Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Программирование»

на тему «Разработка программы сложной структуры методом нисходящего программирования. Интеграция "Google Календарь" с Яндекс.Алиса»

Выполнил: ст. гр. 25ВВВ4
Колос М. Д.
Принял:
Кандидат т.н. Токарев А. Н.

20.05.2026

Пенза 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

_____ М.А. Митрохин

«____»_____20__г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Колосу Максиму Дмитриевичу Группа 25ВВВ4

Тема проекта: ***Разработка программы сложной структуры методом нисходящего программирования.***

Исходные данные (технические требования) на проектирование

"Интеграция Google календаря с Яндекс.Алиса"

1. Создание вебхука для серверов Яндекса.
2. Проверка действительности запроса.
3. Работа с iCalendar стандартом.
4. Возможность узнать расписание на день
5. Сохранение календаря в системе.

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Поддержка базовых CRUD-операций.
4. Контекст и многошаговые диалоги.
5. Поддержка множества пользователей.
6. Обработка ошибок и пограничных случаев.
7. Операционная система Linux.
8. Язык программирования – Go и Assembler.
9. Среда разработки ПО – Microsoft Visual Studio Code.
10. Размещение в каталоге навыков Алисы.

Объем работы по курсу

1. **Расчетная часть**

Разработка программы.

2. **Графическая часть**

Схема данных, схема ресурсов системы, схема работы системы, иерархическая структура программы, схема взаимодействия программы.

3. **Экспериментальная часть**

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания "18" февраля

Дата защиты проекта "___" _____

Руководитель _____

Задание получил "25" февраля 2026 г.

Студент Колос Максим Дмитриевич

Содержание

Введение 7

Постановка задачи 8

Выбор решения 9

Описание разработки программы 10

Отладка и тестирование 11

Описание программы 12

Руководство пользователя 18

Заключение 21

Список используемых источников 22

Приложение А. Структура и листинги программы 23

- Фрагмент конфигурации 23
- Фрагмент HTTP-сервера 23
- Основной выбор сценария навыка 23
- Внутренние структуры календаря 24
- skill.go 24

Приложение Б. Примеры работы программы 35

- Чат с Алисой 35

Введение

Современные голосовые помощники всё чаще используются не только для поиска информации, но и для выполнения практических задач: просмотра расписания, создания напоминаний, работы с календарями и управления повседневными делами. Такое приложение должно принимать естественную речь пользователя, корректно определять намерение, взаимодействовать с внешними сервисами и возвращать короткий понятный ответ.

В данной курсовой работе рассматривается разработка серверного приложения calendar2alice на языке Go. Программа реализует навык для Алисы, который позволяет связать аккаунт пользователя с Google Календарём, узнать расписание на выбранный день и создать новое событие голосовой командой.

Особенностью проекта является его модульная серверная структура. Отдельные пакеты отвечают за протокол Алисы, HTTP-сервер, OAuth-авторизацию Google, хранение данных в SQLite, обращение к Google Calendar API, обработку пользовательских фраз и работу с нейросетевым разбором команд. Такой подход соответствует методу нисходящего программирования: сначала определяется общий сценарий работы навыка, затем он разбивается на самостоятельные подсистемы.

Приложение написано на Go, использует стандартный пакет net/http, библиотеку OAuth2, официальный клиент Google API и встроенную базу данных SQLite через modernc.org/sqlite. Для развёртывания предусмотрены Dockerfile и docker-compose.yml, что позволяет запускать сервис на сервере и публиковать webhook для Алисы через HTTPS.

Постановка задачи

Необходимо разработать серверное приложение calendar2alice, предназначенное для обработки webhook-запросов навыка Алисы и взаимодействия с Google Календарём пользователя.

Программа должна принимать JSON-запросы от платформы Алисы по маршруту /alice, выделять идентификатор пользователя, анализировать исходную фразу и выбирать один из основных сценариев: привязка календаря, просмотр расписания или создание события.

Для привязки календаря необходимо реализовать OAuth 2.0: выдачу временной ссылки, обработку перехода пользователя в браузере, получение токена Google и сохранение токена в локальном хранилище. После привязки пользователь должен иметь возможность обращаться к своему календарю без повторного входа.

Для просмотра расписания программа должна определять дату из пользовательской фразы, запрашивать события Google Календаря за выбранные сутки, форматировать результат на русском языке и возвращать его в структуре ответа Алисы.

Для создания события необходимо разобрать дату, время, длительность и название события. При наличии пересечений с уже существующими событиями программа должна предупредить пользователя. В случае успешного создания события пользователь получает подтверждение.

Также требуется предусмотреть конфигурацию через переменные окружения, хранение служебных данных в SQLite, обработку ошибок внешних API, модульные тесты для ключевых функций и контейнеризацию приложения для развёртывания.

Выбор решения

В качестве языка реализации выбран Go. Он хорошо подходит для серверных приложений: имеет быструю компиляцию, строгую типизацию, развитую стандартную библиотеку для HTTP-сервисов, удобную модель конкурентного выполнения и простое развёртывание в виде одного бинарного файла.

Архитектура приложения построена вокруг HTTP-сервера. Маршрут /healthz используется для проверки работоспособности, /alice принимает запросы от Алисы, /oauth/google и /oauth/google/callback обслуживают процесс OAuth-авторизации Google. Такой набор маршрутов отделяет пользовательский голосовой сценарий от служебной авторизации.

Для хранения данных используется SQLite. В базе сохраняются признаки первого посещения навыка, временные токены привязки, OAuth state и JSON-представление токена Google. SQLite выбран потому, что не требует отдельного сервера базы данных и подходит для небольшого сервиса с персональными настройками пользователей.

Интеграция с Google Календарём выполнена через Google Calendar API. В проекте определён собственный интерфейс calendar.Service, поэтому обработчик навыка зависит не от конкретной реализации Google API, а от абстракции. Это упрощает тестирование и делает код менее связанным.

Для распознавания сложных пользовательских формулировок используется нейросетевой клиент Yandex Responses API. Если нейросетевой разбор недоступен или возвращает некорректный JSON, программа переходит к локальным правилам разбора команд. Такой резервный сценарий повышает устойчивость приложения.

Проект разворачивается в Docker. Контейнер приложения собирается из исходного кода Go, а docker-compose.yml дополнительно описывает сервис Caddy для публикации HTTPS-адреса, необходимого для webhook Алисы и OAuth callback.

Описание разработки программы

Разработка выполнялась поэтапно. Сначала была определена общая схема работы: Алиса отправляет запрос на сервер, сервер разбирает команду, при необходимости обращается к Google Calendar API и возвращает JSON-ответ. Затем были выделены пакеты по зонам ответственности.

Структура проекта calendar2alice включает пакеты internal/alice, internal/server, internal/handler, internal/calendar, internal/auth, internal/store, internal/config, internal/googlehttp и internal/neural. Файлы Dockerfile и docker-compose.yml отвечают за сборку и развёртывание.

Центральным объектом является handler.Skill. Он получает запрос Алисы, определяет пользователя, отмечает первый визит, извлекает текст команды, запускает нейросетевой или локальный разбор и вызывает соответствующий обработчик: answerLink, answerSchedule или answerCreate.

