Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»

**Департамент анализа данных,
принятия решений и финансовых технологий**

Курсовая работа

на тему:

Сервис интернет календарей для расписания университетских занятий

Выполнил:

Студент группы ПИ18-2

Кирилкин Владимир Алексеевич

_____

Научный руководитель:

к.э.н., доцент

Макрушин Сергей Вячеславович

_____

2020

Оглавление

Актуальность задачи

Всё запомнить невозможно, поэтому людям часто приходится где-то искать информацию. В случае с расписанием университетских занятий – искать надо на официальном сайте. Но не всегда это бывает удобным, например, когда пара через несколько минут и срочно нужно узнать кабинет, а интернет с мобильных устройств не самый быстрый. Тогда можно воспользоваться расписанием в мобильном приложении (если таковое имеется) или популярными в последнее время чат-ботами в мессенджерах.

Но если вообще нет мобильного интернета? Может помочь экспорт расписания в календарь. Но в случае изменения какой-либо информации о парах расписание становится не актуально. Именно тут и пригождается сервис интернет календарей, который будет самостоятельно обновляться с определенной периодичностью.

Также к неоспоримым преимуществам интернет календарей нужно отнести интеграцию с другими сервисами и устройствами.

Приведу сравнительную таблицу различных способов доставки расписания

	Веб сайт	**Мобильное приложение**	**Интернет календарь**	**Чат бот в мессенджере**
Требуется Интернет-соединение	В момент использования	Только для обновления	Только для обновления	В момент запроса расписания результат сохраняется
Удобство использования «на ходу»	Сомнительно	Удобно	Удобно	Удобно, но только на ближайшее время
Актуальность информации	Актуальна	Актуальна	Зависит от периода обновления	Актуальна
Интеграция в другие приложения	Нет	Нет	Да	Нет
Поддержка на различных устройствах	Где есть браузер	Мобильные устройства	Мобильные устройства, персональные компьютеры, носимые устройства (смарт часы)	Где работает конкретный мессенджер
Удобство настройки	Удобно	Удобно	В зависимости от устройства могут быть сложности	Удобно

Как мы можем видеть из таблицы, главный недостаток интернет календарей – период обновления. На устройствах с операционной системой Android при использовании Google аккаунта период обновления составляет около 12 часов. Это может вызвать некоторые проблемы, но расписание редко меняют прямо перед парой.

Таким образом, использование интернет календарей выглядит довольно привлекательным, особенно учитывая интеграцию в системный календарь, где можно будет легко планировать другие мероприятия исходя из занятости.

Описание предметной области

Сервис интернет календарей представляет из себя веб сервер, работающий по протоколам HTTP/HTTPS или WebDAV. Формат ответа сервера – файлы расширения ".ical", соответствующие спецификации iCalendar (RFC 5545) [1]. За информацией о расписании сервис использует публичное REST API «Галактика: Расписание учебных занятий», который используется в Финансовом Университете как единственный источник расписания.

Используемые технологии

python 3.8

- asyncio – библиотека для написания асинхронного (конкурентного) кода
- aiohttp – библиотека для асинхронного http сервера/клиента
- aiomisc – библиотека для удобного управления асинхронными сервисами
- aioredis – библиотека для асинхронного подключения к NoSQL базе данных Redis
- icalendar – библиотека для формирования файла календаря, придерживающегося спецификации iCalendar
- marshmallow – библиотека для сериализации/десериализации данных
- ujson – библиотека для работы с json форматом

Redis – NoSQL база данных

Docker – система контейнеризации

- docker-compose – инструмент для оркестрации Docker контейнеров

GitHub Actions – система автоматизации рабочих процессов

Описание работы кода

При написании сервиса я использовал асинхронное (конкурентное) программирование. Суть в том, что функции дробятся на более мелкие подфункции, по ключевым словам async/await, которые в основном используются перед системными вызовами. В этих местах во время выполнения программы после совершения системного вызова выполнение будет остановлено в ожидании ответа от системы и переключено на другую подфункцию. Когда ответ от системного вызова придет, программа продолжится с того же места, где этот вызов был совершен.

Асинхронное программирование использует только один поток, что способствует простому горизонтальному масштабированию в случае недостатка вычислительных мощностей одного процесса.

Асинхронное программирование актуально для ограниченного количества задач – IO-bound, то есть направленных на ввод/вывод, а не на процессорные вычисления. Данную программу можно

[1] iCalendar (RFC 5545) - https://icalendar.org/RFC-Specifications/iCalendar-RFC-5545/

классифицировать как IO-bound, так как основные действия – http запросы, как отправка, так и обработка, а также запросы к базе данных.

