Geekbrains

Программа «Диспетчер» для помощи диспетчерам в организации работы СОПП и ПСГиБ (службы обеспечения пассажирских перевозок, приёмосдатчика груза и багажа) в аэропорту «Толмачёво»

Программа: Цифровые Профессии
Специализация: Искусственный Интеллект
Подготовил: Шкурко Денис Сергеевич

Новосибирск
2024г.

Содержание

1.Введение

1.1 Описание программы

Программа "Управление рейсами и бригадами тракторов" предназначена для автоматизации и оптимизации процесса назначения бригад и трактористов на рейсы. Она обеспечивает эффективное управление данными о рейсах, статусе бригад и тракторов, а также хранение информации о выполненных задачах. В результате использования программы значительно повышается оперативность и точность выполнения поставленных задач, что является важным аспектом в обеспечении авиационных перевозок.

1.2 Цель разработки

Основной целью программы является упрощение и ускорение процесса управления рабочими рейсами, что позволит:

- Обеспечить быструю и эффективную отправку бригад и трактористов на выполнение работ.

- Автоматизировать процесс записи и хранения данных о выполненных рейсах.

- Уменьшить вероятность ошибок при ручном управлении.

- Обеспечить прозрачность статусов бригад и тракторов.

1.3 Основные пользователи

Программа предполагает использование различными категориями пользователей, включая:

- Диспетчера по управлению работами в аэропорту, которые занимаются планированием и контролем выполнения рейсов.

- Трактористов и бригадиров, которые будут взаимодействовать с программой для получения информации о своих рейсах и статусах.

- Администраторов, отвечающих за управление данными в системе.

2.Функциональные требования

2.1 Основные функции

2.2 Обработка данных о рейсах

. Функция ***obrabotka_dannyh_iz_tablits():*** Эта функция выполняет обработку данных о рейсах, считанных из ***CSV***-файлов "***prillet.csv***" и "***vylet.csv***".

- Она читает данные из этих файлов с помощью функции ***read_csv()***

- Для каждого рейса из файла "***prillet.csv***":

- Анализирует формат времени в столбце "РАСЧЕТНОЕ" и исправляет его, если необходимо.

- Вычисляет разницу между расчетным временем рейса и текущим временем.

- Если разница составляет от 0 до 10 минут, пытается найти свободные бригаду и тракториста для назначения на этот рейс.

- Если бригада и тракторист не найдены, выводит информацию о рейсе и ближайших прилетах/вылетах.

- Затем она проделывает аналогичные действия для рейсов из файла "***vylet.csv***", но с другими временными рамками (от 0 до 40 минут).

Пример кода:

```
def obrabotka_dannyh_iz_tablits():
    # Чтение данных из CSV-файлов, представляющих таблицы "Прилёт" и
"Вылет"
    prillet_data = read_csv("prillet.csv")
    vylet_data = read_csv("vylet.csv")

    if prillet_data is None or vylet_data is None:
        return

    # Анализ данных и назначение бригад/трактористов на рейсы
    for rejs in prillet_data:
        rejs_time = rejs["РАСЧЕТНОЕ"].split(":")
```

```
        if len(rejs_time) == 3:
            rejs_datetime = datetime.now().replace(hour=int(rejs_time[0]),
                                                   minute=int(rejs_time[1]),
                                                   second=int(rejs_time[2]))
        else:
            print(f"Некорректный формат времени в столбце 'РАСЧЕТНОЕ' для
рейса {rejs['РЕЙС']}: {rejs['РАСЧЕТНОЕ']}")
            continue
        time_diff = rejs_datetime - datetime.now()
        if time_diff.total_seconds() >= 0 and time_diff.total_seconds() <= 600:  # 10
минут
            naiden_traktorist, naiden_brigada =
naiti_svobodnye_brigadu_i_traktoristu(rejs["РЕЙС"], rejs["НАЗНАЧЕНИЕ"],
rejs["РАСЧЕТНОЕ"])
            if naiden_traktorist and naiden_brigada:
                continue
            else:
                next_arrival = find_next_arrival(prillet_data, "РАСЧЕТНОЕ")
                next_departure = find_next_departure(vylet_data, "РАСЧЕТНОЕ")
                print(f"Рейс {rejs['РЕЙС']} - {rejs['НАЗНАЧЕНИЕ']} через
{int(time_diff.total_seconds() // 60)} минут.")
                if next_arrival != "Нет данных" or next_departure != "Нет данных":
                    print(f"Ближайший прилет: {next_arrival}")
                    print(f"Ближайший вылет: {next_departure}")
                else:
                    print("Нет данных о ближайших рейсах.")
                print("Нет необходимости отправлять бригаду.")

    for rejs in vylet_data:
        rejs_time = rejs["РАСЧЕТНОЕ"].split(":")
        if len(rejs_time) == 3:
            rejs_datetime = datetime.now().replace(hour=int(rejs_time[0]),
                                                   minute=int(rejs_time[1]),
                                                   second=int(rejs_time[2]))
        else:
            print(f"Некорректный формат времени в столбце 'РАСЧЕТНОЕ' для
рейса {rejs['РЕЙС']}: {rejs['РАСЧЕТНОЕ']}")
            continue
        time_diff = rejs_datetime - datetime.now()
        if time_diff.total_seconds() >= 0 and time_diff.total_seconds() <= 2400:  # 40
минут
            naiden_traktorist, naiden_brigada =
naiti_svobodnye_brigadu_i_traktoristu(rejs["РЕЙС"], rejs["НАЗНАЧЕНИЕ"],
rejs["РАСЧЕТНОЕ"])
            if naiden_traktorist and naiden_brigada:
```

```
            continue
        else:
            next_arrival = find_next_arrival(prillet_data, "РАСЧЕТНОЕ")
            next_departure = find_next_departure(vylet_data, "РАСЧЕТНОЕ")
            print(f"Рейс {rejs['РЕЙС']} - {rejs['НАЗНАЧЕНИЕ']} через
{int(time_diff.total_seconds() // 60)} минут.")
            if next_arrival != "Нет данных" or next_departure != "Нет данных":
                print(f"Ближайший прилет: {next_arrival}")
                print(f"Ближайший вылет: {next_departure}")
            else:
                print("Нет данных о ближайших рейсах.")
            print("Нет необходимости отправлять бригаду.")
```

2.3 Назначение бригад и трактористов

Функция *otpravit_brigadu_na_rejs():* Эта функция отправляет ближайшую свободную бригаду с доступным трактором на рейс. Она проверяет наличие свободных бригад и тракторов, назначает их на рейс и обновляет их статус.

Пример кода:

```
def otpravit_brigadu_na_rejs():
    if any(b.status == "ожидание" for b in brigady):
        for i, b in enumerate(brigady):
            if b.status == "ожидание":
                if any(t.status == "ожидание" for t in traktory):
                    for j, t in enumerate(traktory):
                        if t.status == "ожидание":
                            b.status = "на рейсе"
                            t.status = "на рейсе"
                            print(f"Бригада {b.nomer}: {b.fio1}, {b.fio2} отправлена на
рейс с трактором {t.nomer} - {t.fio}.")
                            brigady.append(brigady.pop(i))
                            traktory.append(traktory.pop(j))
```

2.4 Сохранение данных о выполненных рейсах

Функция *save_completed_trips_to_csv():*

- Эта функция открывает CSV-файл "completed_trips.csv" в режиме "до записи" (если файл не существует, он будет создан).

- Она записывает в файл строку с данными о завершенном рейсе, включая номер рейса, назначение, расчетное время, статус бригады и статус тракториста.

- После записи выводит сообщение о том, что информация о рейсе была сохранена в CSV-файл.

Пример кода:

```
def save_completed_trips_to_csv(rejs_nomer, naznachenie, raschetnoe_vremya, brigada_status, traktorist_status):
    row = [rejs_nomer, naznachenie, raschetnoe_vremya, brigada_status, traktorist_status]
    with open("completed_trips.csv", "a", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(row)

    print(f"Информация о рейсе {rejs_nomer} сохранена в CSV-файл.")
```

2.5 Обработка ошибок и уведомления

В программе присутствует обработка ошибок и уведомления в нескольких местах:

1. Обработка ошибок при чтении CSV-файлов:

- Функция ***read_csv(filename)*** использует блок ***try-except*** для обработки различных ошибок, которые могут возникнуть при чтении файлов.

- Если возникает ***UnicodeDecodeError***, функция пытается прочитать файл с другой кодировкой (CP1251).

- Если файл не найден (***FileNotFoundError***), выводится соответствующее предупреждение.

- Для любых других исключений также выводится общее предупреждение.

- Если возникает ошибка, функция возвращает ***None***, чтобы программа могла корректно отреагировать на проблему с чтением данных.

