ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ

«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

**ОТЧЕТ**

по производственной практике

по профессиональному модулю

**ПМ.01. РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ**

специальности 09.02.07 Информационные системы и программирование   
Квалификация Программист

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент гр. | ИС(ПРО)-41 |  |  |  | Горбунов С.И. |
|  |  |  | *подпись* |  | *Фамилия И.О.* |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель практики  от предприятия |  |  |  |  |  |  | Казаков В.В. |
|  | *оценка* |  | *дата* |  | *подпись* |  | *Фамилия И.О.* |
| М.П. |  |  |  |  |  |  |  |
| Руководитель практики  от ГБПОУ РХ ХПК |  |  |  |  |  |  | Отыргашева С.И. |
|  | *оценка* |  | *дата* |  | *подпись* |  | *Фамилия И.О.* |

Абакан 2025 г.

**СОДЕРЖАНИЕ**

[1 Исследование информационной системы предприятия\организации 3](#_Toc195624253)

[2 Анализ предметной области 4](#_Toc195624254)

[3 Проектирование программного модуля 8](#_Toc195624255)

[3.1 Проектирование диаграмм 8](#_Toc195624256)

[3.2 Разработка макетов интерфейса программного модуля 15](#_Toc195624257)

[4 Разработка программного модуля 18](#_Toc195624258)

[4.1 Разработка интерфейсов программного модуля 18](#_Toc195624259)

[4.2 Разработка кода программного модуля 32](#_Toc195624260)

[5 Тестирование программного модуля 42](#_Toc195624261)

[5.1 Разработка тест-кейсов 42](#_Toc195624262)

[5.2 Модульное тестирование 44](#_Toc195624263)

[Приложения 48](#_Toc195624264)

[Спецификация требований 48](#_Toc195624265)

[Репозиторий с документами в системе контроля версий 51](#_Toc195624266)

# 1 Исследование информационной системы предприятия\организации

Данный раздел посвящен всестороннему анализу информационной системы, функционирующей в ГБУЗ РХ «РКБ имени Г.Я. Ремишевской». Основной акцент в исследовании сделан на изучении процессов, связанных с учетом рабочего времени сотрудников и составлением графика их дежурств. В современных условиях медицинские учреждения, сталкиваются с необходимостью эффективного управления персоналом, особенно в контексте круглосуточной работы и высокой нагрузки на сотрудников. Это требует наличия надежной и функциональной информационной системы, способной минимизировать временные затраты на административные задачи и исключить возможные ошибки в планировании.

На данный момент в ГБУЗ РХ РКБ «имени Г.Я. Ремишевской» используется онлайн-сервис Platimun представленный на рисунке 1.

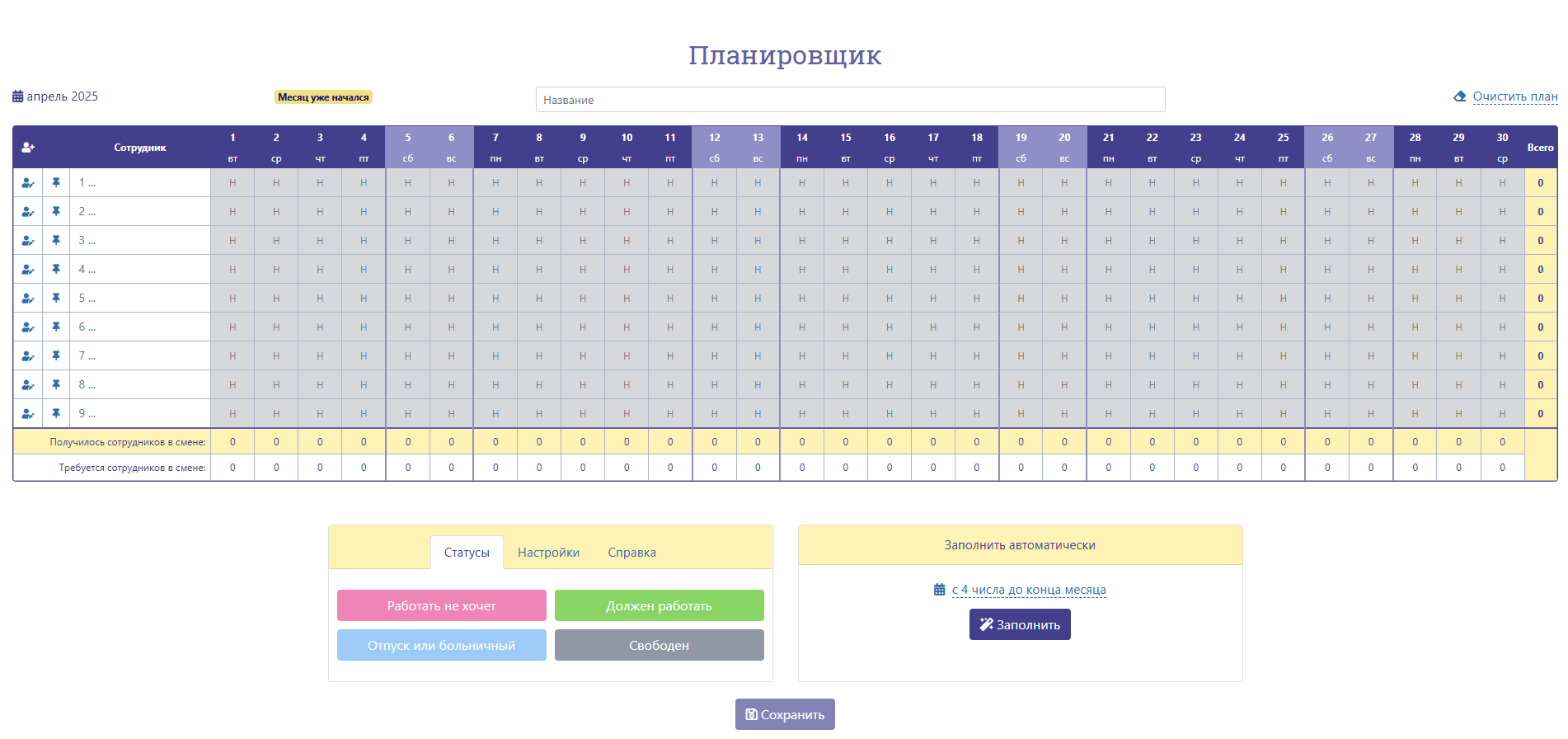


Рисунок 1 – Platinum

Характеристики компьютеров, которые используются на данный момент в ГБУЗ РХ РКБ «имени Г.Я. Ремишевской»:

Операционная система: Windows 10

Процессор: AMD A6-9500E

Видеокарта: AMD Ryzen 3 PRO 2100GE AM4

Оперативная память: DDR3 ECC REG 16 ГБ, 16GB 1600MHz X 2pcs

Жесткий диск: Kingston SSD M.2 2280 NV2 250 Гб PCI-E 4.0 x4 NVMe SNV2S/250G

# 2 анализ предметной области

Предметная область охватывает процессы управления персоналом в медицинском учреждении, с акцентом на планирование и учет графика дежурств сотрудников. ГБУЗ РХ РКБ «имени Г.Я. Ремишевской» является многопрофильной больницей, предоставляющей медицинскую помощь населению региона в круглосуточном режиме. Это подразумевает наличие большого штата сотрудников, включая врачей, средний и младший медицинский персонал, а также административных работников, чья деятельность должна быть скоординирована для обеспечения непрерывности лечебного процесса. Основной задачей в данной предметной области является организация рабочего времени персонала таким образом, чтобы обеспечить оптимальное распределение нагрузки, соблюдение трудового законодательства и оперативное реагирование на изменения в расписании.

Учет графика дежурств представляет собой сложный процесс, который требует учета множества факторов. К ним относятся: количество сотрудников в штате, их специализация и квалификация, продолжительность смен, необходимость соблюдения норм отдыха между сменами, учет отпусков, больничных листов и других непредвиденных обстоятельств. В условиях медицинского учреждения этот процесс усложняется высокой ответственностью за здоровье пациентов, что делает недопустимыми ошибки или сбои в планировании.

Основные участники и их роли.

В предметной области можно выделить следующих ключевых участников:

* Административный персонал (ответственные за составление графиков): обычно это сотрудники отдела кадров или старшие медицинские сестры, которые занимаются планированием дежурств. Их задачи включают сбор данных о доступности сотрудников, распределение смен, учет пожеланий персонала и контроль соблюдения нормативов.
* Медицинский персонал: Врачи, медсестры и санитары, чьи рабочие графики составляются. Они являются конечными пользователями системы, так как от правильности расписания зависит их рабочая нагрузка и возможность отдыха.
* Руководство учреждения: Главный врач, заместители и заведующие отделениями, которые утверждают графики и следят за их соответствием потребностям больницы.

Текущие процессы и их особенности.

* Сбор данных: информация о сотрудниках (их доступности, отпусках, больничных) собирается ответственным лицом (глав. врач, отдел кадров).
* Планирование: на основе собранных данных составляется предварительный график, учитывающий потребности отделений и нормы рабочего времени.
* Согласование: график передается на утверждение руководству и корректируется при необходимости.
* Информирование: сотрудники получают доступ к утвержденному расписанию (например, через распечатанные копии или устное уведомление).

Однако такой подход имеет ряд недостатков:

* Высокая трудоемкость: ручное составление графиков занимает значительное время, особенно при большом штате.
* Риск ошибок: возможны пересечения смен, несоблюдение норм отдыха или пропуск важных данных.
* Низкая гибкость: внесение изменений (например, из-за внезапного выхода сотрудника на больничный) требует переработки всего графика.
* Ограниченная доступность: сотрудники не всегда могут оперативно узнать о своем расписании.

Нормативно-правовая база.

Предметная область регулируется рядом законодательных актов Российской Федерации:

* Трудовой кодекс РФ: устанавливает нормы рабочего времени (не более 40 часов в неделю для обычного графика, особые условия для сменной работы), минимальные периоды отдыха (от 30 минут до двух часов).
* Федеральный закон № 152-ФЗ "О персональных данных": требует обеспечения безопасности информации о сотрудниках при ее обработке в информационной системе.
* Приказы Минздрава РФ: определяют специфику работы медицинских учреждений.

Технологический контекст.