Если бы в данном проекте использовалось бы параллельное программирование, то каждый поток бы существовал всё время от запроса до ответа, хотя время ожидания ответа системного вызова было бы больше 99% всего время существования потока.

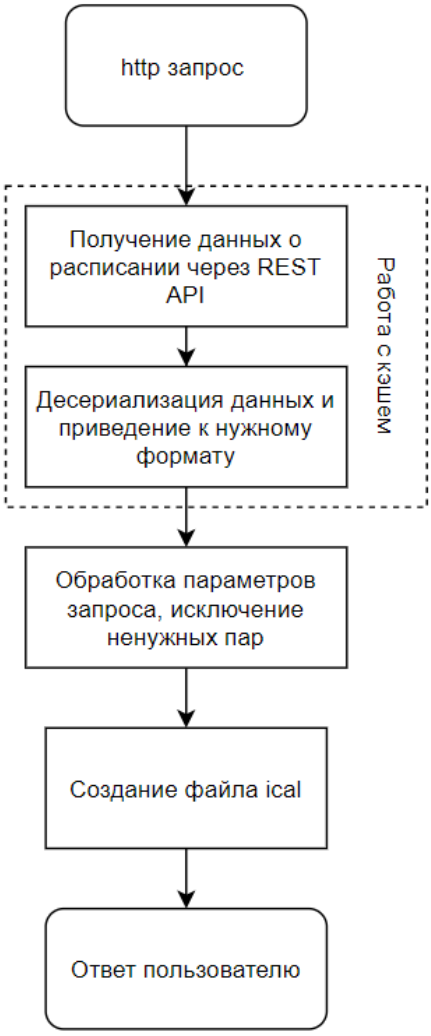
Точкой входа в приложение является файл __main__.py. В нем обрабатываются переданные аргументы, такие как адрес и порт для открытия TCP сокета или тип кеширования; создается event loop (цикл событий асинхронного приложения), куда будут попадать все подзадачи и откуда будут доставаться при получении ответа системного вызова. В этот event loop сразу же попадает инициализация aiohttp веб сервиса, который будет принимать http запросы и отвечать на них.

Сам сервис CalendarService находится в файле services.py. В нем инициализируется указанный тип кеширования и создаются http маршруты с их обработчиками.

В приложении представлен единственный обработчик CalendarView, в котором выполняется валидация запроса, обработка параметров запроса, работа с кэшем, а также обращение к функциям api и генерации календаря.

За генерацию файла календаря отвечает функция create_calendar из файла calendar_creator.py.

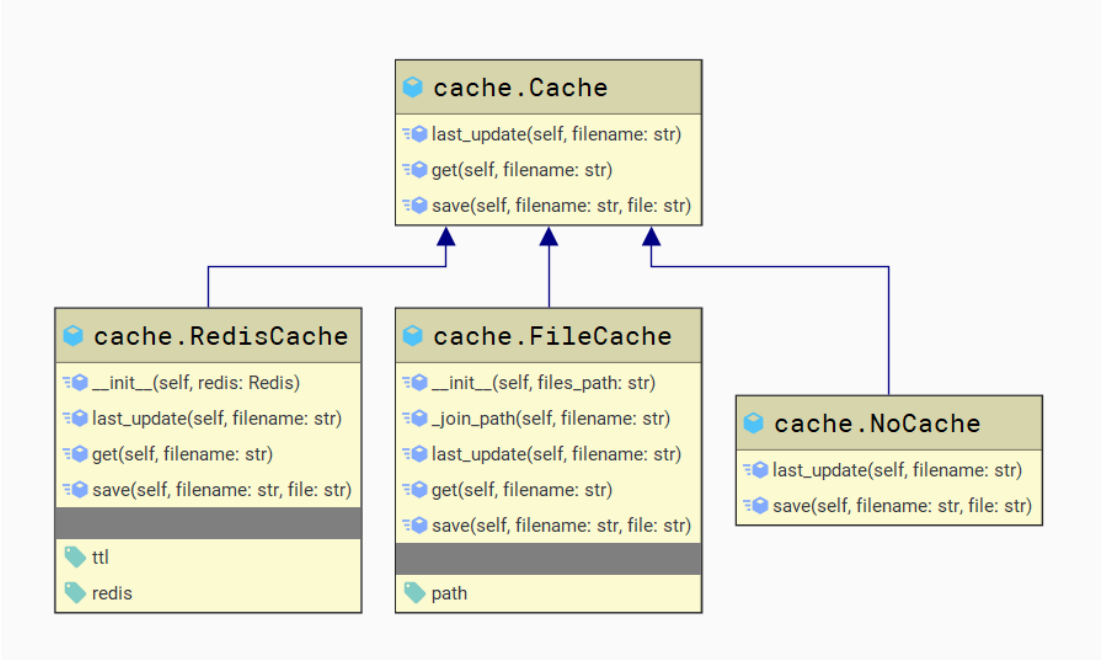
Ниже приведена краткая диаграмма работы сервиса



Кеширование

Для кеширования я решил использовать Redis. Это нереляционная высокопроизводительная NoSQL СУБД, работающая по принципу “key-value” (ключ-значение), благодаря чему её производительность в разы выше традиционных реляционных баз данных.