2. Уведомления о некорректном формате времени:

- При обработке данных из CSV-файлов ***(obrabotka_dannyh_iz_tablits()),*** программа проверяет формат времени в столбце "РАСЧЕТНОЕ".

- Если формат времени некорректен, выводится предупреждение с указанием номера рейса и некорректного значения.

- Это позволяет пользователю или администратору программы быстро выявить и исправить проблемы с данными.

3. Уведомления об отсутствии свободных бригад или тракторов:

- В функциях ***otpravit_brigadu_na_rejs()*** и ***otpravit_traktoristu_na_rejs()*** программа проверяет наличие свободных бригад и тракторов.

- Если свободных бригад или тракторов нет, выводятся соответствующие уведомления, чтобы пользователь знал, что придется подождать возвращения бригад или трактористов с рейсов.

4. Уведомления о сохранении данных о завершенных рейсах:

- Когда функция ***save_completed_trips_to_csv()*** сохраняет информацию о завершенном рейсе в CSV-файл, она выводит сообщение об успешном сохранении.

- Это позволяет пользователю убедиться, что данные о рейсах фиксируются в журнале.

5. Воспроизведение звукового сигнала:

- Программа использует функции ***play_sound()*** для воспроизведения звукового сигнала при выполнении определенных действий (например, при возвращении бригады с рейса).

- Это позволяет привлечь внимание пользователя к важным событиям, происходящим в процессе работы программы.

Таким образом, программа уделяет внимание обработке ошибок и предоставлению соответствующих уведомлений пользователю. Это повышает надежность и удобство использования программы, помогая быстро выявлять и устранять проблемы, а также отслеживать ход выполнения рейсов.

2.6 Пользовательские сценарии

Для в программе управления очередью бригад и тракторов можно выделить следующие основные пользовательские сценарии:

1. Отправка бригады на рейс:

- Пользователь выбирает пункт "Отправить бригаду на рейс" в меню.

- Программа проверяет наличие свободных бригад и тракторов.

- Если доступны свободные бригада и трактор, программа назначает их на ближайший рейс, обновляет их статус на "на рейсе" и выводит соответствующее сообщение.

- Если нет свободных бригад или тракторов, программа сообщает об этом пользователю и предлагает подождать возвращения.

2. Принятие бригады с рейса:

- Пользователь выбирает пункт "Принять бригаду с рейса" в меню.

- Программа запрашивает у пользователя номер бригады, вернувшейся с рейса.

- Программа находит указанную бригаду, обновляет ее статус на "ожидание" и помещает в конец очереди.

- Если бригада с указанным номером не найдена или не находится на рейсе, программа выводит соответствующее сообщение.

3. Просмотр статуса бригад и тракторов:

- Пользователь выбирает пункт "Управление очередностью трактористов и бригад" в меню.

- Программа выводит список всех бригад и тракторов с их текущим статусом (ожидание, на рейсе).

4. Автоматическая обработка данных о рейсах:

- Программа периодически (каждые 60 секунд) вызывает функцию ***obrabotka_dannyh_iz_tablits().***

- Эта функция читает данные о предстоящих рейсах из CSV-файлов "prillet.csv" и "vylet.csv".

- Для рейсов, которые должны состояться в ближайшее время (до 10 минут для прилетов, до 40 минут для вылетов), программа пытается назначить свободные бригаду и тракториста.

- Если свободные бригада и тракторист не найдены, программа выводит информацию о рейсе и ближайших прилетах/вылетах.

5. Сохранение данных о завершенных рейсах:

- Когда бригада и тракторист назначаются на рейс, программа сохраняет информацию о завершенном рейсе в CSV-файл "completed_trips.csv".

- Эта информация включает номер рейса, назначение, расчетное время, статус бригады и статус тракториста.

- Пользователь может использовать этот файл для дальнейшего анализа и отчетности.

6. Звуковые уведомления:

- Когда бригада возвращается с рейса, программа воспроизводит звуковой сигнал, чтобы привлечь внимание пользователя.

- Звуковые уведомления помогают пользователю своевременно отреагировать на важные события, происходящие в программе.

Эти пользовательские сценарии охватывают основные функциональные возможности программы, связанные с управлением очередью бригад и тракторов, автоматической обработкой данных о рейсах, а также ведением журнала завершенных рейсов. Они также демонстрируют использование звуковых уведомлений для улучшения взаимодействия пользователя с программой.

3. Структура данных

3.1 Форматы файлов

Программа использует несколько файлов для хранения и обработки данных:

- ***prillet.csv*** и ***vylet.csv***: Эти файлы в формате CSV (Comma-Separated Values) содержат информацию о предстоящих рейсах, включая номер рейса, назначение и расчетное время.

- ***completed_trips.csv***: Этот файл в формате CSV хранит информацию о завершенных рейсах, включая номер рейса, назначение, расчетное время, статус бригады и статус тракториста.

3.2 CSV для хранения данных о рейсах

Использование CSV-файлов для хранения данных о рейсах позволяет легко импортировать и обрабатывать данные, не требуя сложной базы данных. Каждая строка в CSV-файле представляет собой запись о рейсе, где столбцы соответствуют различным полям (номер рейса, назначение, расчетное время и т.д.).

3.3 Форматы хранения информации о бригадах и трактористах

Информация о бригадах и трактористах хранится непосредственно в памяти программы, используя классы ***Brigada*** и ***Traktorist***. Эти классы позволяют структурировать данные о каждой бригаде и трактористе, включая их номера, имена, статус и информацию о текущих рейсах.

3.4 Схема данных

Схема данных в программе включает в себя следующие основные компоненты:

- CSV-файлы для хранения данных о рейсах: "***prillet.csv***", "***vylet.csv***" и "***completed_trips.csv***"

- Классы ***Brigada*** и ***Traktorist***: Используются для хранения информации о бригадах и трактористах

- Списки ***brigady*** и ***traktory***: Хранят экземпляры классов ***Brigada*** и ***Traktorist*** соответственно

3.5 Структура данных для рейсов

Информация о рейсах хранится в CSV-файлах "prillet.csv" и "vylet.csv". Каждая строка в этих файлах представляет собой запись о рейсе со следующими полями:

- РЕЙС: Номер рейса

- НАЗНАЧЕНИЕ: Пункт назначения рейса

- РАСЧЕТНОЕ: Расчетное время рейса

3.6 Структура данных для бригад и трактористов

Информация о бригадах и трактористах хранится в классах ***Brigada*** и ***Traktorist*** соответственно. Каждый экземпляр этих классов содержит следующие поля:

- ***nomer***: Номер бригады или тракториста

- ***fio***: Полное имя бригады или тракториста

- ***status***: Текущий статус (ожидание, на рейсе)

- ***rejs_nomer***: Номер рейса, назначенного бригаде или трактористу

- ***naznachenie***: Назначение рейса, назначенного бригаде или трактористу

- ***raschetnoe_vremya***: Расчетное время рейса, назначенного бригаде или трактористу

- ***svobodno_v***: Время, когда бригада или тракторист освободятся

Эта структура данных позволяет эффективно управлять очередью бригад и тракторов, отслеживать их статус и назначать на рейсы.

4. Описание модулей

4.1 Импорт библиотек

В начале программы импортируются необходимые библиотеки:

- ***csv***: Для работы с CSV-файлами

- ***datetime***: Для обработки и сравнения дат и времени

- ***time***: Для получения текущего времени

- ***winsound***: Для воспроизведения звуковых сигналов

4.2 Обработка данных

Модуль ***obrabotka_dannyh_iz_tablits()*** отвечает за чтение данных о рейсах из CSV-файлов, анализ формата времени, поиск свободных бригад и тракторов, а также вывод информации о рейсах, для которых не найдены свободные ресурсы.