В современных условиях многие организации переходят на автоматизированные системы управления персоналом, которые включают модули для учета рабочего времени и планирования смен. Однако в ГБУЗ РХ «РКБ имени Г.Я. Ремишевской», как и во многих бюджетных учреждениях, внедрение таких решений может быть ограничено из-за недостатка финансирования, устаревшей технической базы или отсутствия квалифицированных специалистов для их поддержки. Это создает потребность в разработке простого, но эффективного программного модуля, который можно интегрировать с существующими инструментами или использовать автономно.

Основные функции и особенности Planimum.

* Автоматическое составление расписаний: автоматически распределяет рабочие смены и выходные дни между сотрудниками, стремясь к равномерной нагрузке. Алгоритм учитывает заданные условия, такие как минимальное и максимальное количество часов, предпочтения сотрудников и ограничения по сменам.
* Учет предыдущих планов: сервис предоставляет возможность учитывать данные из предыдущих графиков, чтобы избежать перегрузки или недогрузки сотрудников на границе месяцев. Например, можно проверить, не достаются ли "тяжелые" смены одним и тем же сотрудникам регулярно.
* Редактирование и версии планов: пользователи могут вручную корректировать автоматически созданные расписания, а Planimum подсчитывает суммарные часы работы для контроля.
* Гибкость отображения: в редакторе отображаются дни недели, что упрощает ориентацию в расписании.
* Учет часов и смен: пользователь может задавать количество часов в каждой смене, а сервис автоматически подсчитывает общее количество часов за месяц для каждого сотрудника. Эта информация выводится в отдельной колонке.

Преимущества онлайн-сервиса Planimum.

* Экономия времени: автоматизация процесса составления расписаний значительно сокращает время, необходимое для планирования, особенно в организациях с большим штатом.
* Гибкость: учет пожеланий сотрудников и возможность ручной корректировки делают сервис универсальным для разных типов организаций.
* Прозрачность: возможность делиться планами через секретные ссылки упрощает коммуникацию между руководителями и сотрудниками.
* Контроль нагрузки: подсчет часов и учет предыдущих планов помогают избежать переработок и равномерно распределить нагрузку.

Недостатки и ограничения.

* Отсутствие уведомлений: на основе доступной информации Planimum не предоставляет функции автоматического оповещения сотрудников о назначенных сменах, что может быть неудобно для оперативного информирования.
* Ограниченная аналитика: сервис не предлагает глубокую аналитику, например, автоматическое выявление переработок (свыше 160 часов в месяц) или рекомендации по оптимизации графика.
* Интерактивность: нет упоминаний о возможности редактирования расписания через перетаскивание, что могло бы улучшить пользовательский опыт.
* Доступность информации: на сайте Planimum отсутствуют подробные данные о ценовой политике, условиях подписки или наличии бесплатной версии, что может затруднить принятие решения о покупке.

Анализ предметной области показывает, что учет графика дежурств в ГБУЗ РХ РКБ «имени Г.Я. Ремишевской» представляет собой сложный процесс, требующий оптимизации. Текущие методы недостаточно эффективны и не отвечают современным требованиям к автоматизации. Разработка специализированного программного модуля позволит устранить существующие проблемы, повысить производительность труда административного персонала и улучшить условия работы медицинских сотрудников. Предметная область обладает четкими границами и спецификой, что делает задачу разработки реализуемой и востребованной в контексте деятельности учреждения.

# проектирование программного модуля

## **3.1 Проектирование диаграмм**

Проектируемая система предназначена для автоматизации учета и управления графиком дежурств в больнице. Основные функциональные задачи, которые решает система:

Учет и управление персоналом:

* Регистрация нового медицинского персонала при их приеме на работу.
* Ведение информации о сотрудниках: ФИО, специальности, квалификации.

Планирование дежурств:

* Создание графика дежурств для врачей, медсестер и другого персонала.
* Учет смен.

Медицинский учет:

* Фиксация данных о дежурствах.

Управление отделениями:

* Планирование распределения персонала по отделениям в зависимости от потребностей.

Диаграмма Вариантов Использования — это диаграмма, на которой изображаются варианты использования проектируемой системы, заключённые в границу системы, и внешние акторы, а также определённые отношения между акторами и вариантами использования.

Диаграмма Вариантов Использования представлена на рисунке 2.

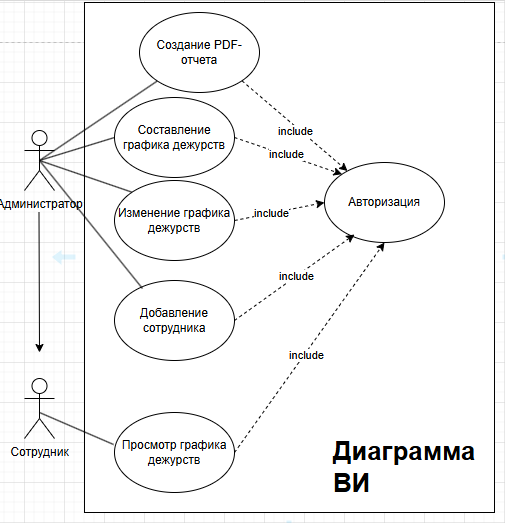


Рисунок 2 – Диаграмма Вариантов Использования

Диаграмма Вариантов Использования описывает взаимодействие администратора и сотрудника с системой. Сотрудник проходит авторизацию в систему и после чего может просматривать график дежурств. Администратора может делать все тоже самое, что и сотрудник, но еще может составлять график дежурств для сотрудников, изменять график дежурств, добавлять новых сотрудников, создавать PDF-отчеты.

Диаграмма классов (англ. class diagram) — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними.

Атрибуты класса «Смена для дежурств»:

* id: int — уникальный идентификатор сметы.
* id\_сотрудника: int — идентификатор сотрудника, связанного с этой сменой. Связан с классом Сотрудник через атрибут id\_сотрудника.
* Дата дежурства: date — дата проведения лекции.

Атрибуты класса «Сотрудник»:

* id: int — уникальный идентификатор сотрудника.
* Фамилия: varchar — фамилия сотрудника.
* Имя: varchar — имя сотрудника.
* Отчество: varchar — отчество сотрудника.
* Должность: int — идентификатор должности сотрудника. Связан с классом Должность(id).
* Количество дежурств: int — количество дежурств, которые отработал сотрудник.
* Отделение: int — идентификатор отделения, к которому относится сотрудник. Связан с классом Отделение(id).

Атрибуты класса «Должности»:

* id: int — уникальный идентификатор должности.
* Название: varchar — название должности.

Атрибуты класса «Отделение»:

* id: int — уникальный идентификатор отделения.
* Название: varchar — название отделения.
* Корпус: int — идентификатор корпуса, к которому относится отделение. Связан с классом Корпус(id).

Атрибуты класса «Корпус»:

* id: int — уникальный идентификатор корпуса.
* Название: varchar — название корпуса.

Связи между классами

Сотрудник – Смена для дежурств:

* Связь: 0..\* (много) к 1 (один).
* Тип: Ассоциация через внешний ключ сотрудник\_id.
* Описание: У сотрудника может быть несколько смен за месяц, но в определенный день сотрудник может быть только 1 раз.

Сотрудник - Должность:

* Связь: 0..\* (много) к 1 (один).
* Тип: Ассоциация через внешний ключ должность.
* Описание: У сотрудника есть одна должность, но одна должности может быть несколько сотрудников.

Сотрудник - Отделение:

* Связь: 0..\* (много) к 1 (один).
* Тип: Ассоциация через внешний ключ отделение.
* Описание: Сотрудник принадлежит к одному отделению, но в отделении может быть много сотрудников.

Отделение - Корпус:

* Связь: 0..\* (много) к 1 (один).
* Тип: Ассоциация через внешний ключ корпус.
* Описание: Отделение находится в одном корпусе, но в корпусе может быть много отделений.

Диаграмма классов представлена на рисунке 3.

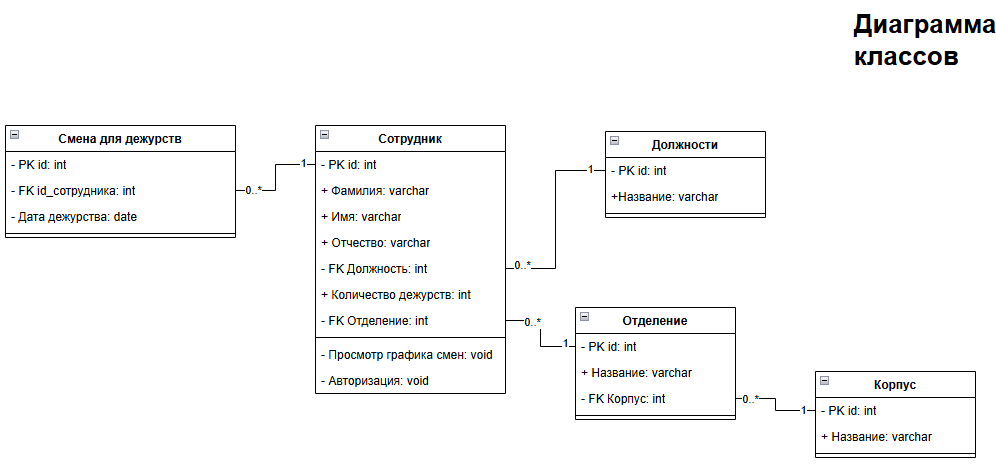


Рисунок 3 – Диаграмма классов

Диаграмма IDEF0 — это методология функционального моделирования и графическая нотация, предназначенная для формализации и описания бизнес-процессов.

Диаграмма IDEF0 представлена на рисунке 4.

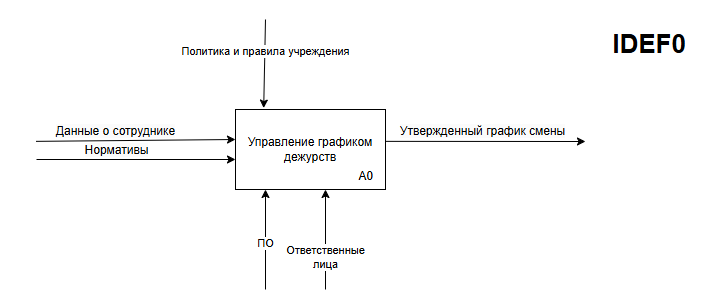


Рисунок 4 – Диаграмма IDEF0

Описание диаграммы:

Центральный блок:

* В центре находится прямоугольник с надписью "Управление графиком дежурств". Это основной процесс.

