

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

КУРСОВАЯ РАБОТА
по дисциплине «Объектно-ориентированное программирование»
Тема: Создание бота с расписанием ЛЭТИ для Telegram

Студенты гр. 4351

Преподаватель

Пашаян А.Н.
Погодин Д.А.

Кулагин М.В.

Санкт-Петербург

2025

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студенты Пашаян А.Н. и Погодин Д.А.

Группа 4351

Тема работы: создание бота с расписанием ЛЭТИ для Telegram

Исходные данные:

Язык: JavaScript (Node.js v24.11.1)

ПО для разработки: Visual Studio Code

Содержание пояснительной записки:

«Содержание», «Введение», «Цели и задачи», «Требования к системе»,
«Проектирование системы», «Диаграмма вариантов использования»,
«Архитектура решения», «Диаграмма классов», «Реализация», «Структура
кода», «Описание интерфейса пользователя программы», «Примеры работы»,
«Заключение», «Список использованных источников», «Приложение. Листинг
файла rasp.js»

Дата выдачи задания: 30.09.2025

Дата сдачи реферата: 29.12.2025

Дата защиты реферата: 29.12.2025

Студенты

Пашаян А.Н.
Погодин Д.А.

Преподаватель

Кулагин М.В.

АННОТАЦИЯ

В данной курсовой работе рассматривается разработка Telegram-бота для получения расписания занятий и экзаменов Санкт-Петербургского государственного электротехнического университета «ЛЭТИ». Бот реализован с использованием языка JavaScript на платформе Node.js и взаимодействует с публичным API ЛЭТИ. В работе описаны архитектура программы, основные классы и их взаимодействие, а также пользовательский интерфейс. Результатом работы является полностью функционирующий бот в Telegram, размещённый на облачном сервисе Render.

СОДЕРЖАНИЕ

	Введение	5
1.	Цели и задачи	6
2.	Требования к системе	7
3.	Проектирование системы	8
3.1.	Диаграмма вариантов использования	8
3.2.	Архитектура решения	9
3.3.	Диаграмма классов	9
4.	Реализация	15
4.1.	Структура кода	15
4.2.	Описание интерфейса пользователя программы	16
4.3.	Примеры работы	16
	Заключение	19
	Список использованных источников	20
	Приложение. Листинг файла rasp.js	21

ВВЕДЕНИЕ

Сегодня всё больше студентов предпочитают использовать мессенджеры для получения информации, в частности Telegram. Именно поэтому целью данной курсовой работы было выбрано создание Telegram-бота, предоставляющего актуальное расписание и информацию об экзаменах для студентов СПбГЭТУ «ЛЭТИ». В работе разработан функциональный бот, интегрированный с публичным API университета, способный корректно обрабатывать информацию о занятиях и представлять её пользователю в удобном формате. Программа разработана на JavaScript (Node.js) и развёрнута в облаке (Render) для круглосуточной доступности.

1. ЦЕЛИ И ЗАДАЧИ

Цель работы: разработка Telegram-бота, способного предоставлять актуальное расписание занятий и экзаменов для любой учебной группы ЛЭТИ. В рамках работы были поставлены следующие задачи:

- изучить структуру публичного API ЛЭТИ и интегрировать его;
- реализовать обработку расписания: по группе, по дню, по типу недели;
- реализовать команды и клавиатуру для удобного взаимодействия с ботом;
- реализовать бота на языке JavaScript;
- обеспечить постоянную работу бота.

2. ТРЕБОВАНИЯ К СИСТЕМЕ

Функциональные требования:

- пользователь может указать номер своей группы (4 цифры);
- бот принимает следующие команды:
 - «Ближайшая пара» – вывод информации о ближайшем занятии с учётом текущего времени и недели;
 - «Сегодня» – расписание занятий на текущий учебный день;
 - «Завтра» – расписание на следующий учебный день (с учётом перехода на следующую календарную дату);
 - «Вся неделя» – расписание на всю учебную неделю с возможностью выбора чётной/нечётной недели;
 - «День недели» – выбор конкретного дня (с понедельника по субботу) и чётности для вывода расписания;
 - «Экзамены» – вывод расписания экзаменов для группы;
 - «Сменить группу» – смена номера группы;
- бот отображает вид занятия (лекция, практика, лабораторная) в выводе;
- бот корректно обрабатывает ошибки сетевого взаимодействия и некорректный ввод.