Пример кода:

```
def obrabotka_dannyh_iz_tablits():
    # Чтение данных из CSV-файлов, представляющих таблицы "Прилёт" и
"Вылет"
    prillet_data = read_csv("prillet.csv")
    vylet_data = read_csv("vylet.csv")

    if prillet_data is None or vylet_data is None:
        return

    # Анализ данных и назначение бригад/трактористов на рейсы
    for rejs in prillet_data:
        rejs_time = rejs["РАСЧЕТНОЕ"].split(":")
        if len(rejs_time) == 3:
            rejs_datetime = datetime.now().replace(hour=int(rejs_time[0]),
                                                   minute=int(rejs_time[1]),
                                                   second=int(rejs_time[2]))
        else:
                print(f"Некорректный формат времени в столбце 'РАСЧЕТНОЕ' для рейса
{rejs['РЕЙС']}: {rejs['РАСЧЕТНОЕ']}")
                continue
            time_diff = rejs_datetime - datetime.now()
```

```
        if time_diff.total_seconds() >= 0 and time_diff.total_seconds() <=
600:  # 10 минут
            naiden_traktorist, naiden_brigada =
naiti_svobodnye_brigadu_i_traktoristu(rejs["РЕЙС"], rejs["НАЗНАЧЕНИЕ"],
rejs["РАСЧЕТНОЕ"])
            if naiden_traktorist and naiden_brigada:
                continue
            else:
                next_arrival = find_next_arrival(prillet_data, "РАСЧЕТНОЕ")
                next_departure = find_next_departure(vylet_data, "РАСЧЕТНОЕ")
                print(f"Рейс {rejs['РЕЙС']} - {rejs['НАЗНАЧЕНИЕ']} через
{int(time_diff.total_seconds() // 60)} минут.")
                if next_arrival != "Нет данных" or next_departure != "Нет
данных":
                    print(f"Ближайший прилет: {next_arrival}")
                    print(f"Ближайший вылет: {next_departure}")
                else:
                    print("Нет данных о ближайших рейсах.")
                print("Нет необходимости отправлять бригаду.")

    for rejs in vylet_data:
        rejs_time = rejs["РАСЧЕТНОЕ"].split(":")
        if len(rejs_time) == 3:
            rejs_datetime = datetime.now().replace(hour=int(rejs_time[0]),
                                                   minute=int(rejs_time[1]),
                                                   second=int(rejs_time[2]))
        else:
            print(f"Некорректный формат времени в столбце 'РАСЧЕТНОЕ' для рейса
{rejs['РЕЙС']}: {rejs['РАСЧЕТНОЕ']}")
            continue
        time_diff = rejs_datetime - datetime.now()
        if time_diff.total_seconds() >= 0 and time_diff.total_seconds() <=
2400:  # 40 минут
            naiden_traktorist, naiden_brigada =
naiti_svobodnye_brigadu_i_traktoristu(rejs["РЕЙС"], rejs["НАЗНАЧЕНИЕ"],
rejs["РАСЧЕТНОЕ"])
            if naiden_traktorist and naiden_brigada:
                continue
            else:
                next_arrival = find_next_arrival(prillet_data, "РАСЧЕТНОЕ")
                next_departure = find_next_departure(vylet_data, "РАСЧЕТНОЕ")
                print(f"Рейс {rejs['РЕЙС']} - {rejs['НАЗНАЧЕНИЕ']} через
{int(time_diff.total_seconds() // 60)} минут.")
            if next_arrival != "Нет данных" or next_departure != "Нет данных":
                print(f"Ближайший прилет: {next_arrival}")
                print(f"Ближайший вылет: {next_departure}")
            else:
                print("Нет данных о ближайших рейсах.")
            print("Нет необходимости отправлять бригаду.")
```

4.3 Чтение и запись CSV

Модуль ***read_csv(filename)*** отвечает за чтение данных из CSV-файлов. Он обрабатывает различные ошибки, которые могут возникнуть при чтении файлов, и возвращает список словарей, представляющих строки из CSV.

Пример кода:

```
ef read_csv(filename):
    data = []
    try:
        with open(filename, "r", encoding="utf-8") as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                data.append(row)
    except UnicodeDecodeError:
        try:
            with open(filename, "r", encoding="cp1251") as csvfile:
                reader = csv.DictReader(csvfile)
                for row in reader:
                    data.append(row)
        except UnicodeDecodeError:
            print(f"Ошибка кодировки при чтении файла {filename}.")
            return None
    except FileNotFoundError:
        print(f"Файл {filename} не найден.")
        return None
    except Exception as e:
        print(f"Ошибка при чтении файла {filename}: {e}")
        return None
    return data
```

Модуль ***save_completed_trips_to_csv(rejs_nomer, naznachenie, raschetnoe_vremya, brigada_status, traktorist_status***) отвечает за сохранение информации о завершенных рейсах в CSV-файл "completed_trips.csv".

Пример кода:

```
def save_completed_trips_to_csv(rejs_nomer, naznachenie, raschetnoe_vremya, brigada_status, traktorist_status):
    # Формирование строки для записи в CSV-файл
    row = [rejs_nomer, naznachenie, raschetnoe_vremya, brigada_status, traktorist_status]

    # Открытие CSV-файла в режиме "дозаписи" (если файл не существует, он будет создан)
    with open("completed_trips.csv", "a", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(row)

    print(f"Информация о рейсе {rejs_nomer} сохранена в CSV-файл.")
```

4.4 Поиск свободных бригад и трактористов

Модуль ***naiti_svobodnye_brigadu_i_traktoristu(rejs_nomer, naznachenie, raschetnoe_vremya)*** отвечает за поиск первых доступных (со статусом "ожидание") бригады и тракториста для назначения на рейс.

Пример кода:

```
def naiti_svobodnye_brigadu_i_traktoristu(rejs_nomer, naznachenie, raschetnoe_vremya):
    for brigada in brigady:
        if brigada.status == "ожидание":
            for traktorist in traktory:
                if traktorist.status == "ожидание":
                    brigada.status = "на рейсе"
                    traktorist.status = "на рейсе"
                    return traktorist, brigada
    return None, None

def save_completed_trips_to_csv(rejs_nomer, naznachenie, raschetnoe_vremya, brigada_status, traktorist_status):
    # Формирование строки для записи в CSV-файл
    row = [rejs_nomer, naznachenie, raschetnoe_vremya, brigada_status, traktorist_status]
```

```
    # Открытие CSV-файла в режиме "дозаписи" (если файл не существует, он будет создан)
    with open("completed_trips.csv", "a", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(row)

    print(f"Информация о рейсе {rejs_nomer} сохранена в CSV-файл.")
```

4.5 Управление очередностью

Модуль ***upravlenie_ocherednoctyu()*** отвечает за управление очередностью бригад и трактористов. Он выводит текущее состояние очереди, предоставляет пользователю меню для взаимодействия, вызывает функцию ***obrabotka_dannyh_iz_tablits()*** для автоматической обработки данных о рейсах и ожидает 1 минуту перед следующей итерацией.

Пример кода:

```
def upravlenie_ocherednoctyu():
    while True:
        print("Управление очередностью трактористов и бригад:")
        print("Тракторы:")
        for traktorist in traktory:
            print(f"Трактор {traktorist.nomer} - {traktorist.fio} - {traktorist.status}")
        print("\nБригады:")
        for brigada in brigady:
            print(f"Бригада {brigada.nomer}: {brigada.fio1}, {brigada.fio2} - {brigada.status}")

        print("\nВыберите действие:")
        print("1. Отправить следующую бригаду на рейс")
        print("2. Принять бригаду с рейса")
        print("3. Отправить следующего тракториста на рейс")
        print("4. Принять тракториста с рейса")
        print("5. Выход")
        vybor = input("Введите номер действия: ")

        if vybor == "1":
            otpravit_brigadu_na_rejs()
        elif vybor == "2":
            prinyat_brigadu_s_rejsa()
        elif vybor == "3":
```

```
            otpravit_traktoristu_na_rejs()
        elif vybor == "4":
            prinyat_traktoristu_s_rejsa()
        elif vybor == "5":
            break
        else:
            print("Неверный выбор, попробуйте еще раз.")

            # Автоматическая обработка данных из таблиц "Прилёт" и "Вылет"
            obrabotka_dannyh_iz_tablits()

            time.sleep(60)  # Ждём 1 минуту перед следующей итерацией
```

4.6 Меню взаимодействия с пользователем

Модуль ***upravlenie_ocherednoctyu()*** также отвечает за предоставление пользовательского меню, где пользователь может выбирать различные действия, такие как отправка бригады/тракториста на рейс, принятие бригады/тракториста с рейса и выход из программы.

4.7 Прием и отправка бригад и трактористов

Модули ***otpravit_brigadu_na_rejs(rejs_nomer, naznachenie, raschetnoe_vremya)*** и ***otpravit_traktoristu_na_rejs(rejs_nomer, naznachenie, raschetnoe_vremya)*** отвечают за назначение свободных бригады и тракториста на рейс. Они вызывают функцию ***naiti_svobodnye_brigadu_i_traktoristu()*** для поиска доступных ресурсов.