Входы:

* Слева к центральному блоку подходят три стрелки, обозначающие входные данные или ресурсы, необходимые для выполнения процесса:
* Данные о сотруднике (ФИО, должность, отделение).
* Нормативы (нормативные документы, стандарты или правила).
* Предпочтения сотрудников (учет пожеланий сотрудников).

Управление:

* Сверху к блоку подходит стрелка с надписью "Политики и правила учреждения". Это означает, что процесс управления регулируется определенными политиками и правилами, установленными в учреждении.

Механизмы:

* Снизу к блоку подходит стрелка с надписью "Ответственные лица". Это указывает на то, что для выполнения процесса привлекаются определенные лица или роли, ответственные за оценку (например, эксперты, сотрудники учреждения).
* Снизу к блоку подходит стрелка с надписью "ПО". Это указывает на то, что используется определенное ПО.

Выход:

* Справа от блока выходит стрелка с надписью "Утвержденный график смены". Это результат процесса — утвержденный график, который показывает, когда и какой сотрудник выходит на дежурство.

Диаграмма IDEF0 первого уровня - используется для моделирования информационных систем и данных, а не процессов.

Диаграмма IDEF0 первого уровня представлена на рисунке 5.

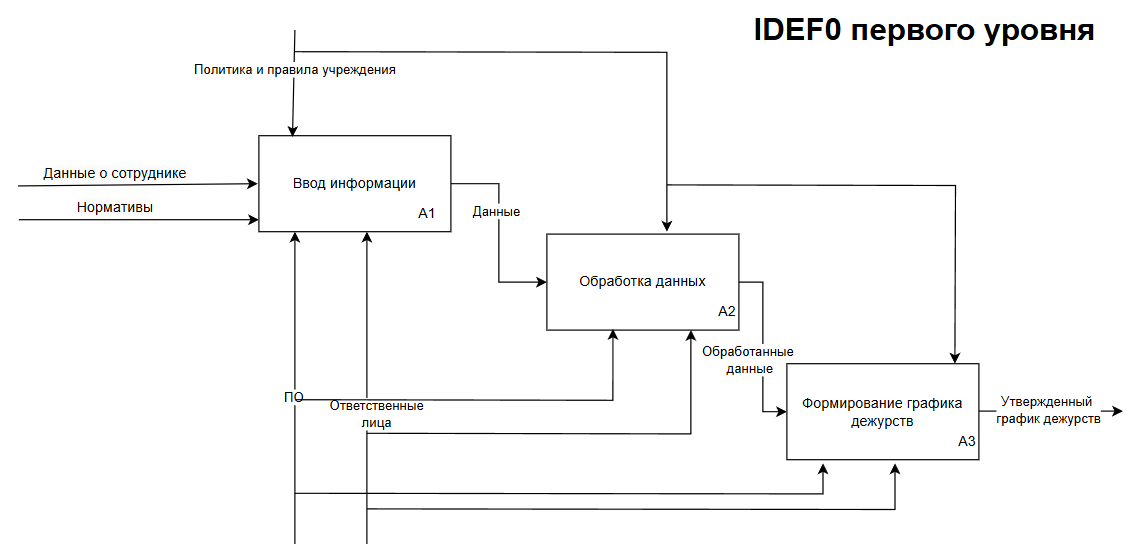


Рисунок 5 – Диаграмма IDEF0 первого уровня

Описание диаграммы:

Блоки:

* Первый блок: "Составление графиков".
* Второй блок: "Анализ и отчетность".

Входы и выходы:

* Вход в блок "Составление графиков":
  + Стрелка с надписью "Данные о сотруднике". Это входные данные, которые используются для составления графиков (ФИО сотрудника, должность и отделение).
* Выход из блока "Составление графиков" и вход в блок "Анализ и отчетность":
  + Стрелка между блоками с надписью "Утвержденные графики". Это означает, что результатом работы первого блока являются утвержденные графики, которые затем передаются во второй блок для анализа и отчетности.
* Выход из блока "Анализ и отчетность":
  + Стрелка с надписью "Отчет". Это конечный результат процесса — отчет, который содержит результаты анализа графиков.

Управление:

* Для блока "Составление графиков": "Политика и правила учреждения" указывает, что процесс составления графиков регулируется политиками и правилами учреждения.

Для блока "Анализ и отчетность": также "Политика и правила учреждения". Анализ и отчетность также подчиняются этим правилам.

ER-диаграмма — это графическое представление структуры базы данных, которое используется для моделирования данных на концептуальном уровне. Она помогает разработчикам и аналитикам визуализировать, как данные организованы, какие сущности (объекты) существуют в системе, какие у них есть атрибуты (свойства), и как эти сущности связаны между собой. ER-диаграммы часто применяются на этапе проектирования баз данных, чтобы создать логическую модель перед реализацией в виде таблиц в реляционной базе данных.

ER-диаграмма представлена на рисунке 6.

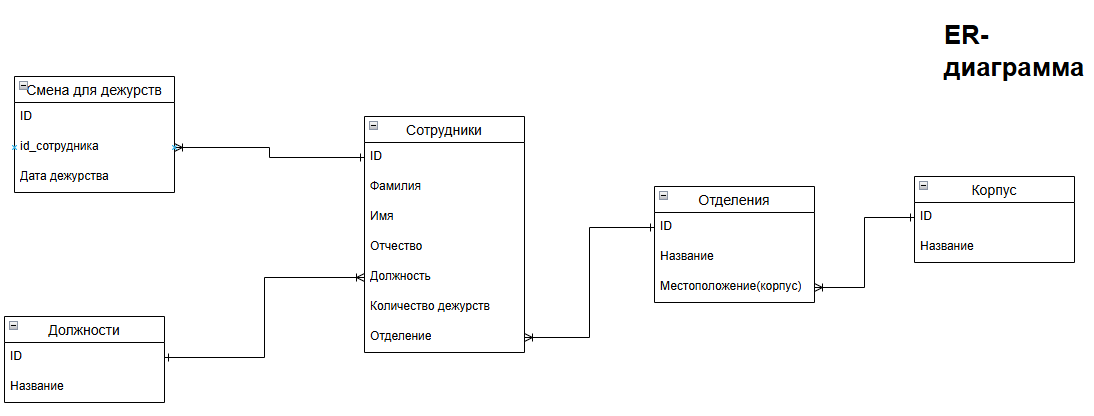


Рисунок 6 – ER-диаграмма

Сущность Сотрудник состоит из следующих атрибутов:

* ID: Уникальный идентификатор сотрудника (первичный ключ).
* Фамилия: Фамилия сотрудника.
* Имя: Имя сотрудника.
* Отчество: Отчество сотрудника.
* Должность: Название должности сотрудника.
* Количество дежурств: Число дежурств, которые отработал сотрудник.
* Отделение: Отделение, в котором работает сотрудник.

Сущность Должность состоит из следующих атрибутов:

* ID: Уникальный идентификатор должности (первичный ключ).
* Название: Название должности.

Сущность Отделение состоит из следующих атрибутов:

* ID: Уникальный идентификатор отделения (первичный ключ).
* Название: Название отделения.
* Местоположение (корпус): Местоположение отделения. Внешний ключ, связывающий отделение с корпусом.

Сущность Корпус состоит из следующих атрибутов:

* ID: Уникальный идентификатор корпуса (первичный ключ).
* Название: Название корпуса.

Связи между сущностями:

* Сотрудник → Должность: Связь "один ко многим". Каждый сотрудник занимает одну должность, но одна должность может быть у нескольких сотрудников. Это указано через атрибут "Должность" в сущности "Сотрудник", который является внешним ключом, ссылающимся на ID должности.
* Сотрудник → Отделение: Связь "один ко многим". Каждый сотрудник работает в одном отделении, но в одном отделении может работать несколько сотрудников. Это указано через атрибут "Отделение" в сущности "Сотрудник", который ссылается на ID отделения.
* Отделение → Корпус: Связь "один ко многим". Каждое отделение находится в одном корпусе, но в одном корпусе может быть несколько отделений. Связь реализована через атрибут "Местоположение (корпус)" в сущности "Отделение", который является внешним ключом, ссылающимся на ID корпуса.

## **3.2 Разработка макетов интерфейса программного модуля**

Приложение для управления дежурствами в медицинском учреждении должно быть интуитивно понятным, функциональным и визуально приятным, с акцентом на удобство для администраторов и сотрудников. Основные принципы дизайна и структуры:

Общая структура:

* Главное окно с вкладками: «График дежурств», «Сотрудники» (для администраторов).
* Кнопка «Выйти из учетной записи» в правом нижнем углу.

Вкладка «График дежурств»:

* Календарь в виде таблицы: дни недели (Пн–Вс) в столбцах, недели в строках.
* Навигация: кнопки «Предыдущий месяц» и «Следующий месяц», метка текущего месяца и года в центре.
* Ячейки с количеством дежурных, цветовое выделение: занятые дни — светло-желтый, текущий день — серый, пустые — белый.
* Кнопка «Сохранить в PDF-отчет» внизу (для администраторов).

Представлено на рисунке 7.



Рисунок 7 – Главное окно

Вкладка «Сотрудники»:

* Список сотрудников с данными: ФИО, должность, отделение, количество смен.
* Кнопки: «Обновить список», «Добавить сотрудника» (для администраторов).

Представлено на рисунке 8.