Для работы с датами применяются функции из пакета time. Программа учитывает часовой пояс пользователя, полученный от Алисы, либо использует значение DEFAULT_TIMEZONE. В проекте отдельно обработан случай, когда Алиса передаёт UTC как техническую заглушку, чтобы расписание не смещалось относительно московского времени.

При создании событий используются два уровня разбора. Сначала программа пытается применить результат нейросетевой модели, где намерение и дата представлены в JSON. Если такой результат отсутствует, используются регулярные выражения и словари русских слов времени.

Отдельное внимание уделено обработке конфликтов. Перед созданием события приложение запрашивает существующие события за нужный интервал и проверяет пересечения временных отрезков. При наличии конфликта ответ содержит предупреждение.

Отладка и тестирование

Тестирование проекта выполняется стандартной командой Go: go test ./... . В репозитории присутствуют модульные тесты для пакетов handler, store и neural. Они проверяют критические участки логики, не требующие реального подключения к Алисе или Google.

В пакете handler тестируются разбор дат и диапазонов расписания, определение относительных дней, склонение слова «событие», обработка конфликтов, создание EventInput из пользовательской фразы и JSON-ответа нейросетевой модели.

В пакете store тестируется создание и поглощение временных токенов, сохранение и чтение OAuth-токена Google, сохранение refresh token при обновлениях, очистка токена и отметка первого посещения навыка.

В пакете neural проверяется извлечение текста из ответа Yandex Responses API. Поскольку формат ответа внешнего API может быть вложенным, в коде реализован поиск текстового содержимого в разных вариантах структуры JSON.

При отладке HTTP-части используется маршрут /healthz, который возвращает строку ok. Для проверки webhook можно отправлять POST-запросы на /alice с JSON, соответствующим структуре запроса Алисы.

Описание программы

Пакет config загружает параметры из .env и переменных окружения. Основные настройки: HTTP_ADDR, DATABASE_PATH, PUBLIC_BASE_URL, DEFAULT_TIMEZONE, параметры OAuth Google, идентификатор календаря, ключи нейросетевого API и тайм-аут NEURAL_DEADLINE.

Пакет server создаёт ServeMux и регистрирует маршруты. Для /alice разрешён только POST. Тело запроса декодируется в alice.Incoming, после чего управление передаётся объекту Skill. Ответ кодируется в JSON с Content-Type application/json; charset=utf-8.

Пакет alice содержит структуры Incoming, Outgoing, RequestPayload, NLU, Entity, Intent и Button. Функции Reply и ReplyWithButtons формируют ответы с текстом, TTS и кнопками.

Пакет store открывает SQLite-базу, выполняет миграции и предоставляет методы для работы с пользователями и токенами. Все операции принимают context.Context, что позволяет отменять запросы при завершении HTTP-запроса.

Пакет auth создаёт OAuth-конфигурацию, генерирует ссылку входа и обрабатывает callback. Пакет calendar преобразует события Google Calendar API во внутреннюю структуру Event, поддерживает чтение событий за интервал и создание новых событий.

Пакет handler объединяет остальные части. Функции wantsSchedule, wantsCreate и wantsLink выполняют быстрый локальный выбор сценария по ключевым словам. Более сложный разбор переносится в nlu_apply.go и create_event.go. Для ускоренного преобразования строк в числа используется отдельная реализация atoi, включая вариант на ассемблере для amd64.

На рисунке 1 представлена схема взаимодействия программы.

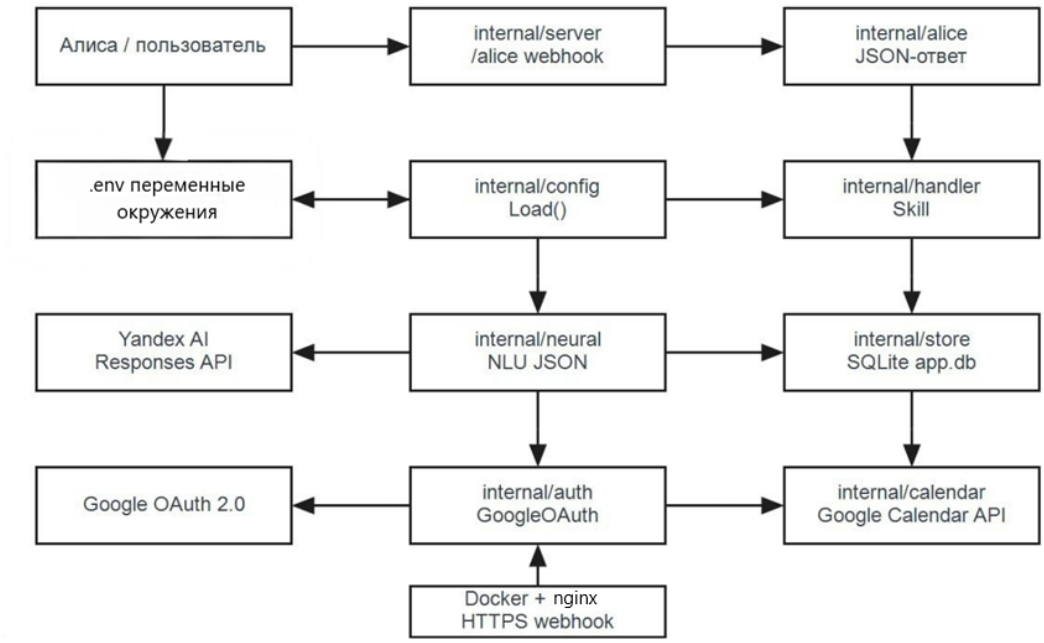


Рисунок 1 - Схема взаимодействия программы

Блок-схема метода Handle представлена на рисунке 2, рисунке 3, рисунке 4.