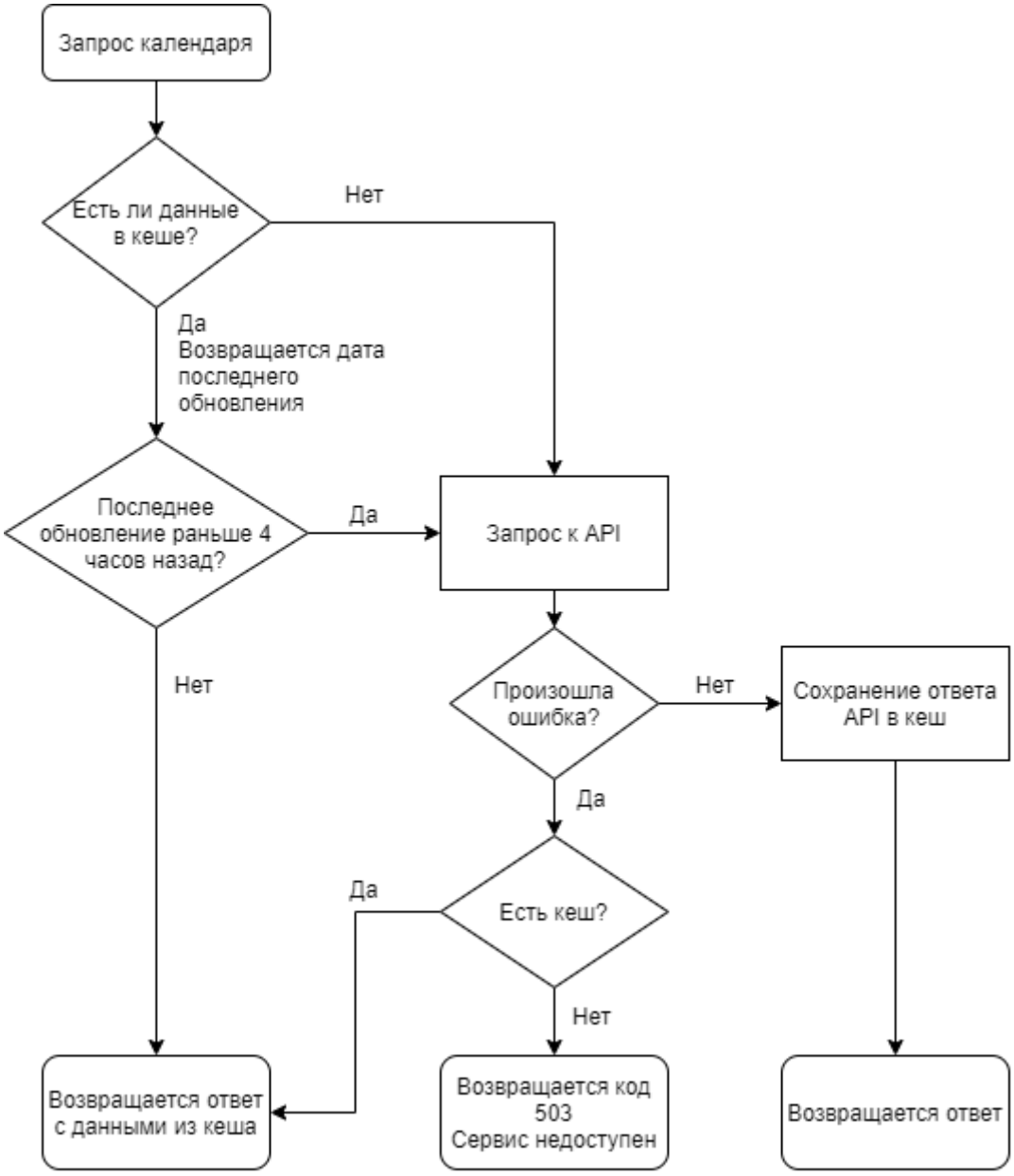
Однако в моем приложении предусмотрены и другие варианты кеширования, выбор осуществляется через переменную среды CACHE_TYPE. Помимо Redis можно выбрать хранение кэша как файлов на диске или вообще отключить кеширование.