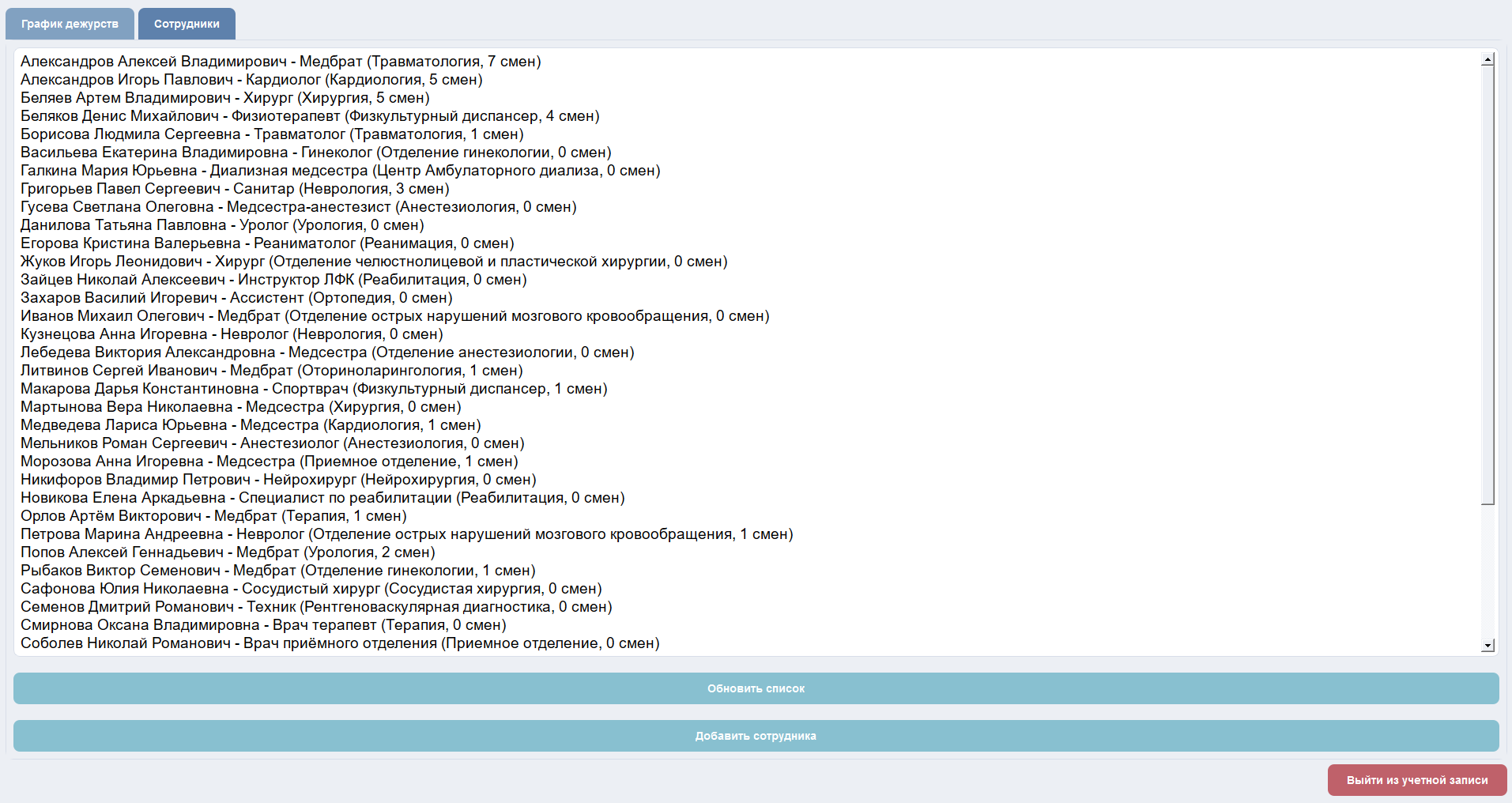


Рисунок 8 – Вкладка «Сотрудники»

Дизайн:

* Цветовая палитра: светлый фон (#ECEFF4), белые элементы с серыми границами (#D8DEE9), кнопки в синих (#88C0D0) и красных (#BF616A) тонах.
* Скругленные углы, шрифт Arial 14px, жирный для заголовков.
* Компактные ячейки с отступами для читаемости.

Функциональность:

* Клик по дню в календаре открывает окно редактирования дежурных.
* Уведомления о новых сменах и переработках (свыше 160 часов).
* Адаптивность: элементы подстраиваются под размер окна.

# 4 разработка программного модуля

## **4.1 Разработка интерфейсов программного модуля**

PySide6 — это библиотека Python для создания графических пользовательских интерфейсов (GUI), являющаяся привязкой к фреймворку Qt. Она предоставляет мощные инструменты для разработки интерактивных приложений с визуальными элементами, такими как окна, кнопки, таблицы, списки и диалоговые окна.

Описание листинга 1:

* Методы: setCentralWidget(), addWidget(), addLayout(), setStyleSheet() — настройка интерфейса; connect() — привязка сигналов к слотам (prev\_month, next\_month, show\_shift\_details, load\_schedule, generate\_pdf, show\_employee\_card\_from\_list, load\_employees, show\_add\_employee\_dialog, logout); setSizePolicy(), setSectionResizeMode() — настройка масштабирования.
* Ветвления: Отсутствуют (метод выполняет последовательную сборку интерфейса без условий).
* Циклы: Отсутствуют.
* Поток: Создает центральный виджет, далее инициализирует вкладки («График дежурств», «Сотрудники»), потом добавляет элементы (таблицу, список, кнопки, метки), затем задает стили и поведение, и завершает сборку интерфейса.

Показано на рисунке 9.



Рисунок 9 – Главное окно

Листинг 1 – Главное окно:

def init\_ui(self):

        central\_widget = QWidget()

        self.setCentralWidget(central\_widget)

        main\_layout = QVBoxLayout(central\_widget)

        central\_widget.setStyleSheet("background-color: #ECEFF4;")

        self.tab\_widget = QTabWidget()

        self.tab\_widget.setStyleSheet("""

            QTabWidget::pane {

                border: 1px solid #D8DEE9;

                background-color: #ECEFF4;

                border-radius: 10px;

            }

            QTabBar::tab {

                background-color: #81A1C1;

                color: white;

                padding: 12px 20px;

                margin-right: 5px;

                border-top-left-radius: 8px;

                border-top-right-radius: 8px;

                font: bold 14px Arial;

            }

Продолжение листинга 1.

            QTabBar::tab:selected {

                background-color: #5E81AC;

            }

        """)

        self.schedule\_widget = QWidget()

        schedule\_layout = QVBoxLayout(self.schedule\_widget)

        nav\_layout = QHBoxLayout()

        self.prev\_btn = QPushButton("Предыдущий месяц")

        self.prev\_btn.setStyleSheet("""

            QPushButton {

                background-color: #88C0D0;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #A3BE8C;

            }

        """)

        self.prev\_btn.clicked.connect(self.prev\_month)

        self.month\_label = QLabel()

        self.month\_label.setAlignment(Qt.AlignCenter)

        self.month\_label.setFont(QFont("Arial", 16, QFont.Bold))

        self.month\_label.setStyleSheet("color: #2E3440;")

        self.next\_btn = QPushButton("Следующий месяц")

        self.next\_btn.setStyleSheet("""

            QPushButton {

                background-color: #88C0D0;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #A3BE8C;

            }

        """)

        self.next\_btn.clicked.connect(self.next\_month)

        nav\_layout.addWidget(self.prev\_btn)

        nav\_layout.addWidget(self.month\_label)

        nav\_layout.addWidget(self.next\_btn)

        self.table = QTableWidget()

        self.table.setStyleSheet("""

            QTableWidget {

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                gridline-color: #D8DEE9;

            }

            QHeaderView::section {

Продолжение листинга 1.

                background-color: #5E81AC;

                color: white;

                padding: 10px;

                border: 1px solid #D8DEE9;

                font: bold 14px Arial;

            }

        """)

        self.table.cellClicked.connect(self.show\_shift\_details)

self.table.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOff)

self.table.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff    self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)

        self.table.verticalHeader().setVisible(False)

        self.table.setShowGrid(True)

        self.table.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        button\_layout = QHBoxLayout()

        self.refresh\_btn = QPushButton("Обновить график")

        self.refresh\_btn.setObjectName("refresh\_schedule\_btn")

        self.refresh\_btn.setStyleSheet("""

            QPushButton {

                background-color: #88C0D0;

                color: white;

                padding: 12px 24px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #A3BE8C;

            }

        """)

        self.refresh\_btn.clicked.connect(self.load\_schedule)

        self.pdf\_btn = QPushButton("Сохранить в PDF")

        self.pdf\_btn.setObjectName("save\_pdf\_btn")

        self.pdf\_btn.setStyleSheet("""

            QPushButton {

                background-color: #D08770;

                color: white;

                padding: 12px 24px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #EBCB8B;

            }

        """)

        self.pdf\_btn.clicked.connect(self.generate\_pdf)

        button\_layout.addWidget(self.refresh\_btn)

        button\_layout.addWidget(self.pdf\_btn)

        schedule\_layout.addLayout(nav\_layout)

        schedule\_layout.addWidget(self.table)

        schedule\_layout.addLayout(button\_layout)

Продолжение листинга 1.

        schedule\_layout.setSpacing(20)

        self.employees\_widget = QWidget()

        employees\_layout = QVBoxLayout(self.employees\_widget)

        self.employees\_list = QListWidget()

        self.employees\_list.setStyleSheet("""

            QListWidget {

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                padding: 5px;

            }

            QListWidget::item:selected {

                background-color: #5E81AC;

                color: white;

            }

        """)      self.employees\_list.itemClicked.connect(self.show\_employee\_card\_from\_list)

        self.refresh\_employees\_btn = QPushButton("Обновить список")      self.refresh\_employees\_btn.setObjectName("refresh\_employees\_btn")

        self.refresh\_employees\_btn.setStyleSheet("""

            QPushButton {

                background-color: #88C0D0;

                color: white;

                padding: 12px 24px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #A3BE8C;

            }

        """)      self.refresh\_employees\_btn.clicked.connect(self.load\_employees)

        self.add\_employee\_btn = QPushButton("Добавить сотрудника")

        self.add\_employee\_btn.setObjectName("add\_employee\_btn")

        self.add\_employee\_btn.setStyleSheet("""

            QPushButton {

                background-color: #88C0D0;

                color: white;

                padding: 12px 24px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #A3BE8C;

            }

        """)      self.add\_employee\_btn.clicked.connect(self.show\_add\_employee\_dialog)

Продолжение листинга 1.

        employees\_layout.addWidget(self.employees\_list)

        employees\_layout.addWidget(self.refresh\_employees\_btn)

        employees\_layout.addWidget(self.add\_employee\_btn)

        employees\_layout.setSpacing(20)

   self.tab\_widget.addTab(self.schedule\_widget, "График дежурств")

        self.tab\_widget.addTab(self.employees\_widget, "Сотрудники")

        self.logout\_btn = QPushButton("Выйти из учетной записи")

        self.logout\_btn.setStyleSheet("""

            QPushButton {

                background-color: #BF616A;

                color: white;

                padding: 12px 24px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #D08770;

            }

        """)

        self.logout\_btn.clicked.connect(self.logout)

        main\_layout.addWidget(self.tab\_widget)

        main\_layout.addWidget(self.logout\_btn, alignment=Qt.AlignRight)

        main\_layout.setContentsMargins(10, 10, 10, 10)

Описание листинга 2:

* Методы: super().\_\_init\_\_() — инициализация родительского класса QDialog; setWindowTitle(), setStyleSheet(), setFixedSize() — настройка окна; update\_employee\_list(), load\_available\_employees() — заполнение списка и комбобокса; setLayout(), addWidget(), addLayout() — сборка интерфейса; connect() — привязка сигналов к слотам (show\_employee\_card, add\_employee\_to\_shift, remove\_employee\_from\_shift, close).
* Ветвления: if self.parent\_window and self.parent\_window.is\_admin — проверка прав администратора для отображения элементов управления; тернарные выражения — получение репозиториев (employee\_repo, schedule\_repo).
* Циклы: Отсутствуют.
* Поток: Инициализирует диалоговое окно далее задает заголовок и стили, далее сохраняет входные данные, затем создает список дежурных, комбобокс и кнопки, после настраивает видимость элементов в зависимости от прав и собирает интерфейс в вертикальном макете.