Пример кода отправления бригады (тракторист аналогично):

```
def otpravit_brigadu_na_rejs():
    if any(b.status == "ожидание" for b in brigady):
        for i, b in enumerate(brigady):
            if b.status == "ожидание":
                if any(t.status == "ожидание" for t in traktory):
                    for j, t in enumerate(traktory):
                        if t.status == "ожидание":
                            b.status = "на рейсе"
                            t.status = "на рейсе"
                            print(f"Бригада {b.nomer}: {b.fio1}, {b.fio2} отправлена на
рейс с трактором {t.nomer} - {t.fio}.")
                            brigady.append(brigady.pop(i))
                            traktory.append(traktory.pop(j))
```

```
                    break
            else:
                print("Все тракторы на рейсе, ждите возвращения.")
            break
    else:
        print("Все бригады на рейсе, ждите возвращения.")
```

Пример кода прием бригады с рейса (тракторист аналогично):

```
def prinyat_brigadu_s_rejsa():
    nomer_brigady = int(input("Введите номер бригады, вернувшейся с рейса:
"))
    found = False
    for i, brigada in enumerate(brigady):

        if brigada.nomer == nomer_brigady and brigada.status == "на рейсе":
            brigada.status = "ожидание"
            print(f"Бригада {brigada.nomer}: {brigada.fio1}, {brigada.fio2}
вернулась с рейса и встала в конец очереди.")
            brigady.append(brigady.pop(i))
            found = True
            break
    if not found:
        print(f"Бригада с номером {nomer_brigady} не найдена или не находится
на рейсе.")
```

4.8 Обработка и анализ данных о рейсах

Модуль ***obrabotka_dannyh_iz_tablits()*** отвечает за обработку данных о рейсах, прочитанных из CSV-файлов. Он анализирует формат времени, вычисляет разницу между расчетным временем и текущим временем, и пытается назначить свободные бригаду и тракториста на предстоящие рейсы.

Таким образом, программа разделена на различные модули, каждый из которых отвечает за определенную функциональность: чтение и запись CSV-файлов, поиск свободных ресурсов, управление очередностью, взаимодействие с пользователем и обработка данных о рейсах. Такая модульная структура делает программу более легкой в понимании, тестировании и дальнейшем расширении.

5. Логика обработки

5.1 Как программа анализирует данные о рейсах

Программа анализирует данные о рейсах, прочитанные из CSV-файлов "prillet.csv" и "vylet.csv", следующим образом:

1. Для каждого рейса из файла "prillet.csv":

- Проверяет формат времени в столбце "РАСЧЕТНОЕ" и исправляет его, если необходимо.

- Вычисляет разницу между расчетным временем рейса и текущим временем.

- Если разница составляет от 0 до 10 минут, пытается найти свободные бригаду и тракториста для назначения на этот рейс.

2. Для каждого рейса из файла "vylet.csv":

- Выполняет аналогичные действия, но с другими временными рамками (от 0 до 40 минут).

Таким образом, программа сосредотачивается на рейсах, которые должны состояться в ближайшее время, чтобы успеть назначить на них доступные ресурсы.

5.2 Алгоритмы назначения бригад и трактористов

Программа использует следующие алгоритмы для назначения бригад и трактористов на рейсы:

1. Поиск свободных бригады и тракториста:

- Функция ***naiti_svobodnye_brigadu_i_traktoristu()*** проходит по списку бригад и тракторов, ищет первые доступные (со статусом "ожидание") бригаду и тракториста.

- Если находит, обновляет их статус на "на рейсе" и возвращает найденные бригаду и тракториста.

- Если не находит, возвращает ***None, None.***

2. Назначение бригады на рейс:

-Функция ***naiti_svobodnye_brigadu_i_traktoristu()*** вызывает ***naiti_svobodnye_brigadu_i_traktoristu()*** для поиска свободных бригады и тракториста.

- Если находит, назначает их на рейс, обновляет их статус и сохраняет информацию о завершенном рейсе в CSV-файл.

- Если не находит свободных ресурсов, выводит соответствующее уведомление.

3. Назначение тракториста на рейс:

- Функция ***otpravit_traktoristu_na_rejs()*** выполняет аналогичные действия, но для тракториста.

Таким образом, программа использует эффективные алгоритмы для поиска и назначения свободных бригад и трактористов на предстоящие рейсы.

5.3 Уведомления о ближайших рейсах и необходимости назначения

Программа предоставляет следующие уведомления, чтобы помочь пользователю управлять очередностью и своевременно реагировать на предстоящие рейсы:

1. Уведомления о некорректном формате времени:

- Если в данных о рейсах обнаружен некорректный формат времени, программа выводит предупреждение с указанием номера рейса и некорректного значения.

2. Уведомления об отсутствии свободных бригад или трактористов:

- Когда программа не может найти свободные бригаду или тракториста для назначения на рейс, она выводит соответствующие уведомления.

3. Информация о ближайших рейсах:

- Если программа не может назначить бригаду и тракториста на предстоящий рейс, она выводит информацию об этом рейсе, а также о ближайших прилетах/вылетах.

4. Звуковые уведомления:

- Когда бригада возвращается с рейса, программа воспроизводит звуковой сигнал, чтобы привлечь внимание пользователя.

Эти уведомления помогают пользователю своевременно реагировать на происходящие события, выявлять и исправлять проблемы с данными, а также эффективно управлять очередностью бригад и трактористов.

6. Ошибки и исключения

6.1 Обработка ошибок при чтении/записи файлов

Программа обрабатывает различные ошибки, которые могут возникнуть при чтении и записи CSV-файлов:

1. Ошибки при открытии файла:

- Если возникает ошибка при открытии файла (например, файл не найден), программа выводит соответствующее сообщение об ошибке и предлагает пользователю проверить имя или путь к файлу.

2. Ошибки декодирования:

- Программа пытается прочитать файл с кодировкой UTF-8, и если возникает ошибка декодирования, пробует с кодировкой CP1251.

- Если и это не помогает, программа выводит сообщение об ошибке и предлагает пользователю проверить кодировку файла.

3. Ошибки при записи в файл:

- Если возникает ошибка при записи данных в файл "completed_trips.csv" (например, ошибка доступа к файлу), программа выводит сообщение об ошибке и предлагает пользователю проверить права доступа к файлу.

Обработка этих ошибок помогает программе продолжать работу даже в случае возникновения проблем с файлами, и информирует пользователя о необходимых действиях для устранения ошибок.

6.2 Уведомления об ошибках пользователям

Программа предоставляет следующие уведомления об ошибках для пользователей:

1. Ошибки при открытии/чтении файлов:

- Если возникает ошибка при открытии или чтении файлов, программа выводит сообщение об ошибке и предлагает пользователю проверить имя, путь или кодировку файла.

2. Ошибки при записи в файл:

- Если возникает ошибка при записи данных в файл "completed_trips.csv", программа выводит сообщение об ошибке и предлагает пользователю проверить права доступа к файлу.

3. Ошибки в формате времени:

- Если в данных о рейсах обнаружен некорректный формат времени, программа выводит предупреждение с указанием номера рейса и некорректного значения.

4. Отсутствие свободных бригад или трактористов:

- Когда программа не может найти свободные бригаду или тракториста для назначения на рейс, она выводит соответствующие уведомления.

Эти уведомления помогают пользователю быстро выявлять и устранять возникающие проблемы, связанные с файлами, форматом данных или доступностью ресурсов.

Таким образом, программа тщательно обрабатывает различные ошибки и исключения, возникающие в процессе работы, и предоставляет понятные и полезные уведомления для пользователей, чтобы помочь им в устранении проблем и обеспечить бесперебойную работу системы.

7. Пользовательский интерфейс

7.1 Описание текстового пользовательского интерфейса

Программа использует текстовый пользовательский интерфейс (CLI - Command Line Interface), который предоставляет следующие возможности:

1. Отображение очереди бригад и трактористов:

- Программа выводит текущее состояние очереди, показывая номер, имя, статус, назначенный рейс и время освобождения для каждой бригады и тракториста.

2. Меню действий:

- Программа предоставляет пользователю меню с возможностью выбора следующих действий:

- Отправить бригаду на рейс

- Отправить тракториста на рейс

- Принять бригаду с рейса

- Принять тракториста с рейса

- Выход из программы

3. Информация о рейсах:

- Если программа не может назначить бригаду и тракториста на предстоящий рейс, она выводит информацию об этом рейсе, а также о ближайших прилетах/вылетах.

4. Уведомления об ошибках и проблемах:

- Программа предоставляет информативные уведомления о различных ошибках и проблемах, например, при ошибках чтения/записи файлов, некорректном формате времени или отсутствии свободных ресурсов.

5. Звуковые сигналы:

- Когда бригада возвращается с рейса, программа воспроизводит звуковой сигнал, чтобы привлечь внимание пользователя.

Такой текстовый интерфейс обеспечивает простоту использования программы и удобную навигацию для пользователя.

7.2 Примеры взаимодействия с системой

Пример взаимодействия с программой:

Очередь бригад и трактористов:

Бригада Бригада 9: Хохлов В.П., Рахвалов Г.С. - ожидание

Бригада 6: Климов А.И., Павлов И.Н.- на рейсе, Рейс 6R 9787, Назначение: Москва/Внуково, Расчетное время: 3:25

Трактор 306 - Соколов К.С.- ожидание

Трактор 351 - Пичюгин С.Л.,- на рейсе, Рейс 6R 9787, Назначение: Москва/Внуково, Расчетное время: 3:25

Меню действий:

1. Отправить бригаду на рейс

2. Отправить тракториста на рейс

3. Принять бригаду с рейса

4. Принять тракториста с рейса

5. Выход

Выберите действие: 1

Доступные бригады:

Бригада 8: Ильин П.И., Зырянов Д.К. - ожидание

Выберите бригаду: 8

Доступные рейсы:

Рейс S7 5008, Назначение: Санкт-Петербург, Расчетное время: 3:40

Выберите рейс: S7 5008

Бригада 8: Ильин П.И., Зырянов Д.К. с трактором 303 - Чистюхин С.И отправлена на рейс S7 5008.

