_requests используется для хранения заявок на отпуск, создаваемых сотрудниками системы. Каждая запись в таблице соответствует одной заявке на отпуск и содержит всю необходимую информацию для её рассмотрения и согласования.

Данные таблицы используются как для отображения заявок в пользовательском интерфейсе, так и для реализации бизнес-логики согласования отпусков и формирования календаря отсутствий.

Основные поля таблицы:

- id — уникальный идентификатор заявки на отпуск;
- user_id — идентификатор пользователя, подавшего заявку;
- start_date — дата начала отпуска;
- end_date — дата окончания отпуска;
- type — тип отпуска (ежегодный, дополнительный и т.д.);
- status — текущий статус заявки (ожидает согласования, одобрена, отклонена);
- comment — комментарий сотрудника или руководителя к заявке;
- created_at — дата и время создания заявки.

7.5 Таблица request_history

Таблица request_history предназначена для хранения истории всех действий, выполняемых над заявками на отпуск. Наличие данной таблицы обеспечивает прозрачность процесса согласования и позволяет отследить все изменения статусов заявок.

Каждая запись таблицы отражает одно действие, выполненное пользователем системы, и содержит информацию о том, кто и когда выполнил данное действие.

Основные поля таблицы:

- id — уникальный идентификатор записи истории;

- request_id — идентификатор заявки на отпуск, к которой относится действие;
- action — описание выполненного действия (создание, согласование, отклонение);
- comment — комментарий к выполненному действию;
- acted_by — идентификатор пользователя, выполнившего действие;
- acted_at — дата и время выполнения действия.

Использование таблицы request_history позволяет реализовать аудит действий пользователей и при необходимости восстановить полную хронологию работы с заявками.

8. Связи между таблицами и бизнес-логика данных

Связи между таблицами базы данных реализованы на основе внешних ключей и отражают реальные бизнес-процессы, связанные с планированием и согласованием отпусков сотрудников. Выбранная модель данных обеспечивает целостность информации, исключает дублирование данных и позволяет однозначно интерпретировать все действия пользователей в системе.

8.1 Связь пользователей и отделов

Связь между таблицами users и departments реализована посредством поля department_id в таблице пользователей, которое ссылается на первичный ключ id таблицы отделов. Данная связь имеет тип «один ко многим», поскольку одному отделу может принадлежать несколько сотрудников, при этом каждый сотрудник относится только к одному отделу.

Такая модель позволяет:

- формировать календарь отпусков по конкретному отделу;
- контролировать количество сотрудников, находящихся в отпуске одновременно;
- учитывать организационную структуру компании при согласовании заявок.

Ограничение max_simultaneous_vacations, заданное в таблице отделов, используется в бизнес-логике системы для предотвращения ситуации, при которой одновременно отсутствует критически важное количество сотрудников.

8.2 Иерархия управления (руководитель — сотрудник)

Иерархия подчинённости внутри компании реализована через самоссылку в таблице users с использованием поля manager_id, которое ссылается на поле id

той же таблицы. Данная связь также имеет тип «один ко многим», так как один руководитель может управлять несколькими сотрудниками.

Использование самоссылки позволяет:

- определять руководителя конкретного сотрудника;
- ограничивать доступ руководителей только заявками своего отдела;
- реализовать маршрутизацию заявок на согласование без создания отдельной таблицы связей.

8.3 Связь пользователей и баланса отпусков

Таблица vacation_balances связана с таблицей users через поле user_id. Данная связь отражает отношение «один ко многим», поскольку у одного пользователя может существовать несколько записей баланса отпусков, разделённых по календарным годам.

Такое решение позволяет:

- хранить историю отпускных балансов за предыдущие годы;
- корректно рассчитывать доступное количество дней отпуска;
- исключить потерю данных при переходе на новый календарный год.

Первичный ключ, состоящий из пары (user_id, year), гарантирует уникальность записи баланса для каждого пользователя в рамках одного года.

8.4 Связь пользователей и заявок на отпуск

Таблица vacation_requests связана с таблицей users через поле user_id. Связь имеет тип «один ко многим», так как один сотрудник может подать несколько заявок на отпуск в разное время.

Данная связь обеспечивает:

- однозначное определение автора каждой заявки;
- возможность отображения списка заявок конкретного сотрудника;
- формирование отчётов по отпускам за выбранный период.

Статус заявки (pending, approved, rejected) используется для реализации логики согласования и управления жизненным циклом заявки.

8.5 История согласований заявок

Таблица request_history связана сразу с двумя сущностями:

- с таблицей vacation_requests через поле request_id;
- с таблицей users через поле acted_by.

Такая структура позволяет фиксировать все действия, выполненные над заявкой, включая согласование, отклонение и добавление комментариев. Связи имеют тип «один ко многим», поскольку одной заявке может соответствовать несколько записей истории.