Показано на рисунке 10.

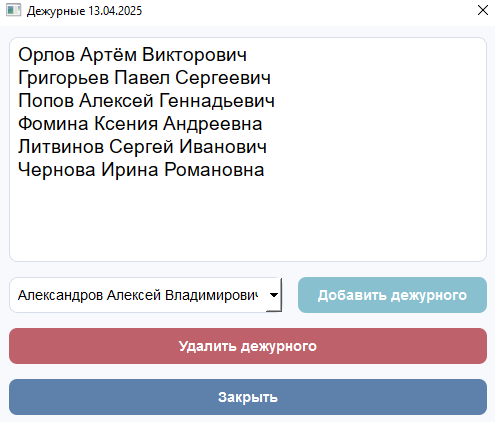


Рисунок 10 – Расписание дежурных сотрудников

Листинг 2 – Расписание дежурных сотрудников:

def \_\_init\_\_(self, shift\_date, employees, parent\_window=None):

        super().\_\_init\_\_(parent\_window)

        self.setWindowTitle(f"Дежурные {shift\_date.strftime('%d.%m.%Y')}")

        self.setStyleSheet("background-color: #F7F9FC; border-radius: 10px;")

        self.setFixedSize(500, 400)

        self.shift\_date = shift\_date

        self.employees = employees

        self.employee\_repo = parent\_window.employee\_repo if parent\_window else None

        self.schedule\_repo = parent\_window.schedule\_repo if parent\_window else None

        self.parent\_window = parent\_window

        layout = QVBoxLayout()

        self.list\_widget = QListWidget()

        self.list\_widget.setStyleSheet("""

            QListWidget {

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                padding: 5px;

            }

            QListWidget::item:selected {

Продолжение листинга 2.

                background-color: #5E81AC;

                color: white;

            }

        """)

        self.update\_employee\_list()

self.list\_widget.itemClicked.connect(self.show\_employee\_card)

        self.add\_layout = QHBoxLayout()

        self.employee\_combo = QComboBox()

        self.employee\_combo.setStyleSheet("""

            QComboBox {

                padding: 8px;

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                font: 14px Arial;

            }

        """)

        self.load\_available\_employees()

        self.add\_btn = QPushButton("Добавить дежурного")

        self.add\_btn.setStyleSheet("""

            QPushButton {

                background-color: #88C0D0;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #A3BE8C;

            }

        """)

        self.add\_btn.clicked.connect(self.add\_employee\_to\_shift)

        self.add\_layout.addWidget(self.employee\_combo)

        self.add\_layout.addWidget(self.add\_btn)

        self.remove\_btn = QPushButton("Удалить дежурного")

        self.remove\_btn.setStyleSheet("""

            QPushButton {

                background-color: #BF616A;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #D08770;

            }

        """)      self.remove\_btn.clicked.connect(self.remove\_employee\_from\_shift)

        close\_btn = QPushButton("Закрыть")

        close\_btn.setStyleSheet("""

            QPushButton {

Продолжение листинга 2.

                background-color: #5E81AC;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #81A1C1;

            }

        """)

        close\_btn.clicked.connect(self.close)

        layout.addWidget(self.list\_widget)

        if self.parent\_window and self.parent\_window.is\_admin:

            layout.addLayout(self.add\_layout)

            layout.addWidget(self.remove\_btn)

        else:

            self.employee\_combo.setEnabled(False)

            self.add\_btn.setEnabled(False)

            self.remove\_btn.setEnabled(False)

            self.employee\_combo.setVisible(False)

            self.add\_btn.setVisible(False)

            self.remove\_btn.setVisible(False)

        layout.addWidget(close\_btn)

        layout.setSpacing(15)

        self.setLayout(layout)

Описание листинга 3:

* Методы: super().\_\_init\_\_() — инициализация родительского класса QDialog; setWindowTitle(), setFixedSize(), setStyleSheet() — настройка диалогового окна; addRow(), addItems(), setLayout() — сборка формы; setPlaceholderText() — задание подсказок; connect() — привязка сигналов к слотам (add\_employee, reject).
* Ветвления: Отсутствуют (метод выполняет последовательную настройку интерфейса).
* Циклы: Используется неявный цикл (otd.name for otd in otdeleniya) для заполнения комбобокса.
* Поток: Инициализирует диалоговое окно, далее сохраняет репозиторий сотрудников, затем задает заголовок, размер и стили, после создает поля ввода (ФИО, должность), комбобокс отделений и кнопки, собирает элементы в форме и устанавливает макет.

Представлено на рисунке 11.

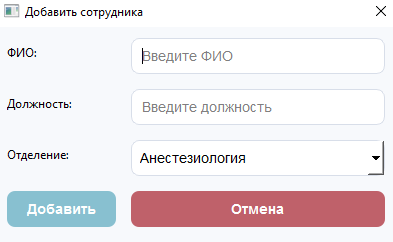


Рисунок 11 – Окно добавление сотрудника

Листинг 3 – Окно добавления сотрудника:

def \_\_init\_\_(self, employee\_repo, parent=None):

        super().\_\_init\_\_(parent)

        self.employee\_repo = employee\_repo

        self.setWindowTitle("Добавить сотрудника")

        self.setFixedSize(400, 300)

        self.setStyleSheet("background-color: #F7F9FC; border-radius: 10px;")

        layout = QFormLayout()

        self.name\_input = QLineEdit()

        self.name\_input.setPlaceholderText("Введите ФИО")

        self.name\_input.setStyleSheet("""

            QLineEdit {

                padding: 8px;

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                font: 14px Arial;

            }

        """)

        self.position\_input = QLineEdit()

        self.position\_input.setPlaceholderText("Введите должность")

        self.position\_input.setStyleSheet("""

            QLineEdit {

                padding: 8px;

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                font: 14px Arial;

            }

        """)

        self.otdelenie\_combo = QComboBox()

        otdeleniya = self.employee\_repo.get\_all\_otdeleniya()

        self.otdelenie\_combo.addItems([otd.name for otd in otdeleniya])

        self.otdelenie\_combo.setStyleSheet("""

Продолжение листинга 3.

            QComboBox {

                padding: 8px;

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                font: 14px Arial;

            }

        """)

        add\_btn = QPushButton("Добавить")

        add\_btn.setStyleSheet("""

            QPushButton {

                background-color: #88C0D0;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #A3BE8C;

            }

        """)

        add\_btn.clicked.connect(self.add\_employee)

        cancel\_btn = QPushButton("Отмена")

        cancel\_btn.setStyleSheet("""

            QPushButton {

                background-color: #BF616A;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #D08770;

            }

        """)

        cancel\_btn.clicked.connect(self.reject)

        layout.addRow("ФИО:", self.name\_input)

        layout.addRow("Должность:", self.position\_input)

        layout.addRow("Отделение:", self.otdelenie\_combo)

        layout.addRow(add\_btn, cancel\_btn)

        layout.setSpacing(15)

        self.setLayout(layout)

Описание листинга 4:

* Методы: super().\_\_init\_\_() — инициализация родительского класса QDialog; setWindowTitle(), setStyleSheet() — настройка диалогового окна; setFont() — установка шрифтов; addRow(), addWidget(), setLayout() — сборка формы; connect() — привязка сигнала кнопки к слоту close.
* Ветвления: Отсутствуют (метод последовательно создает интерфейс).
* Циклы: Отсутствуют.
* Поток: Инициализирует диалоговое окно, задает заголовок и стили, создает метки для отображения данных сотрудника (ФИО, должность, отделение, количество смен), настраивает шрифты, добавляет кнопку закрытия, собирает элементы в форме, устанавливает макет.

Представлено на рисунке 12.

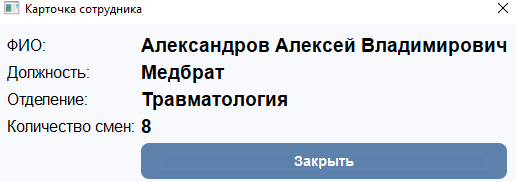


Рисунок 12 – Окно «Карточка сотрудника»

Листинг 4 – Окно «Карточка сотрудника»:

def \_\_init\_\_(self, employee\_data, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Карточка сотрудника")

        self.setStyleSheet("background-color: #F7F9FC; border-radius: 10px;")

        layout = QFormLayout()

        label\_font = QFont("Arial", 12)

        value\_font = QFont("Arial", 14, QFont.Bold)

        name\_label = QLabel("ФИО:")

        name\_label.setFont(label\_font)

        name\_value = QLabel(employee\_data['name'])

        name\_value.setFont(value\_font)

        pos\_label = QLabel("Должность:")

        pos\_label.setFont(label\_font)

        pos\_value = QLabel(employee\_data['position'])

        pos\_value.setFont(value\_font)

        otd\_label = QLabel("Отделение:")

        otd\_label.setFont(label\_font)

        otd\_value = QLabel(employee\_data['otdelenie'])

        otd\_value.setFont(value\_font)

        shifts\_label = QLabel("Количество смен:")

        shifts\_label.setFont(label\_font)

        shifts\_value = QLabel(str(employee\_data['shifts']))

        shifts\_value.setFont(value\_font)

        layout.addRow(name\_label, name\_value)

        layout.addRow(pos\_label, pos\_value)

        layout.addRow(otd\_label, otd\_value)

        layout.addRow(shifts\_label, shifts\_value)

Продолжение листинга 4.

        close\_btn = QPushButton("Закрыть")

        close\_btn.setStyleSheet("""

            QPushButton {

                background-color: #5E81AC;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #81A1C1;

            }

        """)

        close\_btn.clicked.connect(self.close)

        layout.addWidget(close\_btn)

        self.setLayout(layout)

Описание листинга 5:

* Методы: super().\_\_init\_\_() — инициализация родительского класса QDialog; setWindowTitle(), setFixedSize(), setStyleSheet() — настройка диалогового окна; setPlaceholderText(), setEchoMode() — конфигурация полей ввода; addRow(), addWidget(), setLayout() — сборка формы; connect() — привязка сигнала кнопки к слоту check\_credentials.
* Ветвления: Отсутствуют (метод выполняет последовательную настройку интерфейса).
* Циклы: Отсутствуют.
* Поток: Инициализирует диалоговое окно, далее задает заголовок, размер и стили, затем создает поля для ввода логина и пароля, кнопку входа, потом задает начальные значения флагов (is\_admin, employee\_id) и собирает элементы в форме, устанавливает макет.