В кэш попадают приведенные к нужному виду ответы от API.

В случае с Redis в кэш передаются данные с параметром TTL (time to live – время жизни) 172800 секунд (или двое суток). По истечению этого времени запись будет автоматически удалена как устаревшая.

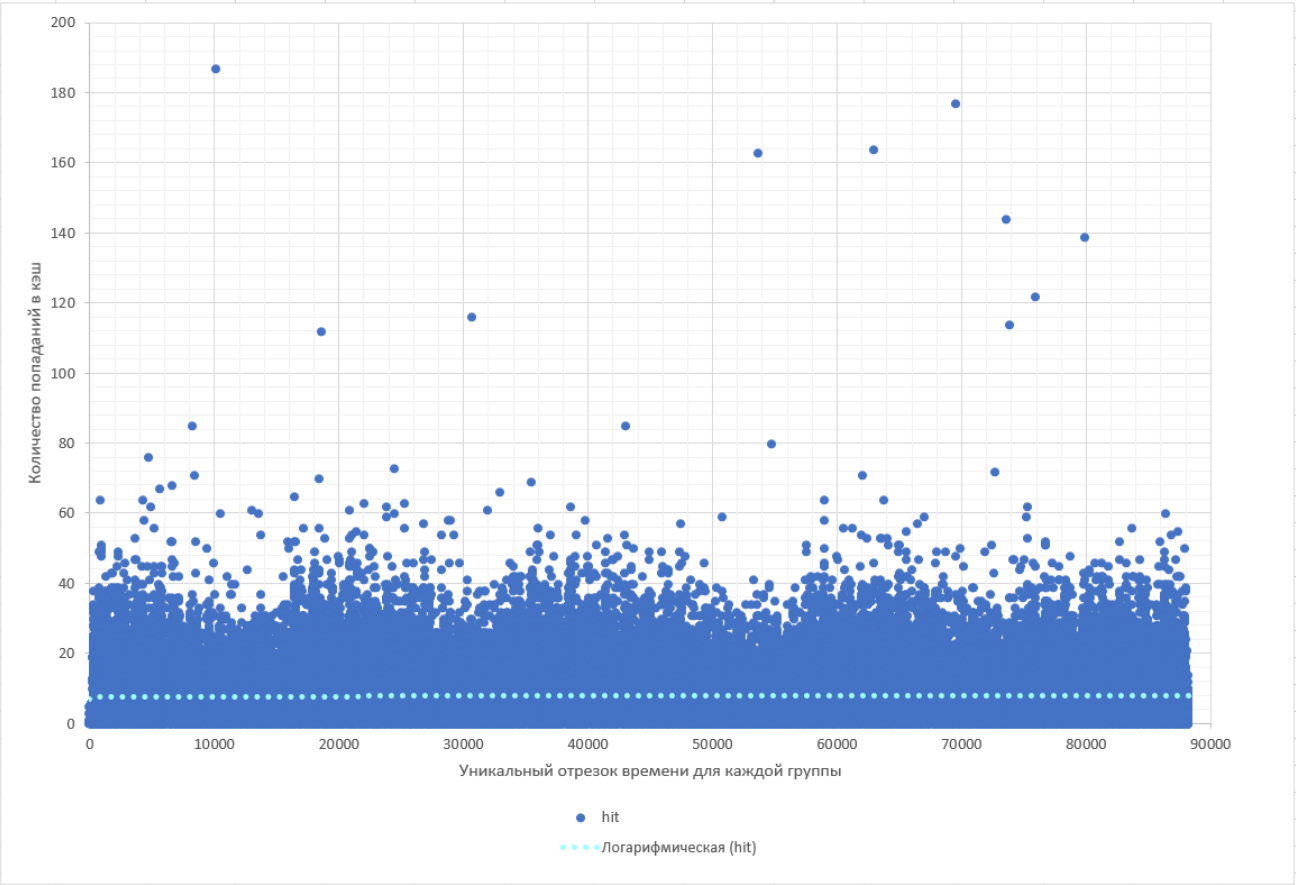
Время последнего обновления для Redis высчитывается из TTL, для кеширования в локальную память из свойств хранимого файла.



Обращение к кэшу в коде описано через интерфейс (абстрактный класс), поэтому для добавления нового вида кеширования нужно будет сделать наследование от базового класса Cache и имплементировать все методы родительского класса.

Обоснованность кеширования

Для того чтобы понять, насколько необходимо кеширование данному сервису, построим точечную диаграмму по запросам к сервису за последний месяц. По оси x уникальный отрезок времени размером в 4 часа (константа времени кеширования) для каждой группы или преподавателя, по оси y количество запросов в этот отрезок времени



Среднее попадание в кэш – 7,783 раз. Цифра не большая, но за месяц это выходит в 253,8 Гб сэкономленного трафика внутри сети.