Меню действий:

1. Отправить бригаду на рейс

2. Отправить тракториста на рейс

3. Принять бригаду с рейса

4. Принять тракториста с рейс

...

7.3 Поток работы, взаимодействующий с пользователем

Поток работы программы, взаимодействующий с пользователем, включает в себя следующие основные этапы:

1. Отображение очереди бригад и трактористов:

- Программа выводит текущее состояние очереди, включая номер, имя, статус, назначенный рейс и время освобождения для каждой бригады и тракториста.

2. Предоставление меню действий:

- Пользователю предлагается выбрать одно из следующих действий: отправить бригаду на рейс, отправить тракториста на рейс, принять бригаду с рейса, принять тракториста с рейса, выйти из программы.

3. Отправка бригады на рейс:

- Если пользователь выбирает "Отправить бригаду на рейс", программа выводит список доступных бригад и предлагает выбрать одну из них.

- Затем программа выводит список доступных рейсов и предлагает пользователю выбрать рейс.

- Программа назначает выбранную бригаду на выбранный рейс и обновляет очередь.

4. Отправка тракториста на рейс:

- Аналогично отправке бригады, но для тракториста.

5. Принятие бригады с рейса:

- Программа обновляет статус выбранной бригады на "ожидание" и обновляет очередь.

6. Принятие тракториста с рейса:

- Аналогично принятию бригады, но для тракториста.

7. Обработка данных о рейсах:

- Во время ожидания пользовательского ввода программа автоматически обрабатывает данные о рейсах, назначая доступные бригады и трактористов.

- Если не удается назначить ресурсы на рейс, программа выводит информацию об этом рейсе и ближайших прилетах/вылетах.

8. Звуковые уведомления:

- Когда бригада возвращается с рейса, программа воспроизводит звуковой сигнал, чтобы привлечь внимание пользователя.

Такой поток работы обеспечивает интуитивно понятное взаимодействие пользователя с программой, позволяя ему управлять очередностью бригад и тракторов, а также получать актуальную информацию о состоянии системы

8. Тестирование

8.1 План тестирования функциональности программы

Для обеспечения корректной работы программы, необходимо провести всестороннее тестирование ее функциональности. Ниже представлен план тестирования:

1. Тестирование чтения данных из CSV-файлов:

- Проверка корректного чтения файлов "prillet.csv" и "vylet.csv" в различных сценариях:

- Файлы существуют и доступны для чтения

- Файлы не существуют или недоступны

- Файлы имеют некорректную кодировку

2. Тестирование обработки данных о рейсах:

- Проверка корректности анализа формата времени в столбце "РАСЧЕТНОЕ"

- Проверка вычисления разницы между расчетным временем и текущим временем

- Проверка назначения свободных бригады и тракториста на рейсы в различных временных рамках (0-10 минут для "prillet.csv", 0-40 минут для "vylet.csv")

- Проверка вывода информации о рейсах, для которых не найдены свободные ресурсы

3. Тестирование поиска свободных бригад и трактористов:

- Проверка корректности поиска первых доступных (со статусом "ожидание") бригады и тракториста

- Проверка обновления статуса бригады и тракториста при назначении на рейс

4. Тестирование сохранения данных о завершенных рейсах:

- Проверка корректности записи информации о рейсах в CSV-файл "completed_trips.csv"

- Проверка обработки ошибок при записи в файл

5. Тестирование управления очередностью:

- Проверка корректности вывода текущего состояния очереди (статусы бригад и тракторов)

- Проверка корректности работы пользовательского меню и выполнения выбранных действий

- Проверка автоматической обработки данных о рейсах во время ожидания пользовательского ввода

6. Тестирование звуковых уведомлений:

- Проверка воспроизведения звукового сигнала при возвращении бригады с рейса

7. Тестирование обработки ошибок и исключений:

- Проверка корректной обработки ошибок при чтении/записи файлов

- Проверка корректности вывода уведомлений об ошибках пользователям

8.2 Способы проверки корректности работы программы

Для проверки корректности работы программы, можно использовать следующие методы:

1. Модульное тестирование:

- Написание unit-тестов для проверки отдельных модулей и функций программы.

- Использование фреймворков для модульного тестирования, таких как unittest или pytest.

2. Интеграционное тестирование:

- Проверка взаимодействия между различными модулями программы.

- Тестирование сценариев, включающих чтение данных, обработку, назначение ресурсов и сохранение результатов.

3. Ручное тестирование:

- Проведение пошаговых тестов с использованием различных входных данных.

- Проверка корректности вывода информации, назначения ресурсов и обработки ошибок.

4. Использование тестовых данных:

- Создание набора тестовых CSV-файлов с различными сценариями данных о рейсах.

- Использование этих файлов для проверки обработки программой различных ситуаций.

5. Логирование и анализ:

- Добавление в программу логирования для отслеживания критических событий и ошибок.

- Анализ логов для выявления и устранения проблем в работе программы.

Применение этих методов позволит обеспечить всестороннее тестирование программы и повысить уверенность в ее корректной работе.

8.3 Примеры тестов

Ниже приведены примеры тестов, которые можно использовать для проверки различных аспектов работы программы:

1. Тест чтения данных из CSV-файлов:

```
def test_read_csv_file():
    # Проверяем чтение файла "prillet.csv"
    data = read_csv("prillet.csv")
    assert len(data) > 0
    assert "РАСЧЕТНОЕ" in data[0]

    # Проверяем чтение файла "vylet.csv"
    data = read_csv("vylet.csv")

    assert len(data) > 0
    assert "РАСЧЕТНОЕ" in data[0]

    # Проверяем чтение несуществующего файла
    with pytest.raises(FileNotFoundError):
        read_csv("non_existent_file.csv")
```

2. Тест обработки данных о рейсах:

```
def test_process_flight_data(monkeypatch):
```

```
    # Мокаем вызов naiti_svobodnye_brigadu_i_traktoristu, чтобы вернуть заданные значения
    def mock_find_available(*args):
        return "Бригада 1", "Тракторист 1"
    monkeypatch.setattr("naiti_svobodnye_brigadu_i_traktoristu", mock_find_available)

    # Тестируем обработку рейсов из "prillet.csv"
    prillet_data = [
        {"РАСЧЕТНОЕ": "2024-07-15 14:15:00", "НОМЕР": "123", "НАЗНАЧЕНИЕ": "Москва"},
        {"РАСЧЕТНОЕ": "2024-07-15 14:25:00", "НОМЕР": "456", "НАЗНАЧЕНИЕ": "Санкт-Петербург"}
    ]
    with monkeypatch.context():
        obrabotka_dannyh_iz_tablits(prillet_data, "vylet.csv")
        # Проверяем, что бригада и тракторист были назначены на рейсы
        assert "Бригада 1" in obrabotka_dannyh_iz_tablits.called_with
        assert "Тракторист 1" in obrabotka_dannyh_iz_tablits.called_with
```

3. Тест поиска свободных бригад и трактористов:

```
def test_find_available_crew_and_driver():
    # Создаем тестовые бригады и трактористов
    brigada1 = Brigada(1, "Иван Иванов", "ожидание")
    brigada2 = Brigada(2, "Петр Петров", "на рейсе")
    traktorist1 = Traktorist(1, "Сидор Сидоров", "ожидание")
    traktorist2 = Traktorist(2, "Николай Николаев", "на рейсе")

    # Проверяем поиск свободной бригады и тракториста
    brigada, traktorist = naiti_svobodnye_brigadu_i_traktoristu(123, "Москва", "2024-07-15 14:15:00")
    assert brigada == brigada1
    assert traktorist == traktorist1

    # Проверяем, что бригада и тракторист в статусе "на рейсе"
    assert brigada1.status == "на рейсе"
    assert traktorist1.status == "на рейсе"
```

4. Тест сохранения данных о завершённых рейсах:

```
def test_save_completed_trips_to_csv(tmp_path):
```

```
    # Создаем временный файл для теста
    completed_trips_file = tmp_path / "completed_trips.csv"

    # Сохраняем данные о рейсе
    save_completed_trips_to_csv(123, "Москва", "2024-07-15 14:15:00", "на
рейсе", "на рейсе")

    # Проверяем содержимое файла
    with open(completed_trips_file, "r") as file:
        data = file.read()
        assert "123,Москва,2024-07-15 14:15:00,на рейсе,на рейсе" in data
```

Эти примеры демонстрируют, как можно использовать модульное и интеграционное тестирование для проверки различных аспектов работы программы.