Представлено на рисунке 13.

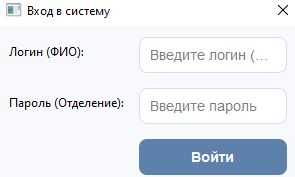


Рисунок 13 – Окно авторизации

Листинг 5 – Окно авторизации:

def \_\_init\_\_(self, parent=None):

        super().\_\_init\_\_(parent)

        self.setWindowTitle("Вход в систему")

        self.setFixedSize(300, 200)

        self.setStyleSheet("background-color: #F7F9FC; border-radius: 10px;")

        layout = QFormLayout()

        self.login\_input = QLineEdit()

        self.login\_input.setPlaceholderText("Введите логин (ФИО)")

        self.login\_input.setStyleSheet("""

            QLineEdit {

                padding: 8px;

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                font: 14px Arial;

            }

        """)

        self.password\_input = QLineEdit()

        self.password\_input.setEchoMode(QLineEdit.Password)

        self.password\_input.setPlaceholderText("Введите пароль")

        self.password\_input.setStyleSheet("""

            QLineEdit {

                padding: 8px;

                border: 1px solid #D8DEE9;

                border-radius: 8px;

                background-color: white;

                font: 14px Arial;

            }

        """)

        login\_btn = QPushButton("Войти")

        login\_btn.setStyleSheet("""

            QPushButton {

                background-color: #5E81AC;

                color: white;

                padding: 10px 20px;

                border-radius: 8px;

                font: bold 14px Arial;

            }

            QPushButton:hover {

                background-color: #81A1C1;

            }

        """)

        login\_btn.clicked.connect(self.check\_credentials)

        layout.addRow("Логин (ФИО):", self.login\_input)

        layout.addRow("Пароль (Отделение):", self.password\_input)

        layout.addWidget(login\_btn)

        layout.setSpacing(15)

        self.setLayout(layout)

Продолжение листинга 5.

        self.is\_admin = False

        self.employee\_id = None

## **4.2 Разработка кода программного модуля**

Описание листинга 6:

* Методы: text(), strip(), currentText() — получение данных; QMessageBox.warning(), QMessageBox.information() — сообщения; accept() — закрытие диалога; employee\_repo.add\_employee — добавление в БД.
* Ветвления: if — проверка пустых полей; if-else — обработка результата добавления.
* Циклы: Отсутствуют.
* Поток: Получает данные → проверяет → добавляет → показывает результат (успех/ошибка).

Листинг 6 – Добавление сотрудника:

def add\_employee(self):

name = self.name\_input.text().strip(

position = self.position\_input.text().strip()

otdelenie\_name = self.otdelenie\_combo.currentText()

if not name or not position or not otdelenie\_name:

QMessageBox.warning(self, "Ошибка", "Все поля должны быть заполнены")

return

if self.employee\_repo.add\_employee(name, position, otdelenie\_name):

QMessageBox.information(self, "Успех", "Сотрудник успешно добавлен")

self.accept()

else:

QMessageBox.warning(self, "Ошибка", "Не удалось добавить сотрудника")

Рисунок 14 показывает диалоговое окно «Добавить сотрудника», созданное с использованием PySide6. Окно содержит: поле ввода «ФИО» (для имени сотрудника), поле ввода «Должность» (для должности), выпадающий список «Отделение» (с примером «Кардиология»), кнопки «Добавить» (голубая) и «Отмена» (красная).  
Окно предназначено для добавления нового сотрудника в систему учета дежурств.

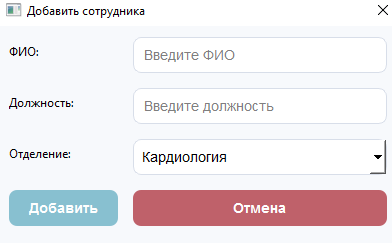


Рисунок 14 – Добавление сотрудника

Описание листинга 7:

* Методы: currentData() — получает ID сотрудника; date() — получает дату; add\_shift(), get\_shifts\_by\_date() — добавляет и получает смены; update\_employee\_list(), load\_available\_employees(), load\_schedule() — обновляют списки и график.
* Ветвления: if — проверяет наличие данных и окна.
* Циклы: Отсутствуют.
* Поток: Проверяет данные, добавляет смену, обновляет списки и график.

Листинг 7 – Добавление сотрудника в график дежурства:

def add\_employee\_to\_shift(self):

if not self.employee\_combo or not self.schedule\_repo:

return

emp\_id = self.employee\_combo.currentData()

if not emp\_id:

return

self.schedule\_repo.add\_shift(emp\_id, self.shift\_date.date())

self.employees = self.schedule\_repo.get\_shifts\_by\_date(self.shift\_date.date())

self.update\_employee\_list()

self.load\_available\_employees()

if self.parent\_window:

self.parent\_window.load\_schedule()

Рисунок 15 показывает диалоговое окно, созданное с использованием PySide6, для управления дежурствами. В окне есть:

* Выпадающий список (например, для выбора сотрудника, с примером "Васильев Василий Васильевич").
* Кнопка "Добавить дежурного" (голубая) для добавления сотрудника в график.
* Кнопка "Удалить дежурного" (красная) для удаления сотрудника из графика.
* Кнопка "Закрыть" (синяя) для закрытия окна.  
  Окно предназначено для управления графиком дежурств на указанную дату.

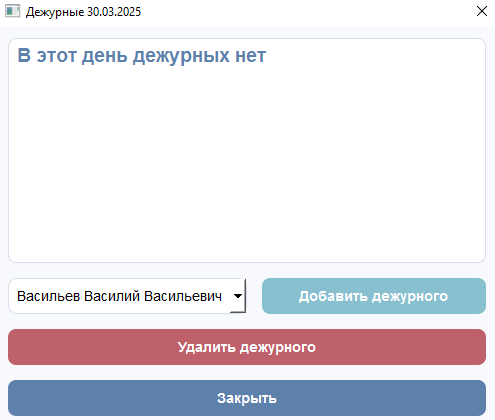


Рисунок 15 – Управление графиком

Описание листинга 8:

* Методы: replace() — задает начало месяца; getSaveFileName() — выбор файла; get\_month\_schedule() — получает данные; SimpleDocTemplate(), Paragraph(), Image() — формируют PDF; build() — сохраняет PDF.
* Ветвления: if — проверяет индекс месяца, наличие файла, логотипа, сотрудников.
* Циклы: for — перебирает даты и сотрудников для отчета.
* Поток: Формирует заголовок, собирает данные, добавляет стили, логотип, текст, сохраняет PDF.

Листинг 8 – Генерация отчета:

def generate\_pdf(self):

start\_of\_month = self.current\_date.replace(day=1)

months = ["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь",

"Июль", "Август", "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"]

month\_index = start\_of\_month.month - 1

if not (0 <= month\_index < len(months)):

month\_name = "Неизвестный месяц"

else:

month\_name = months[month\_index

Продолжение листинга 8.

print(f"month\_name перед Paragraph: {month\_name}")

title\_text = f"График дежурств: {month\_name} {start\_of\_month.year}"

print(f"title\_text: {title\_text}")

filename, \_ = QFileDialog.getSaveFileName(self, "Сохранить PDF", f"График\_дежурств\_{month\_name}\_{start\_of\_month.year}.pdf",

"PDF Files (\*.pdf)")

if not filename:

return

schedule\_data = self.schedule\_repo.get\_month\_schedule(start\_of\_month)

doc = SimpleDocTemplate(filename, pagesize=A4)

elements = []

styles = getSampleStyleSheet()

font\_name = 'DejaVuSans' if pdfmetrics.getFont('DejaVuSans') else 'Helvetica'

print(f"Используемый шрифт: {font\_name}")

logo\_style = ParagraphStyle(

name='LogoStyle',

parent=styles['Normal'],

fontName=font\_name,

fontSize=14,

textColor=colors.black,

alignment=1

)

title\_style = ParagraphStyle(

name='TitleStyle',

parent=styles['Heading1'],

fontName=font\_name,

fontSize=24,

textColor=colors.blue,

spaceAfter=12,

alignment=1

)

info\_style = ParagraphStyle(

name='InfoStyle',

parent=styles['Normal'],

fontName=font\_name,

fontSize=12,

leading=14,

alignment=1

)

schedule\_style = ParagraphStyle(

name='ScheduleStyle',

parent=styles['Normal'],

fontName=font\_name,

fontSize=12,

leading=16,

leftIndent=20,

spaceAfter=6

Продолжение листинга 8.