Хранение истории согласований обеспечивает:

- прозрачность процесса принятия решений;
- возможность аудита действий пользователей;
- восстановление последовательности изменений статуса заявки

8.6 Общая характеристика модели связей

В рамках текущей модели данных прямые связи типа «многие ко многим» не используются. При необходимости подобные связи могут быть реализованы через дополнительные промежуточные таблицы, что соответствует принципам нормализации реляционных баз данных.

Выбранная структура связей обеспечивает логическую целостность данных, удобство расширения функциональности и устойчивость системы при увеличении количества пользователей и объёма информации.

9. Роли пользователей и сценарии работы системы

В системе планирования отпусков реализована ролевая модель доступа, основанная на разграничении прав пользователей в зависимости от их функциональных обязанностей в компании. Использование ролевой модели позволяет обеспечить безопасность данных, корректное распределение ответственности и соответствие бизнес-процессам организации.

В системе предусмотрены три основные роли пользователей: сотрудник, руководитель и HR-специалист. Каждая роль имеет строго определённый набор доступных функций и сценариев взаимодействия с системой.

9.1 Сценарий работы сотрудника

Сотрудник является основным пользователем системы и инициатором процесса планирования отпуска. Работа сотрудника с системой начинается с авторизации с использованием уникального логина и пароля.

После входа в систему сотруднику становится доступен личный кабинет, в котором отображается информация о текущем балансе отпускных дней. Пользователь может просматривать общее количество доступных дней отпуска, количество уже использованных дней, а также дни, запланированные в будущих заявках.

Для оформления отпуска сотрудник создаёт заявку, указывая дату начала и дату окончания отпуска, а также тип отпуска (ежегодный оплачиваемый или отпуск без сохранения заработной платы). После отправки заявка автоматически регистрируется в системе и получает статус «на рассмотрении».

Сотрудник имеет возможность отслеживать состояние всех своих заявок в режиме реального времени. Изменение статуса заявки (согласование или отклонение) отображается в интерфейсе пользователя вместе с комментарием

руководителя. Дополнительно сотруднику доступен календарь отпусков своего отдела, позволяющий учитывать отсутствие коллег при планировании отпуска.

9.2 Сценарий работы руководителя

Руководитель подразделения отвечает за согласование отпусков сотрудников своего отдела. После авторизации руководителю предоставляется доступ к списку заявок, поданных подчинёнными сотрудниками.

При рассмотрении заявки руководитель получает информацию о периоде отпуска, типе отпуска и текущем балансе отпускных дней сотрудника. Также руководителю доступен календарь отдела, в котором визуализированы уже запланированные отпуска других сотрудников.

На основе полученной информации руководитель принимает решение о согласовании или отклонении заявки. При выполнении любого из действий руководитель обязан указать комментарий, который сохраняется в системе и отображается сотруднику. Все действия руководителя фиксируются в истории заявки.

Дополнительно система автоматически проверяет соблюдение ограничения на максимальное количество сотрудников, одновременно находящихся в отпуске в рамках одного отдела. В случае превышения установленного лимита руководителю отображается предупреждение.

9.3 Сценарий работы HR-специалиста

HR-специалист выполняет функции централизованного управления системой отпусков и обладает расширенными правами доступа. После авторизации HR-специалист получает доступ ко всем заявкам на отпуск по компании, независимо от отдела или руководителя.

В рамках системы HR-специалист управляет балансами отпускных дней сотрудников, включая корректировку общего количества дней отпуска и учёт

использованных дней. Эти данные используются системой при проверке корректности новых заявок.

Кроме того, HR-специалист может формировать отчёты по отпускам за выбранный период. Отчёты включают информацию о сотрудниках, датах отпусков и количестве использованных дней и могут быть экспортированы в формате CSV для дальнейшего использования в бухгалтерских или аналитических системах.

Роль HR-специалиста завершает полный цикл управления процессом планирования отпусков и обеспечивает соответствие работы системы внутренним регламентам и кадровой политике компании.

10. Архитектура системы и структура проекта для разработчиков

Проект реализован на основе клиент-серверной архитектуры с чётким разделением ответственности между компонентами системы. Такой подход упрощает сопровождение проекта, масштабирование функциональности и командную разработку.

Система состоит из нескольких логически обособленных уровней: клиентского уровня, серверного уровня, уровня хранения данных и инфраструктурного уровня. Каждый из уровней выполняет строго определённые функции и взаимодействует с другими компонентами через формализованные интерфейсы.

10.1 Общая архитектура клиент-серверного взаимодействия

Клиентская часть приложения представляет собой веб-интерфейс, доступный пользователям через браузер. Клиент отвечает за отображение данных, ввод пользовательской информации и отправку запросов на сервер.