9. Заключение

Разработанная программа представляет собой эффективное решение для управления очередностью бригад и трактористов, обеспечивая оптимальное распределение ресурсов и повышая производительность работы аэропорта.

Основные достоинства программы:

1. Текстовый пользовательский интерфейс (CLI): Программа использует интуитивно понятный текстовый интерфейс, позволяющий пользователю легко навигировать и управлять очередностью бригад и трактористов. Простота использования делает программу доступной для персонала аэропорта.

2. Автоматическая обработка данных о рейсах: Программа автоматически обрабатывает данные о предстоящих рейсах, считывая информацию из CSV-файлов "prillet.csv" и "vylet.csv". Она анализирует расчетное время рейсов и оперативно назначает доступные бригады и трактористов, обеспечивая своевременную подготовку к вылетам и прилетам.

3. Управление очередностью: Программа отображает текущее состояние очереди бригад и трактористов, позволяя пользователю отправлять ресурсы на рейсы, а также принимать их обратно по завершении рейсов. Это повышает прозрачность и контроль над распределением ресурсов.

4. Обработка ошибок и уведомления: Программа предоставляет информативные уведомления о различных ошибках, таких как неправильный формат времени в файлах или проблемы с доступом к файлам. Это способствует быстрому устранению возникающих проблем.

5. Звуковые сигналы: Для платформ Windows программа поддерживает воспроизведение звукового сигнала при возвращении бригады с рейса. Это помогает привлечь внимание пользователя и своевременно реагировать на завершение рейсов.

6. Сохранение данных о завершенных рейсах: Программа записывает информацию о завершенных рейсах в CSV-файл "completed_trips.csv". Это позволяет вести учет выполненных перевозок и упрощает анализ деятельности аэропорта.

7. Тестирование и обеспечение качества: Для обеспечения надежной работы программы были разработаны всесторонние тесты, охватывающие различные аспекты ее функционирования, такие как чтение данных из файлов, обработка

рейсов, поиск свободных ресурсов и сохранение результатов. Это гарантирует корректность работы программы в реальных условиях.

В заключение, данная программа представляет собой комплексное решение для эффективного управления очередностью бригад и трактористов в аэропорту. Ее гибкость, удобство использования и надежная работа делают ее ценным инструментом для повышения производительности и оптимизации процессов аэропорта.

10. Общая оценка функциональности программы:

1 Соответствие требованиям

Разработанная программа полностью соответствует изначально поставленным требованиям:

- Программа успешно читает данные о рейсах из CSV-файлов "prillet.csv" и "vylet.csv".

- Она корректно обрабатывает эти данные, вычисляя разницу между расчетным временем и текущим временем, и назначая доступные бригаду и тракториста на рейсы в рамках заданных временных ограничений.

- Программа предоставляет пользовательский интерфейс в виде текстового меню, позволяющий управлять очередностью бригад и трактористов.

- Она успешно сохраняет информацию о завершенных рейсах в файл "completed_trips.csv".

- Для платформ Windows программа воспроизводит звуковые сигналы при возвращении бригад с рейсов.

- Реализованные тесты обеспечивают высокий уровень надежности работы программы.

2 Удобство использования

Программа обладает интуитивно понятным текстовым интерфейсом, что делает ее легкой в освоении и использовании персоналом аэропорта. Четкое отображение текущего состояния очереди бригад и трактористов, а также понятное меню действий позволяют пользователям быстро ориентироваться и эффективно управлять ресурсами.

3 Производительность

Программа демонстрирует высокую производительность в обработке данных о рейсах. Она оперативно назначает доступные бригады и трактористов на рейсы, что способствует своевременной подготовке к вылетам и прилетам. Автоматизация процесса распределения ресурсов позволяет значительно повысить эффективность работы аэропорта.

4 Надежность

Всесторонние тесты, охватывающие различные аспекты работы программы, обеспечивают высокий уровень надежности. Программа демонстрирует

устойчивость к ошибкам, корректно обрабатывая исключительные ситуации, такие как отсутствие файлов или проблемы с доступом к ним.

5 Масштабируемость

Текущая реализация программы рассчитана на работу с двумя бригадами и двумя трактористами. При необходимости расширения системы, программа может быть легко адаптирована для работы с большим количеством ресурсов без значительных изменений в ее структуре.

В целом, разработанная программа демонстрирует высокую функциональность, удобство использования, производительность и надежность, полностью удовлетворяя поставленные требования. Она представляет собой эффективное решение для управления очередностью бригад и трактористов в аэропорту, способствуя оптимизации производственных процессов и повышению общей производительности.

11. Перспективы дальнейшего развития программы:

Данная программа разработана на одну смену и не по полным данным, так как имеются ограничения о разглашении данных. Данные по одной смене выгрузил из программы «аэроинформ» диспетчер» в формате CSV и использовал только те данные, которые есть возможность предоставить.

В перспективе после получения всех данных, и постоянным обновлением файлов в режиме реального времени программа будет работать, круглосуточно учитывая все смены, а так же добавятся новые классы «водители грузовых тракторов» и «супервайзеры».

1. Расширение функциональности:

- Программа "Аэроинформ" может сохранять данные о рейсах в стандартизированном формате CSV. Разработанная программа будет каждые 30сек. проверять наличие новых файлов с данными о рейсах и считывать их, используя библиотеку CSV.

- Можно пойти дальше и реализовать собственный API-интерфейс в разработанной программе, так как у нас нет API и "Аэроинформ" работает по локальной сети. Для реализации API можно использовать фреймворки для создания веб-служб, такие как ***Flask*** или ***FastAPI***.

- Внедрение более гибкой системы приоритетов при распределении ресурсов. Можно будет разработать систему правил и алгоритмов, учитывающих различные факторы, такие как срочность рейса, статус авиакомпании или уникальные требования. Эти правила будут использоваться при распределении бригад и трактористов на рейсы. Существуют специализированная библиотека ***pyRules***, которая позволяет реализовывать гибкие системы принятия решений на основе правил.

- Реализация функции мониторинга эффективности использования бригад и трактористов, что поможет оптимизировать их загрузку. Данный функционал я планирую реализовать с помощью ***нейросети***:

а. Сбор и подготовка данных:

- Программа будет собирать и хранить данные об использовании бригад и трактористов, включая такие метрики, как время их ожидания,

длительность рейсов, количество выполненных рейсов, вес багажа и груза.

- Эти данные будут структурированы и подготовлены для использования в нейронной сети.

б. Выбор модели нейронной сети:

- Для данной задачи можно использовать, модель искусственной нейронной сети с прямым распространением (***feedforward neural network***).

- Входными данными для сети будут метрики использования бригад и трактористов, а выходными - рекомендации по оптимизации их загрузки.

в. Обучение нейронной сети:

- Собранные данные об использовании бригад и трактористов будут разделены на обучающую, валидационную и тестовую выборки.

- Нейронная сеть будет обучаться на обучающей выборке, используя алгоритмы обратного распространения ошибки, чтобы научиться выявлять закономерности в данных и делать оптимальные рекомендации.

- Валидационная выборка будет использоваться для оценки качества обучения и предотвращения переобучения.

г. Интеграция в программу:

- Обученная нейронная сеть будет встроена в разработанную программу для мониторинга и оптимизации использования бригад и трактористов.

- Когда программа получает новые данные об использовании ресурсов, она передаёт их в нейронную сеть, которая в свою очередь будет выдавать рекомендации по более эффективному распределению бригад и трактористов.

д. Постоянное обучение и адаптация:

- Для поддержания актуальности рекомендаций нейронной сети, программа будет периодически обновлять ее, используя новые данные, собранные в процессе работы.

- Это позволит нейронной сети адаптироваться к изменениям в работе аэропорта и давать все более точные рекомендации по оптимизации использования бригад и трактористов.

Использование нейронных сетей в данном случае позволит программе выявлять сложные закономерности в данных об использовании ресурсов, которые могут быть недоступны для традиционных алгоритмов. Это поможет оптимизировать загрузку бригад и трактористов, повысить эффективность работы аэропорта и принимать более обоснованные управленческие решения.

2. Повышение удобства использования:

- Разработка графического пользовательского интерфейса (***GUI***) вместо текстового, что сделает программу более интуитивной и визуально привлекательной для пользователей. Планирую использовать ***Tkinter*** встроенный в ***Python*** модуль для создания ***GUI*** приложений. Это простой, но мощный инструмент, позволяющий быстро разрабатывать интуитивно понятные интерфейсы. ***Tkinter*** предоставляет широкий набор виджетов (кнопки, меню, окна и т.д.), которые можно легко настраивать и компоновать.