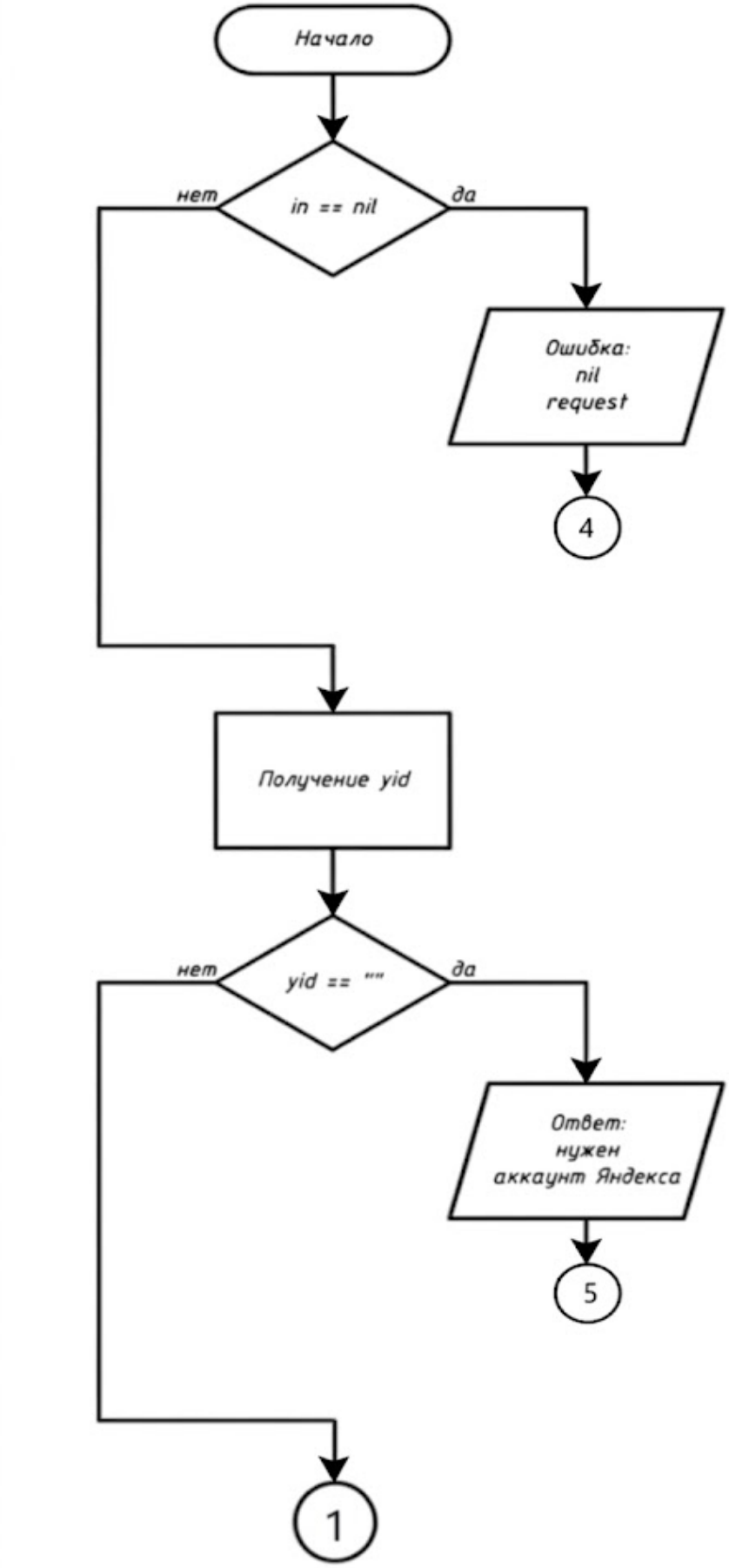


Рисунок 2 - Блок-схема метода Handle()

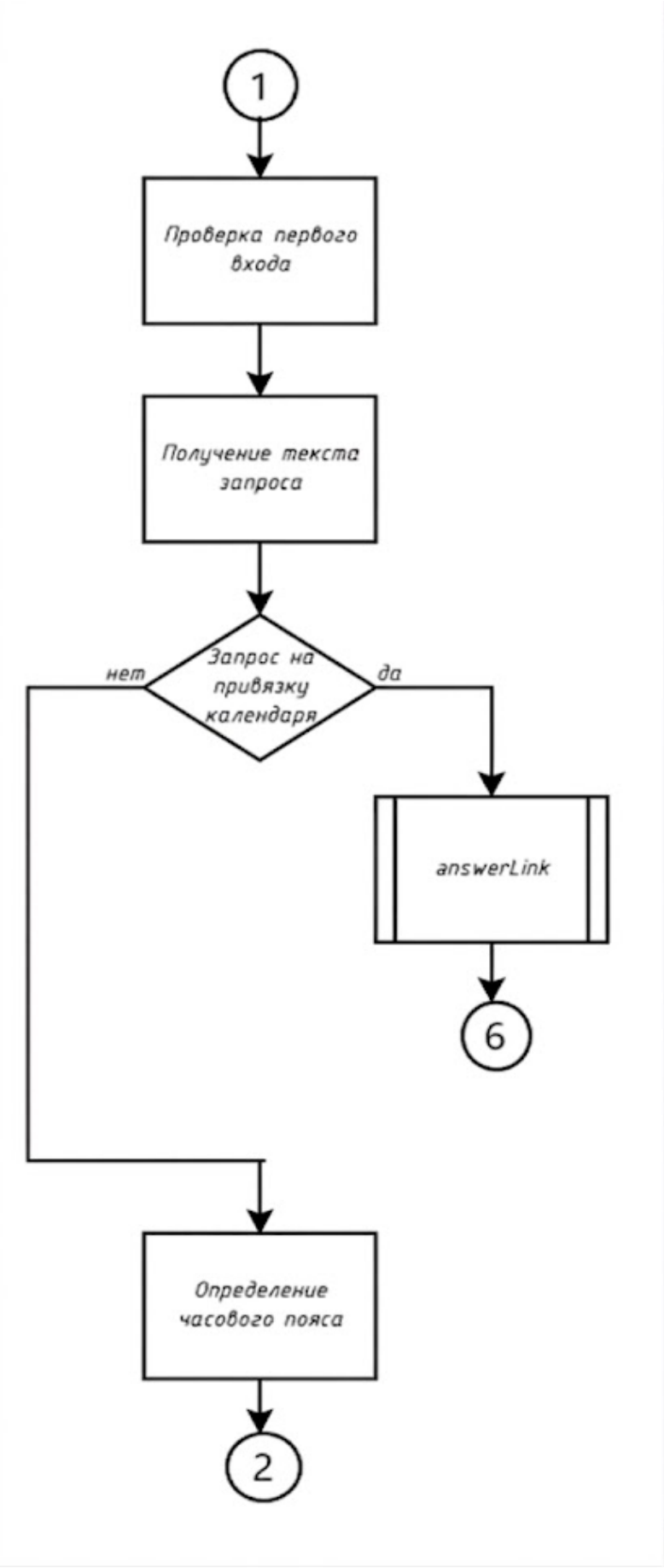


Рисунок 3 - Блок-схема метода Handle()