Серверная часть приложения реализует всю бизнес-логику системы, включая:

- обработку пользовательских запросов;
- проверку прав доступа в зависимости от роли пользователя;
- работу с базой данных;
- реализацию правил согласования отпусков;
- формирование данных для отображения календарей и отчётов.

Взаимодействие между клиентом и сервером осуществляется по протоколу HTTP с использованием REST-подхода. Клиент отправляет запросы к серверным API, а сервер возвращает ответы в структурированном виде.

10.2 Серверная часть (Backend)

Серверное приложение реализовано на языке программирования Python с использованием веб-фреймворка Flask или FastAPI. Выбор фреймворка обусловлен необходимостью быстрого создания API, удобной работы с маршрутами и поддержки масштабируемой архитектуры.

Backend включает следующие основные компоненты:

- слой маршрутов (API), принимающий HTTP-запросы от клиента;
- слой бизнес-логики, реализующий правила работы с заявками, балансами и ролями пользователей;
- слой доступа к данным, отвечающий за взаимодействие с базой данных PostgreSQL.

Разделение серверного кода на логические модули позволяет упростить поддержку проекта и изолировать изменения в отдельных частях системы.

10.3 Работа с базой данных

Взаимодействие с базой данных осуществляется через слой доступа к данным. Все операции создания, чтения, обновления и удаления данных выполняются строго через серверную часть приложения.

База данных PostgreSQL используется в качестве централизованного хранилища информации. Сервер отвечает за:

- выполнение SQL-запросов;
- контроль целостности данных;
- соблюдение ограничений и связей между таблицами;
- обработку транзакций.

Такой подход исключает прямой доступ клиентской части к базе данных и повышает уровень безопасности системы.

10.4 Клиентская часть (Frontend)

Клиентская часть приложения реализована с использованием HTML, CSS и библиотеки Bootstrap для создания адаптивного пользовательского интерфейса. Для отображения календаря отпусков используется библиотека FullCalendar.js.

Frontend отвечает за:

- отображение форм авторизации и подачи заявок;
- визуализацию статусов заявок;
- отображение календарей отпусков сотрудников и отделов;
- взаимодействие с пользователем.

Все данные, отображаемые в интерфейсе, получаются от серверной части приложения через HTTP-запросы.

10.5 Инфраструктура и контейнеризация

Для развёртывания и запуска проекта используется технология контейнеризации Docker. Все компоненты системы (серверное приложение и база данных) запускаются в отдельных контейнерах.

Docker Compose применяется для описания конфигурации проекта и управления запуском всех сервисов одной командой. Использование контейнеризации обеспечивает:

- воспроизводимость окружения;
- простоту развёртывания на различных платформах;
- единый процесс запуска проекта для всех участников команды.

10.6 Структура проекта для разработчиков

Проект организован по модульному принципу, что облегчает навигацию по коду и распределение задач между участниками команды. Структура проекта логически разделена на каталоги, соответствующие функциональным частям системы.

В рамках проекта выделяются:

- каталог серверного приложения с кодом backend;
- файлы конфигурации Docker и Docker Compose;
- скрипты инициализации базы данных и seed-данных;
- файлы клиентского интерфейса;
- документация проекта и README-файл.

Такой подход позволяет новым разработчикам быстро понять устройство системы и приступить к работе с проектом без длительного погружения.

11. Безопасность и управление доступом

В системе планирования отпусков реализованы базовые механизмы безопасности и разграничения доступа, направленные на защиту данных пользователей и предотвращение несанкционированных действий. Архитектура системы предполагает, что все критически важные операции выполняются исключительно на стороне серверного приложения.

Доступ к функционалу системы осуществляется только после прохождения процедуры аутентификации. Пользователь идентифицируется с помощью уникального логина и пароля, после чего сервер определяет его роль и предоставляет доступ только к разрешённым операциям.

Разграничение прав доступа основано на ролевой модели и реализуется на уровне серверной бизнес-логики. Каждому запросу, поступающему от клиента, сопоставляется роль пользователя, и сервер проверяет возможность выполнения запрошенного действия. Такой подход исключает возможность получения доступа к данным или функциям, не предусмотренным для конкретной роли.

Хранение данных осуществляется в централизованной базе данных PostgreSQL, к которой отсутствует прямой доступ со стороны клиентского приложения. Все операции с данными выполняются через серверное API, что повышает уровень защищённости системы.

Дополнительно реализован механизм фиксации действий пользователей при работе с заявками на отпуск. Все изменения статусов заявок, а также действия согласования и отклонения сохраняются в истории, что позволяет обеспечить прозрачность процессов и аудит пользовательской активности.