Автоматизация рабочих процессов

Автоматизация рабочих процессов помогает перенести часть задач с разработчика на отдельный сервис. К рабочим процессам относятся:

- Непрерывная интеграция (сборка, тестирование и развертывание ПО)
- Публикация python-модулей или docker образов
- Оповещения

На рынке присутствует множество различных продуктов, но я выбрал GitHub Actions[2], так как GitHub является крупнейшей платформой для хранения удаленных git репозиториев, и большинство моих проектов находятся именно там.

В моём проекте используется два сценария, оба срабатывают на отправку изменений в удаленный репозиторий

[2] GitHub Actions - https://github.com/features/actions

Первый сценарий – автоматической проверки работы кода – unit-тесты

Второй сценарий – сборка Docker образа и публикация в Docker репозитории docker.pkg.github.com, который тоже относится к инфраструктуре GitHub.

Сценарии для GitHub Actions расположены в папке .github/workflows/, а сценарии сборки контейнеров в Dockerfile.

Разворачивание сервиса на удаленном сервере

Для того чтобы развернуть сервис на удаленном сервере с предварительно установленными программами Docker и docker-compose нужно создать файл следующего содержания с именем docker-compose.yml

```yaml
version: '3.1'

services:
  app:
    image: docker.pkg.github.com/v1ack/fu-calendar/calendar  # Адрес Docker хранилища
    expose:  # Открытый порт
      - 4040
    ports:
      - "4040:4040"
    environment:  # Переменные окружения
      APP_API_ADDRESS: 0.0.0.0
      API_FILES_FOLDER: /mnt/files
      CACHE_TYPE: redis  # Тип кеша (redis, file или no)
      REDIS_URL: redis://redis/0  # Адрес Redis

  redis:
    image: redis  # Образ
    command: ["redis-server", "--appendonly", "yes"]  # Команда запуска
    expose:  # Порт, если нужен для доступа извне
      - 6379
    ports:
      - 6379:6379
```

И запустить его командой docker-compose up

Сервис будет доступен по адресу 127.0.0.1:4040/calendar/<type>/<id>

Где <type> тип расписания – group для группы, lecturer для преподавателя

<id> - идентификатор группы или преподавателя из РУЗ

Статистика по пользователям

Так как сервис развернут и имеет аудиторию, то мы можем работать с реальными цифрами. Для анализа возьмем логи прокси сервера nginx, через которого проходят все запросы к сервису.

Анализировать будем период с 19 февраля по 20 марта. Цифры по cache-hit приведены по этому же периоду.

Так как календарь не индивидуальный, а на группу, а из способов идентификации пользователей только useragent клиента календаря, то мы можем проанализировать нижнюю границу количества устройств, запросивших календарь. В одной группе может быть несколько человек, использующих календарь через синхронизацию Google аккаунта, поэтому useragent у всех таких запросов будет один. В iOS в useragent содержится только версия ОС, поэтому тоже точных данных быть не может.

Уникальных пар (id, useragent)	4679
Из них преподавателей	111
Уникальных групп	592
Уникальных преподавателей	47