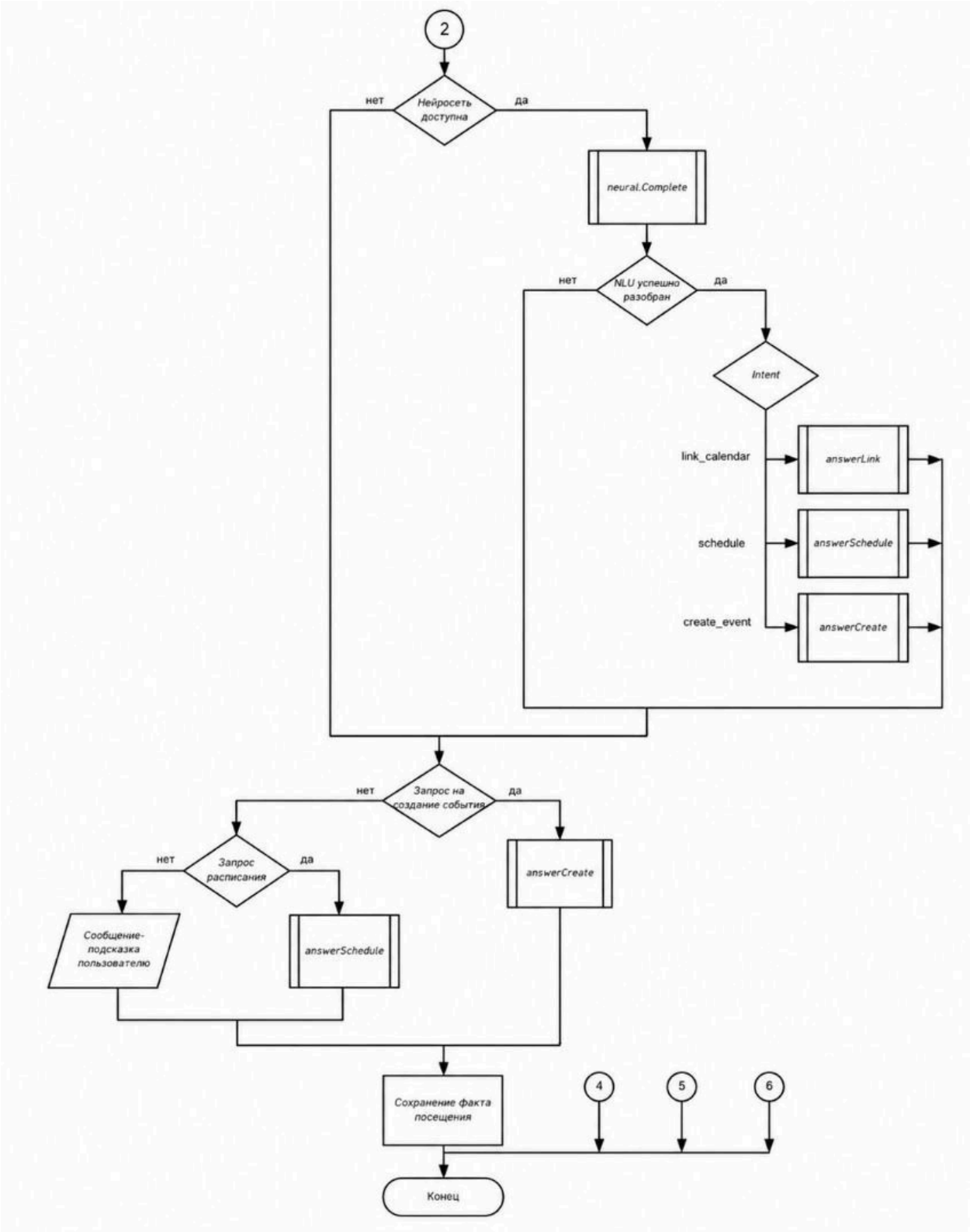


Рисунок 4 - Блок-схема метода Handle()

На рисунке 5 представлена блок-схема ассемблерной вставки.

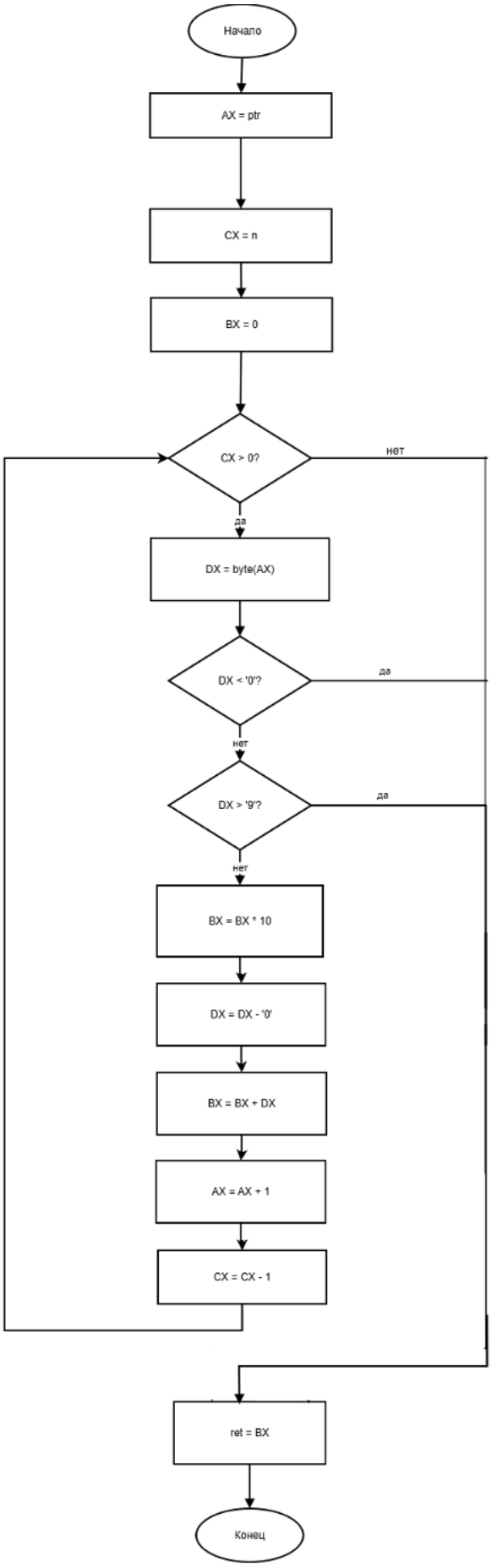


Рисунок 5 - Ассемблерная вставка

Руководство пользователя

Для запуска приложения необходимо подготовить .env-файл с параметрами сервера, Google OAuth и при необходимости нейросетевого API. Минимально требуется задать внешний HTTPS-адрес PUBLIC_BASE_URL, OAuth client id и secret, redirect URL, а также путь к базе данных или оставить значение по умолчанию data/app.db.

Для серверного развёртывания предусмотрена команда docker compose up -d --build. После запуска маршрут /healthz должен возвращать ok, а webhook навыка Алисы должен указывать на https://<домен>/alice.

Со стороны пользователя установка приложения не требуется. Пользователь открывает навык в Алисе, произносит «привязать календарь», переходит по ссылке авторизации Google и разрешает доступ к календарю. После привязки можно пользоваться голосовыми командами для просмотра расписания и создания событий.

Первый сценарий пользователя - привязка календаря. В Алисе нужно сказать: «привязать календарь» (Рисунок 6). Навык вернёт ссылку и кнопку. Пользователь открывает ссылку, входит в Google и разрешает доступ к календарю. После успешного OAuth callback токен сохраняется в SQLite.