)

footer\_style = ParagraphStyle(

name='FooterStyle',

parent=styles['Normal'],

fontName=font\_name,

fontSize=10,

leading=12,

alignment=1

)

logo\_path = os.path.join(os.getcwd(), 'logo.jpg')

if os.path.exists(logo\_path):

logo = Image(logo\_path, width=250, height=250)

elements.append(logo)

else:

logo\_text = Paragraph("Логотип больницы", logo\_style)

elements.append(logo\_text)

title = Paragraph(title\_text, title\_style)

elements.append(title)

hospital\_info = Paragraph(

"Республиканская клиническая больница имени Г.Я. Ремишевской<br/>Адрес: пр. Ленина д. 23<br/>Телефон: +7 390 224-82-54",

info\_style

)

elements.append(hospital\_info)

elements.append(Paragraph("<br/>", styles['Normal']))

schedule\_text = "Список дежурств:<br/>\n"

for i, (date, employees) in enumerate(sorted(schedule\_data.items()), 1):

schedule\_text += f"<b>{i}.</b> Дежурства на {date}:<br/>\n"

if employees:

schedule\_text += " - <i>Дежурные:</i><br/>\n"

for j, employee in enumerate(employees, 1):

schedule\_text += f" <b>{j}.</b> {employee}<br/>\n"

else:

schedule\_text += " - <i>Дежурных нет</i><br/>\n"

schedule\_text += "<br/>\n"

schedule\_paragraph = Paragraph(schedule\_text, schedule\_style)

elements.append(schedule\_paragraph)

footer\_text = (

f"Сформировано: {datetime.now().strftime('%d.%m.%Y %H:%M')}<br/>"

"Отдел кадров: 8 (3902) 248-263<br/>"

"Электронная почта: kadry@gospital.ru"

)

footer = Paragraph(footer\_text, footer\_style)

elements.append(footer)

Продолжение листинга 8.

doc.build(elements)

print(f"PDF сохранен как {filename}")

Рисунок 16 показывает PDF-отчет «График дежурств: Апрель 2025» для Республиканской клинической больницы им. Г.Я. Ремишевской. В отчете есть:

* Логотип РКБ в верхней части.
* Текстовое поле: информация о больнице: адрес (пр. Ленина, д. 23), телефон (+7 390 224-82-54).
* Список дежурств.
* Подвал: дата формирования (19.03.2025 14:34:44), контакт отдела кадров (8 (3902) 248-263), email ([kadry@gospital.ru](mailto:kadry@gospital.ru)).



Рисунок 16 – Создание отчета

Описание листинга 9:

* Методы: warning() — показывает ошибки; selectedItems() — получает выбранный элемент; data() — извлекает ID смены; remove\_shift(), get\_shifts\_by\_date() — удаляет и получает смены; update\_employee\_list(), load\_available\_employees(), load\_schedule() — обновляют списки и график.
* Ветвления: if — проверяет окно, права, выбор сотрудника, результат удаления.
* Циклы: Отсутствуют.
* Поток: Проверяет условия → удаляет смену → обновляет данные или показывает ошибку.

Листинг 9 – Удаление сотрудника из дежурства:

def remove\_employee\_from\_shift(self):

if not self.parent\_window:

QMessageBox.warning(self, "Ошибка", "Нет доступа к родительскому окну")

return

selected\_items = self.list\_widget.selectedItems()

if not selected\_items or not self.parent\_window.is\_admin:

QMessageBox.warning(self, "Ошибка", "Выберите сотрудника для удаления или нет прав администратора")

return

shift\_id, \_ = selected\_items[0].data(Qt.UserRole)

if self.schedule\_repo and self.schedule\_repo.remove\_shift(shift\_id):

self.employees = self.schedule\_repo.get\_shifts\_by\_date(self.shift\_date.date())

self.update\_employee\_list()

self.load\_available\_employees()

if self.parent\_window:

self.parent\_window.load\_schedule()

else:

QMessageBox.warning(self, "Ошибка", "Не удалось удалить дежурство")

Рисунок 17 диалогового окна PySide6 для управления дежурствами включает:

* Список сотрудников: «Иванов Иван Иванович», «Кузнецов Олег Викторович», «Козлов Мария Андреевна».
* Выпадающий список (например, «Васильев Василий Васильевич»).
* Кнопки: «Добавить дежурного» (голубая), «Удалить дежурного» (красная), «Закрыть» (синяя).

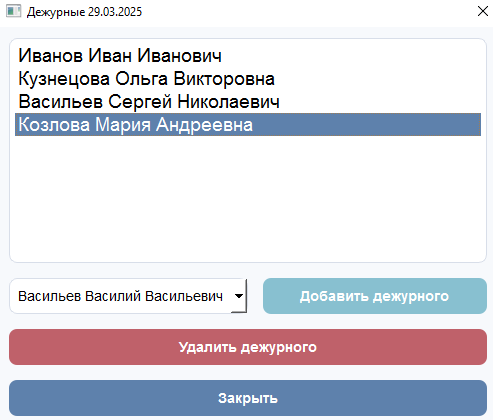


Рисунок 17 – Удаление сотрудника из дежурства

Описание листинга 10:

* Методы: replace() — задает даты; weekday(), days — вычисляет дни; setText(), setRowCount(), setColumnCount(), clearContents() — настраивает таблицу; get\_shifts\_by\_date() — получает смены; setItem(), setBackground() — заполняет ячейки.
* Ветвления: if-else — определяет следующий месяц; if-elif-else — задает цвет ячеек.
* Циклы: for — перебирает дни и настраивает размеры таблицы.
* Поток: Вычисляет даты, настраивает таблицу, заполняет ячейки данными и цветами.

Листинг 10 – Отображение графика дежурств:

def load\_schedule(self):

start\_of\_month = self.current\_date.replace(day=1)

if start\_of\_month.month < 12:

next\_month = start\_of\_month.replace(month=start\_of\_month.month + 1, year=start\_of\_month.year)

else:

next\_month = start\_of\_month.replace(month=1, year=start\_of\_month.year + 1)

days\_in\_month = (next\_month - start\_of\_month).days

months = ["Январь", "Февраль", "Март", "Апрель", "Май", "Июнь",

"Июль", "Август", "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"]

Продолжение листинга 10.

self.month\_label.setText(f"{months[start\_of\_month.month - 1]} {start\_of\_month.year}")

first\_day\_weekday = start\_of\_month.weekday()

total\_slots = days\_in\_month + first\_day\_weekday

weeks = (total\_slots + 6) // 7

self.table.setRowCount(weeks)

self.table.setColumnCount(7)

self.table.clearContents()

today = datetime.now()

for day in range(days\_in\_month):

current\_date = start\_of\_month + timedelta(days=day)

total\_position = day + first\_day\_weekday

row = total\_position // 7

col = total\_position % 7

employees = self.schedule\_repo.get\_shifts\_by\_date(current\_date.date())

item = QTableWidgetItem()

employee\_count = len(employees) if employees else 0

item.setText(f"{day + 1}\n{employee\_count} чел.") # Отображение номера дня и количества дежурных

item.setData(Qt.UserRole, {

'date': current\_date,

'employees': employees if employees else []

})

item.setTextAlignment(Qt.AlignCenter)

if current\_date.date() == today.date():

item.setBackground(QColor("#E5E9F0"))

elif employee\_count > 0:

item.setBackground(QColor("#EBE9D8"))

else:

item.setBackground(QColor("#FFFFFF"))

self.table.setItem(row, col, item)

for col in range(7):

self.table.setColumnWidth(col, 125)

for row in range(weeks):

self.table.setRowHeight(row, 80)

Рисунок 18 показывает календарь дежурств за март 2025 года в приложении. Включает:

* Заголовок: «Март 2025».
* Кнопки: "Предыдущий месяц" и «Следующий месяц».
* Таблица с днями недели (Пн–Вс) и числами месяца.
* Ячейки: показывают день и количество дежурных (например, «1\n1 чел.», «4\n2 чел.», «0 чел.» — нет дежурных).
* Цвета: текущий день (21 марта) выделен голубым, дни с дежурными — желтым, без дежурных — белым, выбранный день (29 марта) — синим.



Рисунок 18 – Отображение графика дежурств

# 5 тестирование программного модуля

## **5.1 Разработка тест-кейсов**

Тест-кейс - документ или описание, которое определяет конкретный сценарий тестирования программного обеспечения. Он включает набор условий, шагов, входных данных и ожидаемых результатов, которые используются для проверки того, что система или её часть работает корректно и соответствует требованиям.

Название: Неуспешный вход с неверным паролем

ID: 1

Предусловие: В базе данных есть сотрудник "Иванов Иван Иванович" в отделении "Терапия".

Шаги:

* Ввести логин "Иванов Иван Иванович".
* Ввести пароль "Хирургия".
* Нажать кнопку "Войти".

Ожидаемый результат: Появляется сообщение "Неверный пароль", поле пароля очищается.

Фактический результат: Появляется сообщение "Неверный пароль", поле пароля очищается.

Название: Добавление нового сотрудника

ID: 2

Предусловие: Пользователь вошел как администратор, в базе есть отделение "Терапия".

Шаги:

* Перейти на вкладку "Сотрудники".
* Нажать кнопку "Добавить сотрудника".
* Ввести ФИО "Петров Петр Петрович", должность "Врач", выбрать отделение "Терапия".
* Нажать "Добавить".

Ожидаемый результат: Появляется сообщение "Сотрудник успешно добавлен", сотрудник отображается в списке.

Фактический результат: Появляется сообщение "Сотрудник успешно добавлен", сотрудник отображается в списке.

Название: Добавление дежурства

ID: 3

Предусловие: Пользователь вошел как администратор, есть сотрудник "Иванов Иван Иванович".

Шаги:

* На вкладке "График дежурств" кликнуть на ячейку с датой "20.03.2025".
* В открывшемся окне нажать на выпадающий список сотрудников, выбрать "Иванов Иван Иванович".
* Нажать "Добавить дежурного".

Ожидаемый результат: "Иванов Иван Иванович" появляется в списке дежурных на 20.03.2025, график обновляется.

Фактический результат: "Иванов Иван Иванович" появляется в списке дежурных на 20.03.2025, график обновляется.

Название: Удаление дежурства

ID: 4

Предусловие: Пользователь вошел как администратор, на 20.03.2025 назначен "Иванов Иван Иванович".

Шаги:

* Кликнуть на ячейку с датой "20.03.2025".
* Выбрать "Иванов Иван Иванович" в списке.
* Нажать "Удалить дежурного".

Ожидаемый результат: "Иванов Иван Иванович" удален из списка дежурных, график обновляется.

Фактический результат: "Иванов Иван Иванович" удален из списка дежурных, график обновляется.

Название: Просмотр карточки сотрудника

ID: 5

Предусловие: Пользователь вошел в систему, есть сотрудник "Иванов Иван Иванович".

Шаги:

* На вкладке "Сотрудники" выбрать "Иванов Иван Иванович".
* Нажать на него.

Ожидаемый результат: Открывается карточка сотрудника с данными: ФИО, должность, отделение, количество смен.

Фактический результат: Открывается карточка сотрудника с данными: ФИО, должность, отделение, количество смен.