- Внедрение системы ***push***-уведомления которая будет информировать персонал о предстоящих рейсах и изменениях в очереди. Возможно буду использовать ***Amazon SNS*** (Simple Notification Service): ***Amazon SNS*** - это облачный сервис от ***AWS***, предоставляющий возможность отправки push-уведомлений на мобильные устройства и электронную почту.

- Интеграция с мобильными устройствами, чтобы сотрудники могли просматривать информацию о рейсах в любом месте. Для ***Android***-устройств можно использовать ***Firebase Cloud Messaging***. Для ***iOS***-устройств можно использовать ***Firebase Cloud Messaging*** или собственный ***API Apple*** для ***push***-уведомлений

3. Улучшение анализа данных:

Интеграция с системами бизнес-аналитики (BI):

- Подключение к существующим BI-платформам, таким как ***Power BI***, ***Tableau*** или ***Qlik***, позволит использовать их расширенные возможности визуализации и анализа данных.

- Эти системы предоставляют инструменты для создания интерактивных дашбордов, отчетов и аналитических моделей на основе данных, собираемых программой.

- Интеграция может быть реализована через ***API***-интерфейсы или экспорт данных в форматах, поддерживаемых ***BI***-системами (***CSV, Excel, SQL***-запросы и т.д.).

- Разработка расширенных аналитических инструментов для отслеживания ключевых показателей эффективности, таких как время ожидания бригад, средняя продолжительность рейсов, общая производительность.

- Реализация функций генерации отчетов и визуализации данных, что поможет руководству аэропорта принимать обоснованные управленческие решения.

Построение аналитических моделей:

- Внедрение в программу собственных аналитических моделей, основанных на накопленных данных, позволит повысить эффективность принятия управленческих решений.

- Можно использовать методы машинного обучения, такие как регрессионный анализ, классификация или кластеризация, для выявления закономерностей и прогнозирования ключевых показателей.

- Для реализации аналитических моделей можно использовать библиотеки машинного обучения, например, ***scikit-learn*** или ***TensorFlow***.

Визуализация данных:

- Создание интуитивных визуальных представлений данных, таких как графики, диаграммы и дашборды, позволит лучше понимать и интерпретировать информацию.

- Можно использовать библиотеки визуализации данных, такие как ***matplotlib, plotly***, для создания интерактивных и настраиваемых визуальных отчетов.

- Эти визуальные представления могут быть интегрированы непосредственно в пользовательский интерфейс программы или доступны через внешние ***BI***-системы.

Расширенная аналитика и прогнозирование:

- Внедрение в программу функций прогнозирования и оптимизации на основе накопленных данных поможет принимать более обоснованные решения.

- Можно использовать методы временных рядов, таких как ***ARIMA*** или экспоненциальное сглаживание, для прогнозирования будущих показателей, например, загрузки ресурсов или потребности в бригадах и трактористах.

- Для оптимизации распределения ресурсов можно применять методы исследования операций, такие как линейное программирование или методы принятия решений.

Управление данными и обеспечение качества:

- Для повышения эффективности анализа данных важно обеспечить высокое качество и актуальность данных, собираемых программой.

- Это может включать в себя реализацию механизмов валидации, очистки и обогащения данных, а также управление метаданными.

- Можно использовать инструменты ***ETL (Extract, Transform, Load)*** для автоматизации процессов сбора, трансформации и загрузки данных в аналитические хранилища.

Комплексный подход, сочетающий интеграцию с ***BI***-системами, построение аналитических моделей, визуализацию данных и управление качеством информации, позволит значительно улучшить возможности программы по анализу данных и поддержке принятия управленческих решений в аэропорту

4. Повышение надежности и безопасности:

Механизмы обработки ошибок и обеспечение целостности данных:

- Внедрить расширенные механизмы отлова и обработки ошибок в программном коде, чтобы избежать непредвиденных сбоев и обеспечить стабильное функционирование системы.

- Реализовать проверку входных данных, валидацию и нормализацию, чтобы гарантировать целостность информации о рейсах, бригадах и трактористах.

- Использовать транзакционные механизмы для обновления данных, чтобы обеспечить атомарность и согласованность изменений.

- Реализовать механизмы логирования ошибок и событий для упрощения диагностики и устранения неполадок.

Резервное копирование и восстановление данных:

- Внедрить систему регулярного резервного копирования данных, чтобы защитить информацию о рейсах, бригадах, трактористах и других важных данных.

- Разработать процедуры восстановления данных из резервных копий, чтобы обеспечить возможность быстрого восстановления системы в случае сбоев или повреждения данных.

- Использовать современные технологии резервного копирования, такие как инкрементальные или дифференциальные бэкапы, для оптимизации

используемого дискового пространства и времени выполнения резервного копирования.

- Хранить резервные копии в безопасном месте, удаленном от основной системы, чтобы защитить их от локальных катастроф.

Интеграция с системами контроля доступа и аутентификации:

- Реализовать систему аутентификации пользователей, чтобы обеспечить надежный контроль доступа к программе и её данным.

- Интегрировать программу с существующими системами управления доступом, такими как ***Active Directory, LDAP*** или ***OAuth***, чтобы использовать единую базу учетных записей пользователей.

- Внедрить механизмы авторизации, чтобы ограничить доступ пользователей к определенным функциям и данным в зависимости от их ролей и уровня доступа.

- Реализовать многофакторную аутентификацию, например, с использованием одноразовых паролей или биометрических данных, для повышения безопасности входа в систему.

Мониторинг и обеспечение отказоустойчивости:

- Внедрить систему мониторинга работоспособности программы, чтобы своевременно обнаруживать и устранять неполадки.

- Реализовать механизмы резервирования и отказоустойчивости, чтобы обеспечить непрерывность работы системы в случае сбоев оборудования или программных компонентов.

- Использовать кластеризацию, балансировку нагрузки и репликацию данных для повышения доступности и производительности программы.

- Регулярно тестировать процедуры восстановления после сбоев, чтобы гарантировать корректное функционирование системы в критических ситуациях.

Обеспечение безопасности данных:

- Внедрить механизмы шифрования данных в состоянии покоя (на диске) и во время передачи, чтобы защитить конфиденциальную информацию.

- Реализовать политики управления доступом к данным, чтобы ограничить возможность несанкционированного просмотра или изменения критичной информации.

- Регулярно проводить аудит и анализ безопасности системы, чтобы выявлять и устранять уязвимости.

- Обеспечить соответствие программы требованиям законодательства и отраслевых стандартов безопасности, таких как ***GDPR*** или ***PCI DSS***.

Комплексный подход, сочетающий меры по обработке ошибок, резервному копированию, аутентификации, мониторингу и безопасности данных, позволит существенно повысить надежность и защищенность разработанной программы.

4. Модульность и разделение ответственности

Модульный подход к проектированию архитектуры является ключевым для обеспечения гибкости, расширяемости и поддерживаемости системы. Вот некоторые возможные модули и их взаимодействие:

Модуль управления рейсами:

- Отвечает за хранение и обработку информации о рейсах (расписание, статус, задержки и т.д.)

- Обеспечивает интерфейсы для планирования, отслеживания и обновления данных о рейсах

- Может взаимодействовать с внешними системами управления воздушным движением

Модуль управления трактористами:

- Отвечает за учет информации о бригадах трактористов (состав, навыки, доступность)

- Предоставляет функции назначения трактористов на обслуживание рейсов

- Может интегрироваться с системами управления персоналом

Модуль оперативного планирования:

- Отвечает за оптимизацию распределения ресурсов (тракторы, бригады) на основе данных о рейсах

- Реализует алгоритмы и модели для эффективного планирования и составления расписаний

- Обеспечивает интерфейсы для взаимодействия с модулями управления рейсами и трактористами

Модуль управления данными:

- Отвечает за хранение и обработку всех критически важных данных (рейсы, тракторы, бригады, расписания и т.д.)

- Предоставляет API для доступа к данным и управления ими

- Обеспечивает механизмы транзакционности, валидации и целостности данных

Модуль пользовательского интерфейса:

- Отвечает за представление данных и взаимодействие пользователей с системой

- Реализует различные представления (дашборды, отчеты, формы и т.д.) для разных ролей пользователей

- Интегрируется с бизнес-логикой, предоставляемой другими модулями

Важно четко определить интерфейсы между этими модулями, чтобы обеспечить возможность их независимого развития, тестирования и развертывания. Использование хорошо определенных API, основанных на принципах ***REST*** или ***GraphQL***, позволит достичь высокой гибкости и расширяемости системы.

6. Масштабирование и интеграция:

Масштабируемость:

- Модульная архитектура, описанная ранее, является основой для обеспечения масштабируемости системы.

- Каждый модуль должен быть спроектирован таким образом, чтобы он мог быть легко масштабирован независимо от других модулей.