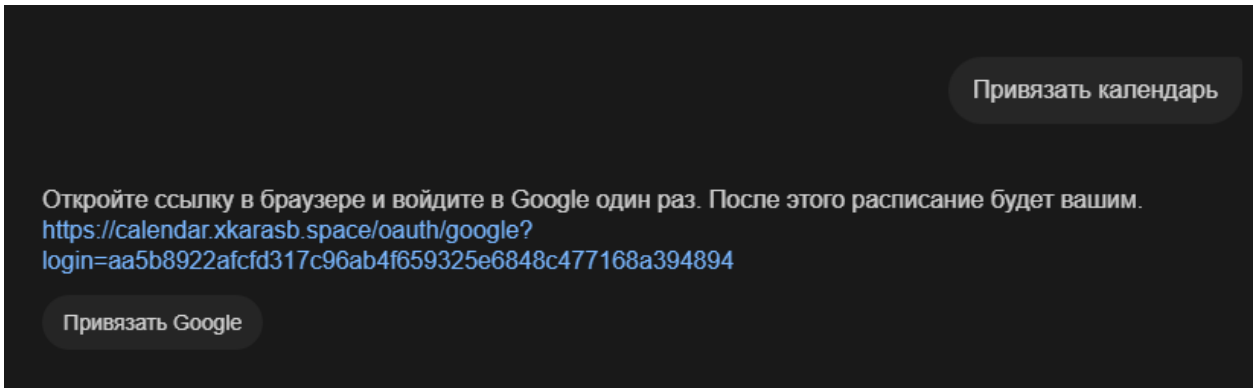


Рисунок 6 - Привязка календаря

Пользователь может обращаться к навыку естественными фразами, а не строго заданными командами. Для распознавания намерения и даты используется языковая модель, которая анализирует текст запроса и преобразует его во внутреннее представление команды. Поэтому приведённые

ниже фразы являются не полным списком допустимых команд, а примерами пользовательских формулировок:

– «что у меня сегодня»;
– «покажи расписание на завтра» (Рисунок 7);
– «какие встречи в пятницу»;
– «что в календаре 25 мая».

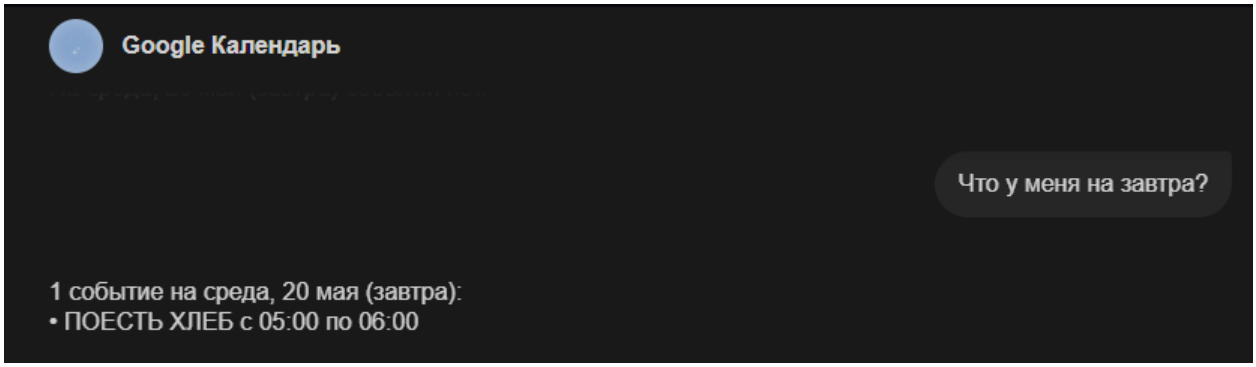


Рисунок 7 - Запуск команды

Навык определит дату, запросит события и перечислит их с временем или пометкой события на весь день.

Для создания события используются команды (навык не ограничивается только этими вариантами: похожие по смыслу запросы также обрабатываются языковой моделью, которая определяет намерение пользователя и нужную дату.):

– «создай встречу завтра в 15 на час» (Рисунок 8);
– «добавь созвон в пятницу в 10:30»;
– «запланируй событие послезавтра вечером».

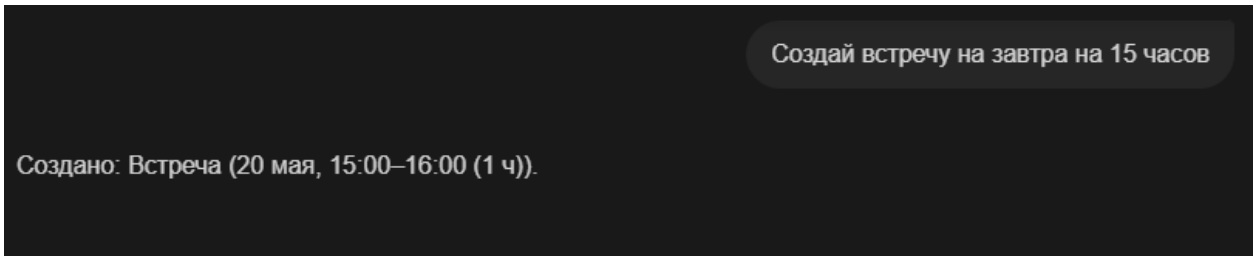


Рисунок 8 - Запуск команды

Если время или дата распознаны недостаточно точно, пользователь получит сообщение о невозможности создать событие. Так же если навык

видит, что на время, когда создаёт человек событие, уже есть что-то, то навык оповестит пользователя.

Заключение

В ходе выполнения курсовой работы было разработано серверное приложение calendar2alice на языке Go. Программа реализует голосовой навык для Алисы, связывает пользователя с Google Календарём через OAuth 2.0, читает расписание и создаёт события по естественным русскоязычным командам.

Проект имеет модульную структуру: протокол Алисы, HTTP-сервер, обработчик команд, календарный сервис, OAuth, конфигурация, хранилище и нейросетевой клиент разделены по пакетам. Это упрощает сопровождение программы, тестирование отдельных частей и дальнейшее развитие.

Практическая ценность приложения заключается в том, что пользователь может работать с календарём голосом, не открывая интерфейс Google Calendar. Разработанная структура позволяет расширять навык: добавлять удаление событий, подтверждение конфликтов, поддержку нескольких календарей и более детальные ответы о расписании.

Список используемых источников

1. Документация языка Go [Электронный ресурс]. – URL: https://go.dev/doc/ (дата обращения: 01.05.2026).

2. Пакет net/http стандартной библиотеки Go [Электронный ресурс]. – URL: https://pkg.go.dev/net/http (дата обращения:01.05.2026).

3. OAuth 2.0 Documentation [Электронный ресурс]. – URL: https://oauth.net/2/ (дата обращения: 01.05.2026).

4. Google Calendar API Documentation [Электронный ресурс]. – URL: https://developers.google.com/workspace/calendar/api/guides/overview?hl=ru (дата обращения: 01.05.2026).

5. Документация платформы навыков Алисы [Электронный ресурс]. – URL: https://yandex.ru/dev/dialogs/alice/doc/ru/about (дата обращения: 01.05.2026).

6. Документация SQLite [Электронный ресурс]. – URL: https://www.sqlite.org/docs.html (дата обращения: 03.05.2026).

7. Документация Docker [Электронный ресурс]. – URL: https://docs.docker.com (дата обращения: 02.05.2026).