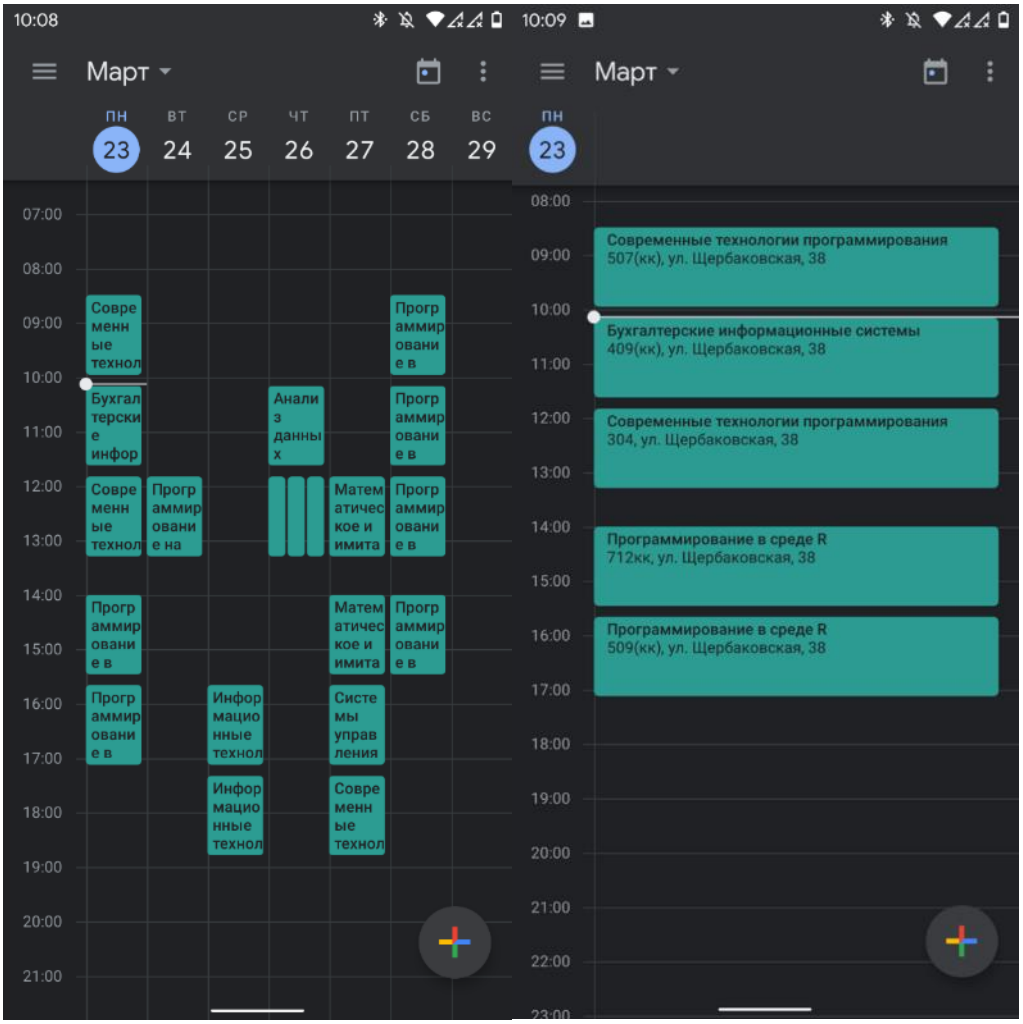
За этот период было отдано 169.2 Гб файлов ical

Источники

1. Школа бэкенд разработки Яндекса – 2019: https://habr.com/ru/company/yandex/blog/498856/
2. Asynchronous HTTP Client/Server for asyncio and Python: https://docs.aiohttp.org/en/stable/
3. Docker: https://docker.com/get-started
4. GitHub Actions - https://github.com/features/actions

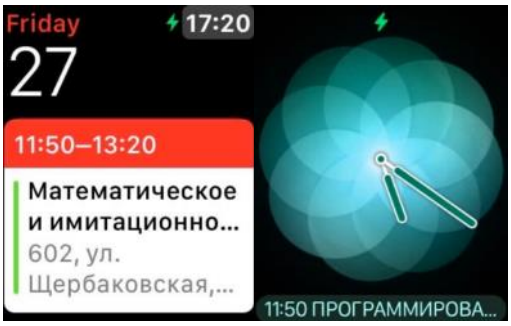
5. Гриднев Д.В., Иванов М.Н., Кирилкин В.А. Разработка ботов ВКонтакте и телеграм для расписания университета // XIII Международная научно-практическая конференция «Новые информационные технологии в образовании и науке НИТО 2020»