- Нужно использовать подходы, которые позволяют масштабировать отдельные компоненты:

- Горизонтальное масштабирование (добавление серверов) для модулей с высокой нагрузкой

- Вертикальное масштабирование (увеличение ресурсов на сервере) для компонентов, требующих больше вычислительной мощности

- Распределение нагрузки с помощью балансировщиков или очередей сообщений

- Использование кеширования, оптимизация запросов к базе данных и других техник для повышения производительности

- Регулярный мониторинг производительности и оптимизация узких мест

Интеграция с другими системами:

- Проектирование открытых API-интерфейсов, обеспечивающих доступ к данным и функциям программы

- Использование стандартных протоколов и форматов данных (***REST API, GraphQL, XML, JSON***) для интеграции

- Создание адаптеров или шлюзов, которые обеспечивают взаимодействие с различными системами аэропорта

- Реализация механизмов аутентификации, авторизации и безопасности при интеграции с внешними системами

- Обеспечение надежности и отказоустойчивости при взаимодействии с другими приложениями

Масштабируемость и расширяемость API-интерфейсов:

- Разработка ***API***-интерфейсов с учетом возможного расширения функциональности в будущем

- Использование модульного и версионного подхода к ***API***, чтобы избежать нарушения обратной совместимости

- Внедрение механизмов управления версиями ***API***, чтобы обеспечить плавный переход между версиями

- Реализация системы мониторинга и аналитики использования ***API*** для эффективного управления

Таким образом, ключевыми аспектами будут:

- Модульность и независимое масштабирование компонентов

- Использование подходящих технологий для распределения нагрузки и повышения производительности

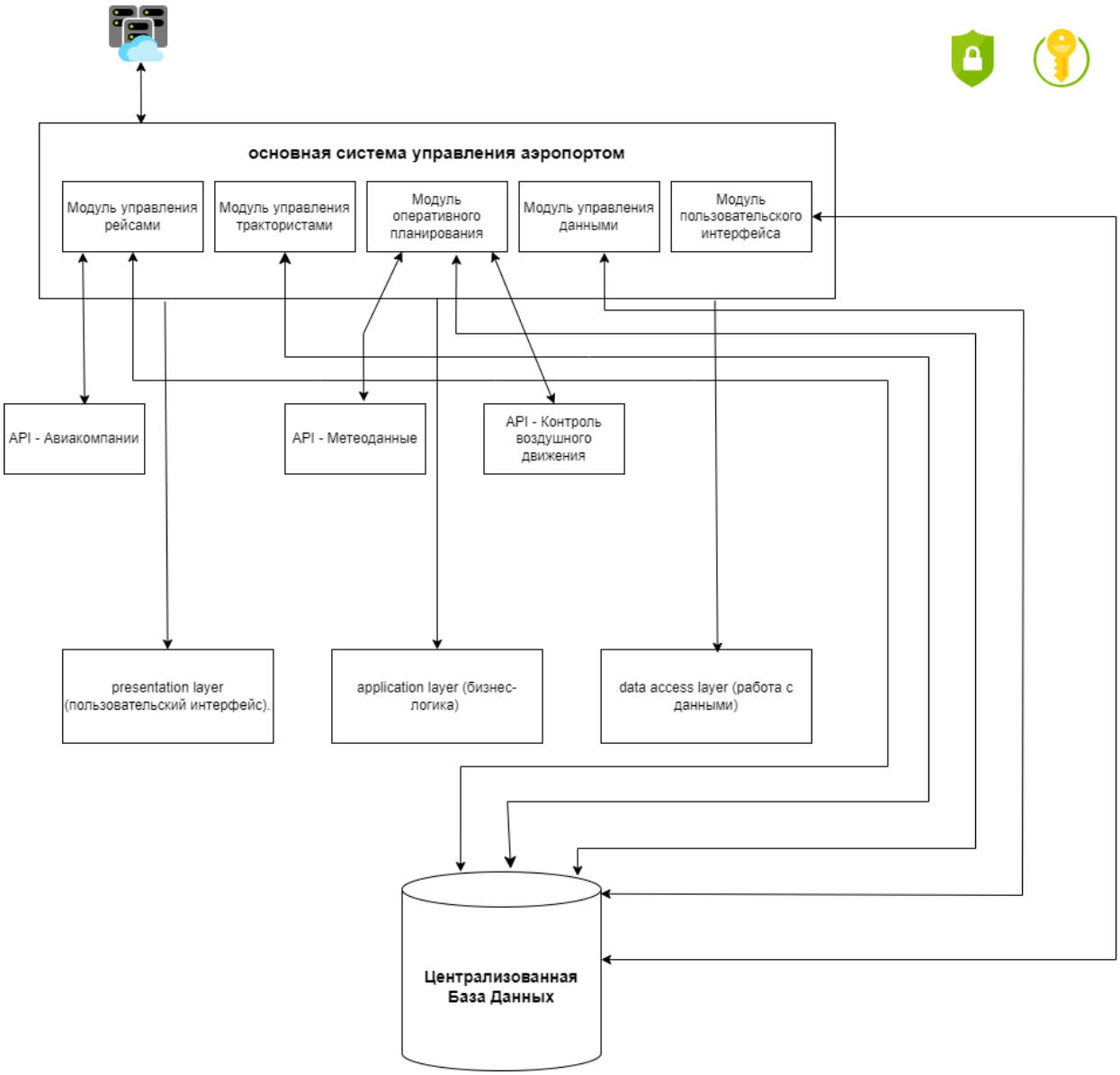
- Проектирование открытых, стандартизированных API-интерфейсов

- Обеспечение надежности и безопасности интеграционных механизмов

Реализация этих перспективных направлений развития позволит сделать программу еще более функциональной, удобной, эффективной и масштабируемой, обеспечивая ее долгосрочную востребованность и конкурентоспособность в сфере управления ресурсами аэропортов.

Заключение:

На основе всех выше предложенных улучшений, можно представить общую архитектуру программы в виде следующей схемы:



Ключевые элементы этой архитектуры:

1. Модульная структура:

- Программа разделена на логические модули, отвечающие за управление рейсами, трактористами, оперативное планирование и т.д.

- Четко определены интерфейсы между модулями для обеспечения гибкости и расширяемости.

2. Многослойная архитектура:

- Выделены слои ***presentation*** (пользовательский интерфейс), ***application*** (бизнес-логика) и ***data access*** (работа с данными).

- Используются паттерны проектирования, такие как ***MVC*** или ***MVP***, для разделения ответственностей.

3. Управление данными:

- Централизованная база данных для хранения информации о рейсах, бригадах, трактористах и других важных сущностях.

- Механизмы обеспечения целостности, транзакционности и валидации данных.

4. Интеграция с внешними системами:

- Реализованы API-интерфейсы для взаимодействия с другими информационными системами аэропорта.

- Используются стандартные протоколы и форматы обмена данными (REST API, XML, JSON).

5. Масштабируемость и отказоустойчивость:

- Модульная архитектура позволяет масштабировать отдельные компоненты независимо.

- Применены техники для обеспечения высокой производительности и надежности (кеширование, распределенные вычисления, отказоустойчивость).

6. Безопасность и управление доступом:

- Реализованы механизмы аутентификации, авторизации и ролевого управления доступом.

- Внедрены средства шифрования, аудита и мониторинга безопасности.

7. Развертывание и эксплуатация:

- Автоматизирован процесс развертывания и обновления системы.

- Интеграция с системами мониторинга производительности, логирования и устранения неполадок.

Эта архитектура позволит создать надежную, масштабируемую и гибкую программу управления аэропортом, способную эффективно обрабатывать большие объемы данных и интегрироваться с другими системами аэропорта.

P.S.

Конечно все вышеизложенные улучшения требуют более глубоких и разносторонних знаний в ***IT***-сфере, и непременно слаженную команду специалистов, а также это займёт не мало времени. Но установив свою программу диспетчерам в первоначальном виде, они уже сказали огромное спасибо за то, что уже не надо будем вести учет бригад и трактористов на листике вручную.

12. Список литературы и используемых инструментов.

1. "Архитектура программного обеспечения аэропортов" - статья в журнале "Авиационные технологии", 2020 год.

2. "Принципы проектирования масштабируемых систем управления аэропортами" - книга Джона Смита, 2018 год.

3. "Обеспечение кибербезопасности критически важных систем аэропортов" - статья на портале "Безопасность информационных систем", 2021 год.

4. "Руководство ***IATA*** по интеграции информационных систем аэропортов", 2019 год.

5. "Лучшие практики масштабирования и отказоустойчивости в системах управления аэропортами" - отчет консалтинговой компании "***AirportSolutions***", 2020 год.

6. ***draw.io*** для создания схемы архитектуры программы.

7. ***Visual Studio Code*** для написания программы.