Приложение А. Структура и листинги программы

Фрагмент конфигурации

```
type Config struct {
    HTTPAddr            string
    DefaultTimezone     string
    IgnoreAliceTimezone bool
    NeuralDeadline      time.Duration
    DatabasePath        string
    PublicBaseURL       string

    GoogleOAuthClientID     string
    GoogleOAuthClientSecret string
    GoogleOAuthRedirectURL  string
    GoogleCalendarID        string

    NeuralAPIKey   string
    NeuralFolderID string
    NeuralPromptID string
}
```

Фрагмент HTTP-сервера

```
mux := http.NewServeMux()
mux.HandleFunc("/healthz", s.handleHealth)
mux.HandleFunc("/alice", s.handleAlice)
if googleOAuth != nil {
    mux.HandleFunc("/oauth/google", googleOAuth.HandleLogin)
    mux.HandleFunc("/oauth/google/callback",
googleOAuth.HandleCallback)
}
```

Основной выбор сценария навыка

```
if nlu != nil {
    switch strings.ToLower(strings.TrimSpace(nlu.Intent)) {
    case "link_calendar":
        return s.answerLink(ctx, in, yid)
    case "schedule":
        return s.answerSchedule(ctx, in, yid, utter, nlu,
firstVisit)
    case "create_event":
        return s.answerCreate(ctx, in, yid, utter, nlu,
firstVisit)
    }
}

switch {
case wantsCreate(utterIn):
    return s.answerCreate(ctx, in, yid, utter, nil, firstVisit)
case wantsSchedule(utterIn):
```

```
    return s.answerSchedule(ctx, in, yid, utter, nil,
firstVisit)
default:
    return alice.Reply(in, "Можно спросить расписание или
создать событие.", false), nil
}
```

Внутренние структуры календаря

```
type Event struct {
    ID       string
    Summary  string
    Start    time.Time
    End      time.Time
    AllDay   bool
    Calendar string
}

type EventInput struct {
    Summary string
    Start   time.Time
    End     time.Time
    AllDay  bool
}
```

skill.go

```
package handler

import (
     "context"
     "database/sql"
     "errors"
     "fmt"
     "log"
     "strings"
     "time"

     "calendar2alice/internal/alice"
     "calendar2alice/internal/auth"
     "calendar2alice/internal/calendar"
     "calendar2alice/internal/config"
     "calendar2alice/internal/googlehttp"
     "calendar2alice/internal/neural"
     "calendar2alice/internal/store"

     "golang.org/x/oauth2"
)

var ErrNotLinked = errors.New("calendar not linked")

type Skill struct {
     cfg         *config.Config
```

```
	store       *store.Store
	googleOAuth *auth.GoogleOAuth
	neural      *neural.Client
	now         func() time.Time
}

func NewSkill(cfg *config.Config, st *store.Store, googleOAuth
*auth.GoogleOAuth, n *neural.Client) *Skill {
	return &Skill{
		cfg:         cfg,
		store:       st,
		googleOAuth: googleOAuth,
		neural:      n,
		now:         time.Now,
	}
}

func (s *Skill) Handle(ctx context.Context, in *alice.Incoming)
(*alice.Outgoing, error) {
	if in == nil {
		return nil, fmt.Errorf("nil request")
	}

	yid := yandexUserID(in)
	if yid == "" {
		return alice.Reply(in, "Нужен идентификатор
пользователя Яндекса. Включите передачу данных пользователю в
консоли навыка и войдите в аккаунт Яндекса.", false), nil
	}

	seenBefore, errSeen := s.store.HasSkillSeen(ctx, yid)
	if errSeen != nil {
		log.Printf("skill_seen: чтение: %v", errSeen)
		seenBefore = true
	}
	firstVisit := !seenBefore
	defer func() {
		if err := s.store.MarkSkillSeen(ctx, yid); err != nil {
			log.Printf("skill_seen: запись: %v", err)
		}
	}()

	utterIn := strings.TrimSpace(in.Request.OriginalUtterance)
	if utterIn == "" {
		utterIn = strings.TrimSpace(in.Request.Command)
	}

	utter := calendarUtteranceFocus(utterIn)

	if wantsLink(utterIn) {
		return s.answerLink(ctx, in, yid)
	}
```

```
     loc := userLocation(in, s.cfg.DefaultTimezone,
s.cfg.IgnoreAliceTimezone)
     ref := s.now().In(loc)

     var nlu *NLUOutput
     if s.neural == nil {
          log.Printf("neural: запрос без агента (клиент не
сконфигурирован), utter=%q", truncateForLog(utterIn, 100))
     } else {
          nd := s.neuralDeadline()
          nctx, cancel := context.WithTimeout(ctx, nd)
          text, err := s.neural.Complete(nctx, utter, ref, loc)
          cancel()
          if err != nil {
               if errors.Is(err, context.DeadlineExceeded) {
                    log.Printf("neural: таймаут ожидания агента
(%v), локальный разбор | utter=%q", nd, truncateForLog(utterIn,
100))
               } else {
                    log.Printf("neural: ошибка API: %v |
utter=%q", err, truncateForLog(utterIn, 100))
               }
          } else if strings.TrimSpace(text) == "" {
               log.Printf("neural: пустой ответ модели |
utter=%q", truncateForLog(utterIn, 100))
          } else {
               var perr error
               nlu, perr = parseNLUJSON(text)
               if perr != nil || nlu == nil {
                    log.Printf("neural: ответ получен, но JSON не
разобран: %v | превью ответа=%q", perr, truncateForLog(text,
200))
               } else {
                    log.Printf("neural: OK intent=%s | utter=%q",
strings.TrimSpace(nlu.Intent), truncateForLog(utterIn, 100))
               }
          }
     }

     if nlu != nil {
          switch strings.ToLower(strings.TrimSpace(nlu.Intent)) {