## **5.2 Модульное тестирование**

Модульное тестирование является важной частью разработки программного обеспечения, позволяя разработчикам проверять корректность работы отдельных компонентов программы изолированно от остальной системы. В языке Python для проведения модульного тестирования широко используется библиотека unittest, которая предоставляет мощный и гибкий инструментарий для создания, выполнения и анализа тестов. В данной главе рассмотрен процесс проведения модульного тестирования с использованием unittest на примере системы авторизации, реализованной в приложении для управления графиком дежурств больницы.

В рамках тестирования проверяется логика авторизации, которая проверяет учетные данные пользователя (логин и пароль). Логин представляет собой ФИО сотрудника, а пароль – название отделения, в котором он работает.

Для тестирования был создан изолированный класс LoginTester, продемонстрированный в листинге 11, который воспроизводит логику авторизации без использования GUI, и написаны тесты для проверки двух сценариев: успешного входа и входа с неверным паролем. Результаты тестов будут отображаться в диалоговом окне с помощью QMessageBox, наблюдать результат выполнения можно на рисунке 19.

Листинг кода 11 – Модульное тестирование 1:

def check\_credentials(self, login, password, mock\_db, mock\_msgbox):

        if login == "admin" and password == "admin":

            self.is\_admin = True

            self.accept()

            return

        mock\_employee = Employee(id=1, name="Иванов Иван Иванович")

        mock\_otdelenie = Otdelenie(name="Хирургия")

        mock\_db.return\_value.session.query.return\_value.filter.return\_value.first.return\_value = (

            mock\_employee, mock\_otdelenie) if login == "Иванов Иван Иванович" else None

        if login == "Иванов Иван Иванович" and mock\_otdelenie.name == password:

            self.employee\_id = mock\_employee.id

            self.accept()

        else:

Продолжение листинга 11.

            mock\_msgbox.warning(None, "Ошибка", "Неверный пароль")

            self.password\_input.clear()

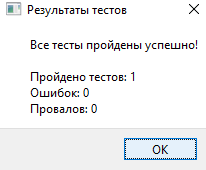


Рисунок 19 – Результат выполнения модульного тестирования 1

Данное модульное тестирование проверяет функциональность удаления дежурства из графика в приложении управления дежурствами больницы. Тест-кейс (ID: 4) имитирует действия администратора, удаляющего сотрудника "Иванов Иван Иванович" из графика на 20.03.2025. Тест изолирует логику метода remove\_employee\_from\_shift, проверяя, что сотрудник удаляется из списка дежурных, график обновляется, и ошибки не возникают. Результаты отображаются в диалоговом окне. Полный код модульного тестирования представлен в листинге 12. Результаты выполнения теста, демонстрирующие успешное прохождение или возможные ошибки, показаны на Рисунке 20.

Листинг кода 12 – Модульное тестирование 2:

 def remove\_employee\_from\_shift(self, selected\_shift\_id, selected\_emp\_id):

        if not self.parent\_window or not self.parent\_window.is\_admin:

            return False

        selected\_items = [Mock()]

        selected\_items[0].data.return\_value = (selected\_shift\_id, selected\_emp\_id)

        self.list\_widget.selectedItems.return\_value = selected\_items

        shift\_id, \_ = selected\_items[0].data()

        if self.parent\_window.schedule\_repo.remove\_shift(shift\_id):

            self.employees = self.parent\_window.schedule\_repo.get\_shifts\_by\_date(self.shift\_date)

            self.update\_employee\_list()

            self.parent\_window.load\_schedule()

            return True

        return False

Продолжение листинга 12.

        return employee\_data

Листинг 13 – Модульное тестирование 3:

 def setUp(self):

        self.shift\_date = datetime.strptime("20.03.2025", "%d.%m.%Y")

        self.employee\_id, self.shift\_id = 1, 100

        self.employees = [(self.shift\_id, self.employee\_id, "Иванов Иван Иванович")]

        self.parent\_window = MainWindow()

        self.tester = ShiftTester(self.shift\_date, self.employees, self.parent\_window)

    def test\_remove\_employee\_from\_shift(self):

    self.parent\_window.schedule\_repo.remove\_shift.return\_value = True       self.parent\_window.schedule\_repo.get\_shifts\_by\_date.return\_value = []       self.assertTrue(self.tester.remove\_employee\_from\_shift(self.shift\_id, self.employee\_id))

        self.assertEqual(self.tester.employees, [])

        self.tester.list\_widget.clear.assert\_called\_once()

    self.tester.parent\_window.load\_schedule.assert\_called\_once()

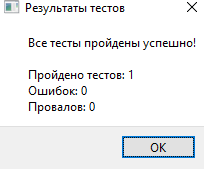


Рисунок 20 – Результат модульного тестирования 2

Модульное тестирование разработано для проверки функциональности просмотра карточки сотрудника в приложении управления дежурствами больницы. Тест-кейс (ID: 5) моделирует сценарий, в котором пользователь, войдя в систему, выбирает сотрудника «Иванов Иван Иванович» на вкладке «Сотрудники» и открывает его карточку. Логика тестирования реализована в классе EmployeeCardTester, который изолирует метод show\_employee\_card от GUI. Тест проверяет, что при выборе сотрудника возвращаются корректные данные: ФИО, должность, отделение и количество смен (5), а также что эти данные передаются для отображения в интерфейсе.

Листинг 14 – Модульное тестирование 3:

def show\_employee\_card(self, employee\_id):

        employee = Employee(id=employee\_id, name="Иванов Иван Иванович", position="Хирург")

        employee\_data = {

            "name": employee.name,

            "position": employee.position,

            "otdelenie": "Хирургия",

            "shifts\_count": 5

        }

        self.dialog.name\_label.setText(employee\_data["name"]       self.dialog.position\_label.setText(employee\_data["position"]       self.dialog.otdelenie\_label.setText(employee\_data["otdelenie"]  self.dialog.shifts\_label.setText(str(employee\_data["shifts\_count"]))

        return employee\_data

Листинг 15 – Модульное тестирование 3:

def test\_show\_employee\_card(self):

        result = self.tester.show\_employee\_card(1)

        expected = {"name": "Иванов Иван Иванович", "position": "Хирург", "otdelenie": "Хирургия", "shifts\_count": 5}

        self.assertEqual(result, expected)

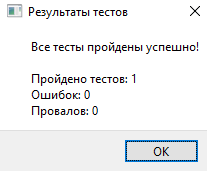


Рисунок 21 – Результат выполнения модульного тестирования 3

# приложения

## **Спецификация требований**

Приложение представляет собой настольное приложение для управления графиком дежурств сотрудников больницы. Оно разработано с использованием Python, PySide6 (Qt для интерфейса), SQLAlchemy (для работы с базой данных PostgreSQL) и ReportLab (для генерации PDF-отчетов).

Системные характеристики:

* ОС: Windows/Linux
* Библиотеки: PySide6, SQLAlchemy, ReportLab
* База данных: PostgreSQL
* Минимальные требования к оборудованию: Процессор 2 ГГц, 4 ГБ ОЗУ, 500 МБ свободного места на диске.
* Сеть: Поддержка работы в локальной сети для многопользовательского доступа (в текущей версии реализована только локальная работа, но архитектура позволяет расширение до клиент-серверной модели).

Пользовательские требования - это описание функций и возможностей системы, которые необходимы пользователям для выполнения их задач. Они отражают ожидания пользователей от программы, включая удобство интерфейса, доступные действия и производительность.

Пользовательские требования:

* Вход по логину и паролю.
* Просмотр карточки сотрудника.
* Отображение графика дежурств в виде таблицы по месяцам.
* Навигация между месяцами.
* Просмотр дежурных на конкретный день.
* Сохранение графика за месяц в PDF-отчете.
* Удобный и понятный интерфейс.

Бизнес-правила - это ограничения, условия и политики, которые определяют, как должна работать система в соответствии с требованиями бизнеса или организации. Они описывают логику работы приложения, учитывая специфику предметной области.

Бизнес-правила:

* Аутентификация: логин сотрудника — его ФИО, пароль — название отделения, к которому он привязан; администратор имеет полный доступ ко всем функциям; обычный сотрудник может только просматривать данные.
* Управление сотрудниками: добавление сотрудника возможно только при наличии существующего отделения в базе; у сотрудника обязательно должны быть указаны ФИО, должность и отделение.
* график дежурств: один сотрудник не может быть назначен на дежурство в один и тот же день дважды; график отображается по месяцам, начиная с первого числа; дежурства хранятся в базе с привязкой к дате и сотруднику.
* Экспорт в PDF: PDF-файл включает заголовок ("График дежурств за [месяц]"), логотип больницы (если загружен), информацию о больнице (название, адрес), список дежурств (дата, ФИО сотрудника, отделение), футер (дата формирования отчета); даты в PDF сортируются в хронологическом порядке.
* Ограничения: удаление дежурства возможно только администратором; нельзя добавить дежурство на прошедшую дату (не реализовано, но может быть добавлено в дальнейшем).

Детальные спецификации:

Детальные спецификации будут включать модули для учета сотрудников, управления графиком дежурств, генерации отчетов и графический интерфейс для пользователей.

* Модуль учета сотрудников: Хранение данных о сотрудниках (ФИО, должность, отделение), управление через класс EmployeeRepository, отображение в виде списка в интерфейсе.
* Модуль управления графиком дежурств: Хранение и обработка данных о дежурствах через ScheduleRepository, отображение в виде таблицы с навигацией по месяцам. Поддержка добавления/удаления дежурств администратором.
* Модуль отчетов: Генерация PDF-файлов с использованием ReportLab, включающих логотип (если доступен), заголовок, информацию о больнице и список дежурств за месяц.

Графический интерфейс: Разработан на PySide6 с использованием классов QMainWindow, QTableWidget, QDialog и др. Основные элементы: вкладки ("График дежурств", "Сотрудники"), кнопки навигации, формы ввода данных. Стилизация в тематике Nord с закругленными углами и цветовой палитрой (#5E81AC, #88C0D0, #BF616A).  
Система будет поддерживать интеграцию с локальной базой данных PostgreSQL через SQLAlchemy, с возможностью расширения для подключения к внешним системам учета (например, кадровым базам) в будущем.

## **Репозиторий с документами в системе контроля версий**

Ссылка на проект загруженный на систему контроля версий [Электронный ресурс] URL: https://github.com/Mishanya322/Kyrsivaya.git (Дата обращения: хх.хх.2025)