Приложение. Результат на конечных устройствах

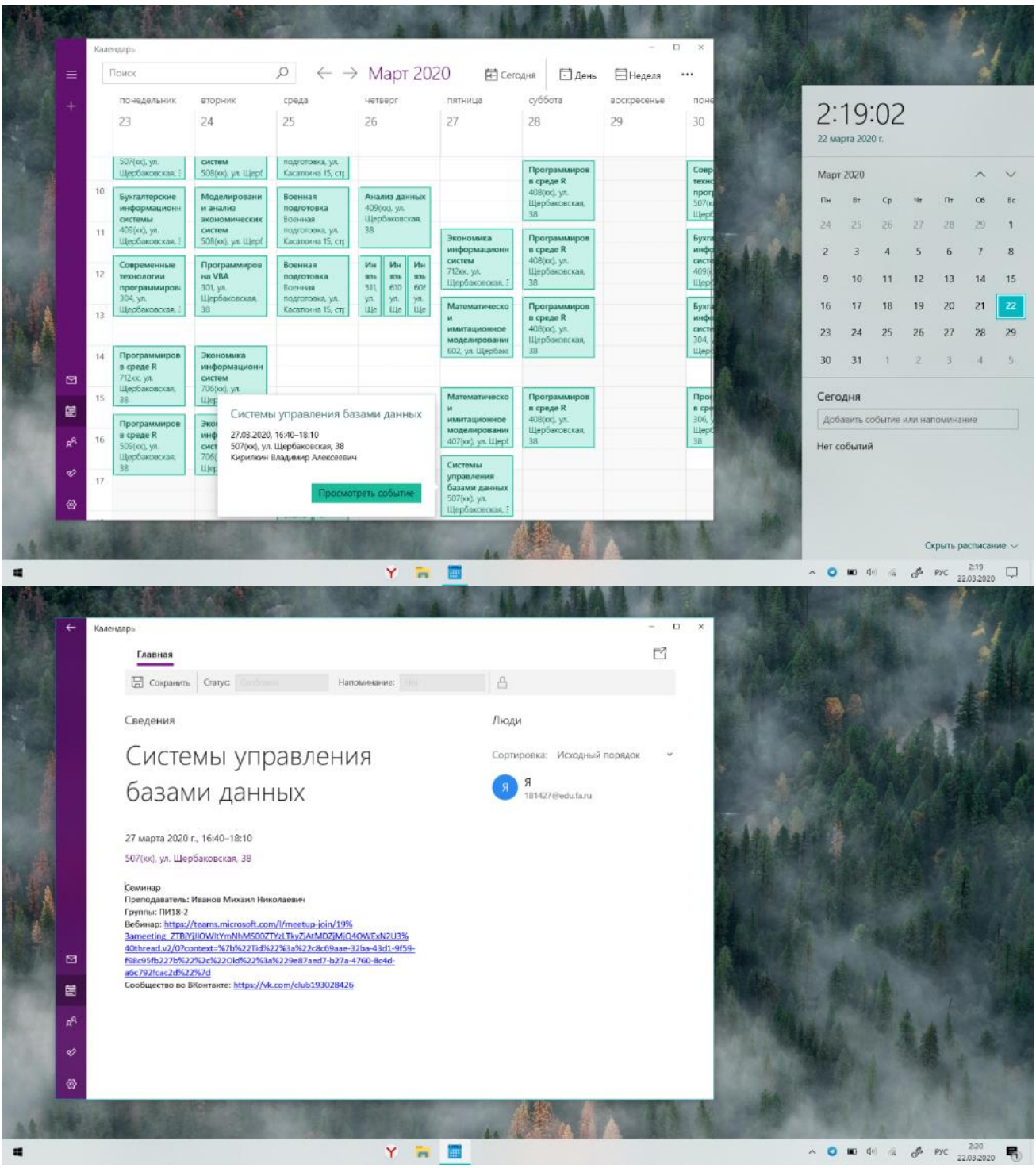


Пример работы с приложением Календарь на Android 10

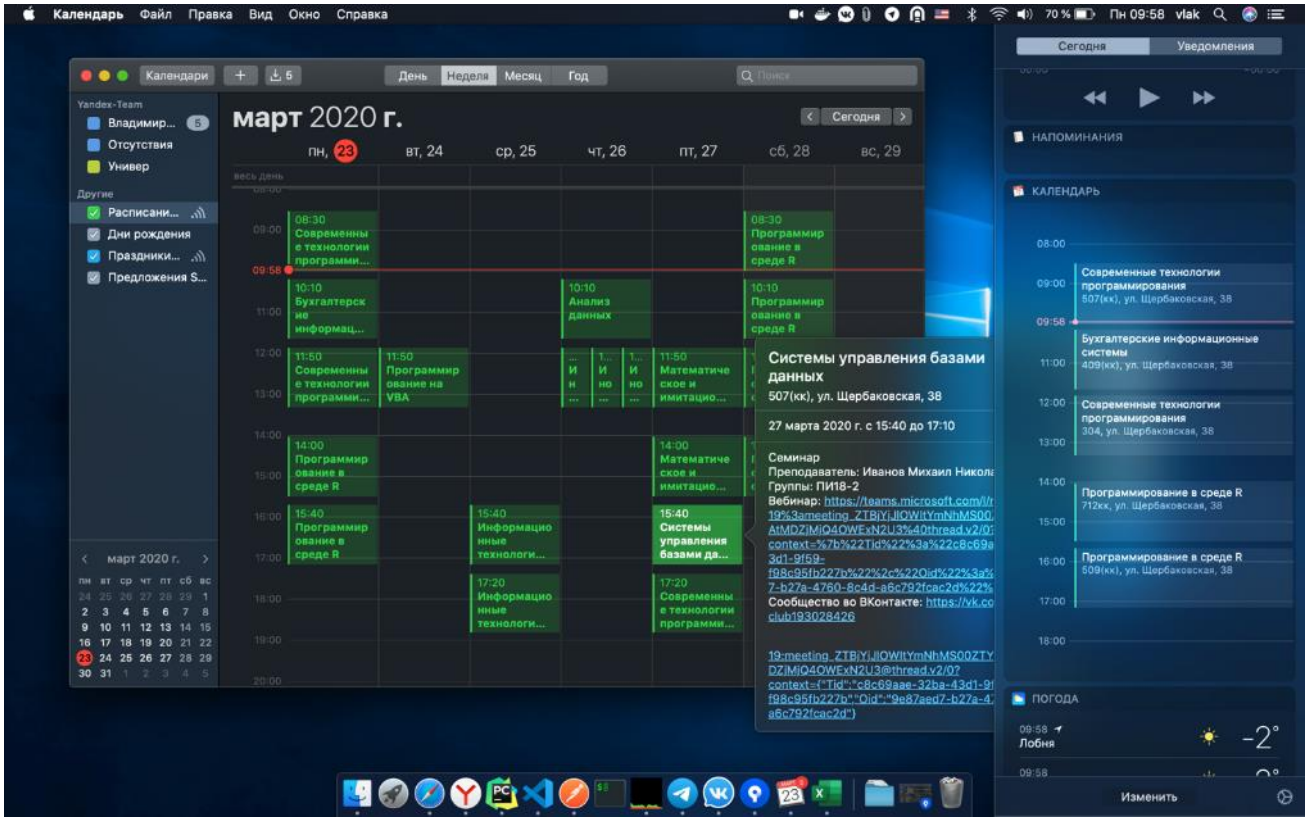
Пример работы на часах Galaxy Watch