          case "link_calendar":
               return s.answerLink(ctx, in, yid)
          case "schedule":
               return s.answerSchedule(ctx, in, yid, utter, nlu,
firstVisit)
          case "create_event":
               return s.answerCreate(ctx, in, yid, utter, nlu,
firstVisit)
          }
     }

     switch {
     case wantsCreate(utterIn):
```

```
          return s.answerCreate(ctx, in, yid, utter, nil,
firstVisit)
     case wantsSchedule(utterIn):
          return s.answerSchedule(ctx, in, yid, utter, nil,
firstVisit)
     default:
          msg := "Можно спросить расписание на день или попросить
создать событие."
          if firstVisit {
               msg += " Для привязки Google скажите «привязать
календарь»."
          }
          return alice.Reply(in, msg, false), nil
     }
}

func (s *Skill) neuralDeadline() time.Duration {
     if s.cfg != nil && s.cfg.NeuralDeadline > 0 {
          return s.cfg.NeuralDeadline
     }
     return 4 * time.Second
}

func yandexUserID(in *alice.Incoming) string {
     if in.Session.User != nil {
          if u := strings.TrimSpace(in.Session.User.UserID); u !=
"" {
               return u
          }
     }
     return strings.TrimSpace(in.Session.UserID)
}

func wantsSchedule(utter string) bool {
     u := strings.ToLower(utter)
     return strings.Contains(u, "расписан") ||
          strings.Contains(u, "план") ||
          strings.Contains(u, "встреч") ||
          strings.Contains(u, "событ") ||
          strings.Contains(u, "календар")
}

func wantsCreate(utter string) bool {
     u := strings.ToLower(utter)
     return strings.Contains(u, "создай") ||
          strings.Contains(u, "добавь") ||
          strings.Contains(u, "запланируй") ||
          strings.Contains(u, "новое событие")
}

func wantsLink(utter string) bool {
     u := strings.ToLower(utter)
     if strings.Contains(u, "привяз") {
          return true
```

```
	}
	if strings.Contains(u, "подключ") && strings.Contains(u,
"календар") {
		return true
	}
	return strings.Contains(u, "авториз") ||
		strings.Contains(u, "oauth") ||
		(strings.Contains(u, "гугл") && strings.Contains(u,
"календар"))
}

func (s *Skill) answerLink(ctx context.Context, in
*alice.Incoming, yid string) (*alice.Outgoing, error) {
	if s.googleOAuth == nil {
		return alice.Reply(in, "Привязка Google на сервере не
настроена: задайте OAuth client id/secret и
GOOGLE_OAUTH_REDIRECT_URL.", false), nil
	}
	if s.cfg.PublicBaseURL == "" {
		return alice.Reply(in, "Задайте PUBLIC_BASE_URL —
внешний адрес сервера с HTTPS, чтобы открыть ссылку привязки в
браузере.", false), nil
	}
	token, err := s.store.CreateLinkToken(ctx, yid,
20*time.Minute)
	if err != nil {
		return alice.Reply(in, "Не удалось выдать ссылку:
"+shortErr(err), false), nil
	}
	url := s.cfg.PublicBaseURL + "/oauth/google?login=" + token
	text := "Откройте ссылку в браузере и войдите в Google один
раз. После этого расписание будет вашим.\n" + url
	tts := "Откройте ссылку на экране или кнопку ниже и войдите
в Google."
	btns := []alice.Button{
		{Title: "Привязать Google", URL: url, Hide: false},
	}
	return alice.ReplyWithButtons(in, text, tts, btns, false),
nil
}

func (s *Skill) answerSchedule(ctx context.Context, in
*alice.Incoming, yid, utter string, nlu *NLUOutput, firstVisit
bool) (*alice.Outgoing, error) {
	cal, err := s.calendarForUser(ctx, yid)
	if errors.Is(err, ErrNotLinked) {
		if firstVisit {
			return alice.Reply(in, "Календарь не привязан.
Скажите «привязать календарь» и откройте ссылку в браузере.",
false), nil
		}
		return alice.Reply(in, "Календарь не привязан.",
false), nil
	}
```

```
	if err != nil {
		return alice.Reply(in, "Не удалось подготовить доступ к
календарю: "+shortErr(err), false), nil
	}

	loc := userLocation(in, s.cfg.DefaultTimezone,
s.cfg.IgnoreAliceTimezone)
	ref := s.now().In(loc)
	dayStart, _ := scheduleDayRange(utter, nlu, ref, loc)
	from := dayStart
	to := from.AddDate(0, 0, 1)

	events, err := cal.ListEvents(ctx, from, to)
	if err != nil {
		return s.replyCalendarAPIErr(ctx, in, yid, err, "Не
удалось прочитать календарь: ")
	}
	if len(events) == 0 {
		heading := formatScheduleDayHeading(ref, dayStart, loc)
		return alice.Reply(in, fmt.Sprintf("На %s событий
нет.", heading), false), nil
	}
	var b strings.Builder
	heading := formatScheduleDayHeading(ref, dayStart, loc)
	timedCount := 0
	for _, e := range events {
		if !e.AllDay {
			timedCount++
		}
	}
	b.WriteString(fmt.Sprintf("%d %s на %s:\n", timedCount,
ruPluralEventsTimed(timedCount), heading))
	for _, e := range events {
		b.WriteString(formatEventLine(e, loc))
		b.WriteByte('\n')
	}
	return alice.Reply(in, strings.TrimSpace(b.String()),
false), nil
}

func (s *Skill) answerCreate(ctx context.Context, in
*alice.Incoming, yid, utter string, pre *NLUOutput, firstVisit
bool) (*alice.Outgoing, error) {
	cal, err := s.calendarForUser(ctx, yid)
	if errors.Is(err, ErrNotLinked) {
		if firstVisit {
			return alice.Reply(in, "Сначала привяжите
календарь: скажите «привязать календарь».", false), nil
		}
		return alice.Reply(in, "Календарь не привязан.",
false), nil
	}
	if err != nil {
```