Выбранный подход к безопасности соответствует требованиям учебного проекта и может быть расширен в дальнейшем за счёт внедрения более

сложных механизмов защиты, таких как шифрование данных, двухфакторная аутентификация и разграничение доступа на уровне API.

12. Развёртывание проекта и процедура запуска системы

Для обеспечения удобного развёртывания, воспроизводимости окружения и упрощения процесса запуска проект реализован с использованием технологий контейнеризации Docker и Docker Compose. Выбранный подход позволяет запускать систему в стандартизированном окружении без необходимости ручной установки и настройки отдельных компонентов, таких как база данных или сервер приложений.

Развёртывание проекта осуществляется на локальной машине разработчика или тестовом сервере при наличии установленного Docker и Docker Compose. Все основные компоненты системы, включая серверное приложение и базу данных PostgreSQL, запускаются в изолированных контейнерах, что исключает конфликты версий программного обеспечения и зависимостей.

Для начала работы с проектом необходимо получить исходный код из репозитория системы контроля версий и перейти в корневой каталог проекта. Запуск системы выполняется одной командой, которая инициирует сборку Docker-образов и старт всех сервисов, описанных в конфигурации Docker Compose. В процессе запуска автоматически создаётся внутренняя сеть Docker, настраивается взаимодействие между контейнерами и производится инициализация базы данных.

При первом запуске системы выполняется создание структуры базы данных, включая все таблицы, связи и ограничения, необходимые для корректной работы приложения. Дополнительно в базу данных загружаются тестовые данные, позволяющие сразу приступить к проверке функциональности системы без необходимости ручного наполнения данных. Это обеспечивает воспроизводимость тестового окружения и упрощает демонстрацию проекта.

После завершения процесса развёртывания серверное приложение становится доступным по заданному сетевому адресу, а пользователи могут получить доступ к веб-интерфейсу системы через браузер. Клиентская часть приложения автоматически взаимодействует с сервером по HTTP-протоколу, обеспечивая получение и отправку данных в рамках реализованной бизнес-логики.

Остановка системы выполняется корректным завершением работы контейнеров, при этом все данные сохраняются в постоянном хранилище базы данных и не теряются при повторном запуске. При необходимости повторного развёртывания или обновления системы достаточно повторно выполнить процедуру запуска, что делает эксплуатацию проекта простой и удобной.

Использование контейнеризации и единой процедуры запуска обеспечивает готовность проекта к демонстрации, тестированию и дальнейшему развитию, а также позволяет использовать разработанное решение в учебных и практических целях без дополнительных затрат на настройку инфраструктуры.

13. Заключение и перспективы развития проекта

В рамках производственной практики был разработан программный продукт, предназначенный для автоматизации процесса планирования и согласования отпусков сотрудников в компании среднего размера. Проект решает актуальную практическую задачу, связанную с отказом от ручного и разрозненного учёта отпусков, и предлагает централизованное, прозрачное и управляемое решение.

Разработанная система охватывает полный жизненный цикл работы с отпусками: от подачи заявки сотрудником до её согласования руководителем и централизованного контроля со стороны HR-специалистов. Использование ролевой модели доступа обеспечивает корректное разграничение прав пользователей и соответствует организационной структуре компании. Архитектура клиент-серверного приложения, реализованная в проекте, позволяет обеспечить надёжность, масштабируемость и удобство сопровождения системы.

В процессе разработки были применены современные технологии и инструменты, включая язык программирования Python, реляционную базу данных PostgreSQL, веб-фреймворки Flask или FastAPI, а также технологии контейнеризации Docker и Docker Compose. Это обеспечивает переносимость проекта, воспроизводимость окружения и удобство командной разработки. Структура базы данных спроектирована с учётом принципов нормализации и отражает реальные бизнес-процессы предметной области.

Проект может быть использован как основа для внедрения в реальную корпоративную среду, а также как демонстрационный пример при обучении технологиям веб-разработки, проектирования баз данных и командной работе над программными продуктами.

В дальнейшем проект может быть расширен и доработан. В качестве перспектив развития можно выделить внедрение автоматического расчёта количества рабочих дней с учётом выходных и праздничных дат, реализацию системы уведомлений о смене статуса заявок, интеграцию с корпоративной электронной почтой или внутренними кадровыми системами, а также расширение аналитических отчётов для HR-отдела. Дополнительно возможно усиление механизмов безопасности, включая расширенное логирование, разграничение доступа на уровне API и внедрение дополнительных методов аутентификации.

Таким образом, разработанный проект является завершённым, логически целостным и функционально обоснованным программным решением, полностью соответствующим поставленным задачам производственной практики и демонстрирующим навыки проектирования, разработки и документирования современных информационных систем.