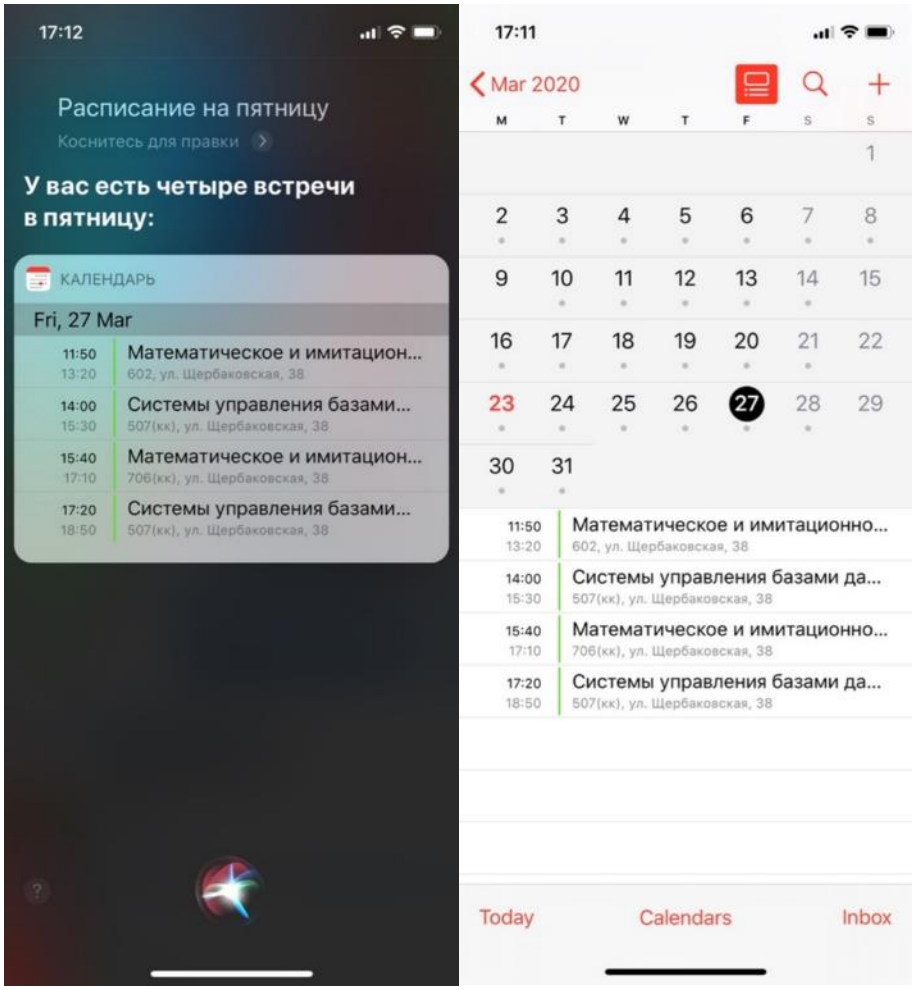
Пример работы на часах Apple Watch



Пример работы с приложением Календарь на Windows 10



Пример работы с приложением Календарь на macOS X



Пример работы с приложением Календарь на iOS 13