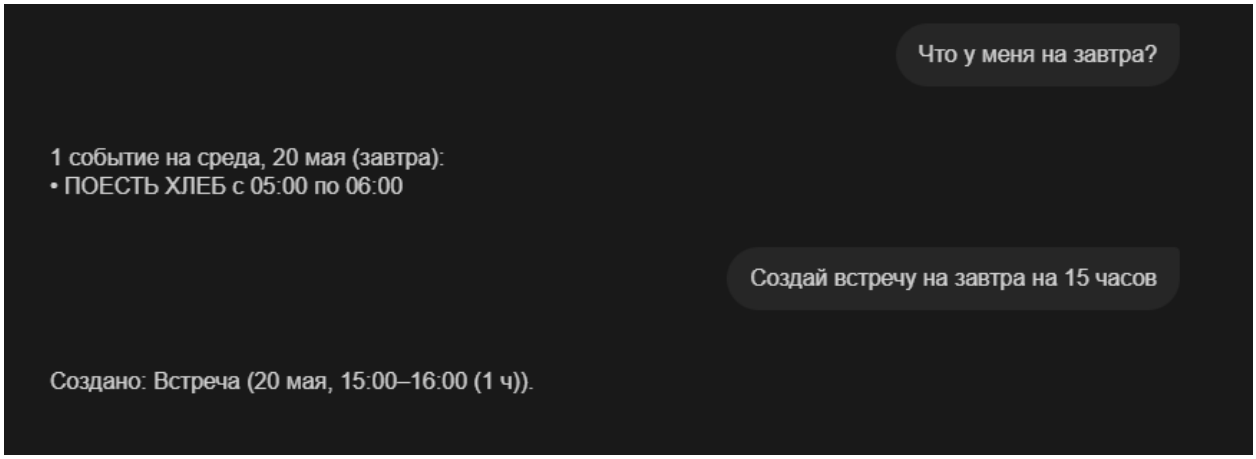
```
          return alice.Reply(in, "Не удалось подготовить доступ к
календарю: "+shortErr(err), false), nil
     }

     loc := userLocation(in, s.cfg.DefaultTimezone,
s.cfg.IgnoreAliceTimezone)
     ref := s.now().In(loc)
     var inp calendar.EventInput
     var ok bool
     if pre != nil {
          inp, ok = eventInputFromNLU(pre, utter, ref, loc)
     }
     if !ok {
          inp, ok = parseCreateEventInput(utter, ref, loc)
     }
     if !ok {
          return alice.Reply(in, "Не понял дату или время.
Скажите, например: «завтра в 20:00», «на целый день на субботу»
или «на 7 мая в 9 30».", false), nil
     }
     var conflicts []calendar.Event
     if events, lerr := cal.ListEvents(ctx, inp.Start, inp.End);
lerr == nil {
          conflicts = overlappingEvents(events, inp.Start,
inp.End)
     }
     ev, err := cal.CreateEvent(ctx, inp)
     if err != nil {
          return s.replyCalendarAPIErr(ctx, in, yid, err, "Не
удалось создать событие: ")
     }
     when := formatCreatedWhen(ev, loc)
     msg := fmt.Sprintf("Создано: %s (%s).", ev.Summary, when)
     if note := conflictNoticeRu(conflicts, loc); note != "" {
          msg += " " + note
     }
     return alice.Reply(in, msg, false), nil
}

func (s *Skill) calendarForUser(ctx context.Context, yid string)
(calendar.Service, error) {
     if s.googleOAuth != nil {
          tok, err := s.store.GetGoogleOAuthToken(ctx, yid)
          if err == nil && tok != nil && tok.RefreshToken != "" {
               cid, _ := s.store.GetCalendarID(ctx, yid)
               if cid == "" {
                    cid = "primary"
               }
               client, err := googlehttp.OAuthClient(ctx,
s.googleOAuth.OAuth2(),
                    func(c context.Context) (*oauth2.Token,
error) {
                         return s.store.GetGoogleOAuthToken(c,
yid)
```

```
                    },
                    func(c context.Context, t *oauth2.Token)
error {
                         return s.store.SaveGoogleOAuthToken(c,
yid, t)
                    },
               )
               if err != nil {
                    return nil, err
               }
               return calendar.NewGoogleAPIService(ctx, client,
cid)
          }
          if err != nil && !errors.Is(err, sql.ErrNoRows) {
               return nil, err
          }
     }

     return nil, ErrNotLinked
}

func ruPluralEventsTimed(n int) string {
     if n < 0 {
          n = -n
     }
     mod100 := n % 100
     mod10 := n % 10
     if mod10 == 1 && mod100 != 11 {
          return "событие"
     }
     if mod10 >= 2 && mod10 <= 4 && (mod100 < 10 || mod100 >= 20)
{
          return "события"
     }
     return "событий"
}

func formatEventLine(e calendar.Event, displayLoc
*time.Location) string {
     title := strings.TrimSpace(e.Summary)
     if title == "" {
          title = "Событие"
     }
     if e.AllDay {
          return fmt.Sprintf("• %s (весь день)", title)
     }
     st := e.Start
     en := e.End
     if displayLoc != nil {
          st = st.In(displayLoc)
          en = en.In(displayLoc)
     }
     return fmt.Sprintf("• %s с %s по %s", title,
st.Format("15:04"), en.Format("15:04"))
```

```
}

func shortErr(err error) string {
     msg := err.Error()
     if len(msg) > 120 {
          return msg[:120] + "…"
     }
     return msg
}

func scheduleDayRange(utter string, nlu *NLUOutput, ref
time.Time, loc *time.Location) (dayStart time.Time, dayLabel
string) {
     ref = ref.In(loc)
     u := strings.ToLower(strings.TrimSpace(utter))
     if d, _, ok := resolveScheduleDayUtterCore(u, ref, loc); ok
{
          ds := startOfDayInLoc(d, loc)
          return ds, relativeDayLabel(ref, ds, loc)
     }
     if nlu != nil && nlu.When != nil {
          if ds, ok := dayStartFromNLUWhen(nlu.When, utter, ref,
loc); ok {
               return ds, relativeDayLabel(ref, ds, loc)
          }
     }
     return resolveScheduleDay(utter, ref, loc)
}

func relativeDayLabel(ref, dayStart time.Time, loc
*time.Location) string {
     ref0 := startOfDayInLoc(ref.In(loc), loc)
     d0 := startOfDayInLoc(dayStart.In(loc), loc)
     n := int(d0.Sub(ref0) / (24 * time.Hour))
     switch n {
     case 0:
          return "сегодня"
     case 1:
          return "завтра"
     case 2:
          return "послезавтра"
     case -1:
          return "вчера"
     default:
          return formatRuDayTitle(dayStart, loc)
     }
}

func truncateForLog(s string, maxRunes int) string {
     r := []rune(strings.TrimSpace(s))
     if len(r) <= maxRunes {
          return string(r)
     }
     return string(r[:maxRunes]) + "…"
```

```
}

func formatCreatedWhen(ev *calendar.Event, loc *time.Location)
string {
     if ev == nil {
          return ""
     }
     title := formatRuDayTitle(ev.Start, loc)
     if ev.AllDay {
          return title + " (весь день)"
     }
     st := ev.Start.In(loc)
     en := ev.End.In(loc)
     dur := ev.End.Sub(ev.Start)
     s := fmt.Sprintf("%s, %s–%s", title, st.Format("15:04"),
en.Format("15:04"))
     if d := formatDurationRu(dur); d != "" {
          s += " (" + d + ")"
     }
     return s
}

func formatDurationRu(d time.Duration) string {
     if d <= 0 {
          return ""
     }
     m := int(d.Round(time.Minute) / time.Minute)
     if m <= 0 {
          return ""
     }
     h := m / 60
     mm := m % 60
     switch {
     case h == 0:
          return fmt.Sprintf("%d мин", mm)
     case mm == 0:
          if h == 1 {
               return "1 ч"
          }
          return fmt.Sprintf("%d ч", h)
     default:
          if h == 1 {
               return fmt.Sprintf("1 ч %d мин", mm)
          }
          return fmt.Sprintf("%d ч %d мин", h, mm)
     }
}

func overlappingEvents(events []calendar.Event, start, end
time.Time) []calendar.Event {
     var out []calendar.Event
     for _, e := range events {
          if intervalsOverlap(start, end, e.Start, e.End) {
               out = append(out, e)
```

```
            }
        }
        return out
}

func intervalsOverlap(aStart, aEnd, bStart, bEnd time.Time) bool
{
        if !aEnd.After(aStart) || !bEnd.After(bStart) {
            return false
        }
        return aStart.Before(bEnd) && bStart.Before(aEnd)
}

func conflictNoticeRu(conflicts []calendar.Event, loc
*time.Location) string {
        if len(conflicts) == 0 {
            return ""
        }
        max := len(conflicts)
        if max > 2 {
            max = 2
        }
        parts := make([]string, 0, max)
        for i := 0; i < max; i++ {
            parts = append(parts, formatConflictBrief(conflicts[i],
loc))
        }
        if len(conflicts) > max {
            return fmt.Sprintf("Внимание: время пересекается с %s и
ещё %d %s.",
                strings.Join(parts, "; "), len(conflicts)-max,
ruPluralEventsTimed(len(conflicts)-max))
        }
        return "Внимание: время пересекается с " +
strings.Join(parts, "; ") + "."
}

func formatConflictBrief(e calendar.Event, loc *time.Location)
string {
        title := strings.TrimSpace(e.Summary)
        if title == "" {
            title = "событием"
        }
        if e.AllDay {
            return fmt.Sprintf("«%s» (весь день)", title)
        }
        st := e.Start.In(loc)
        en := e.End.In(loc)
        return fmt.Sprintf("«%s» %s–%s", title, st.Format("15:04"),
en.Format("15:04"))
}
```

Приложение Б. Примеры работы программы

Чат с Алисой