Нефункциональные требования:

- система должна быть отказоустойчивой при кратковременных ошибках API;
- бот должен работать круглосуточно на облачной платформе;
- токен бота должен храниться в переменных окружения, а не в коде.

3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1 Диаграмма вариантов использования.

Актером выступает пользователь (студент).

Варианты использования:

- поиск ближайшей пары;
- получение расписания на сегодня;
- получение расписания на завтра;
- получение расписания на всю неделю;
- получение расписания на конкретный день;
- получение расписания экзаменов;
- смена номера группы.

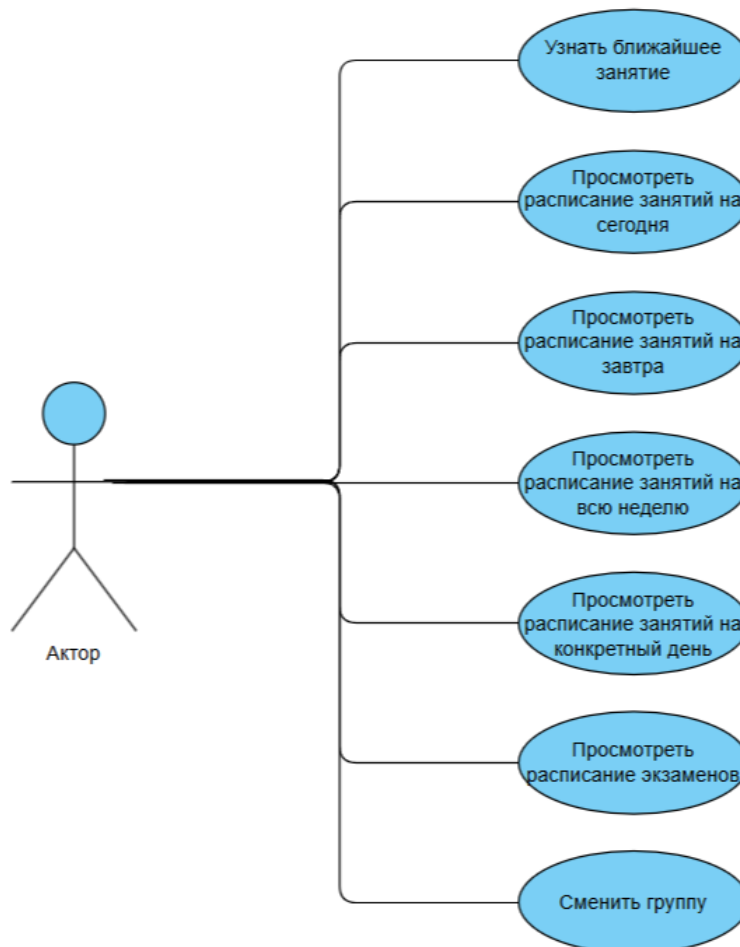


Рис. 1. Диаграмма вариантов использования системы (use cases)

3.2 Архитектура решения

Программа имеет модульную архитектуру, на основе классов/сервисов.

- **UserStateService**: хранилище состояния пользователей (в оперативной памяти); хранит группу пользователя, временный выбранный день и др.
- **ApiService**: отвечает за взаимодействие с внешним API ЛЭТИ (запросы расписания и экзаменов).
- **KeyboardService**: формирует клавиатуру с кнопками команд.
- **ScheduleService**: набор статических утилит: работа с датами, вычисление чётности недели, фильтрация занятий, форматирование вывода.
- **BotApp**: главный класс, который создаёт экземпляр бота, регистрирует обработчики и связывает сервисы.

3.3 Диаграмма классов

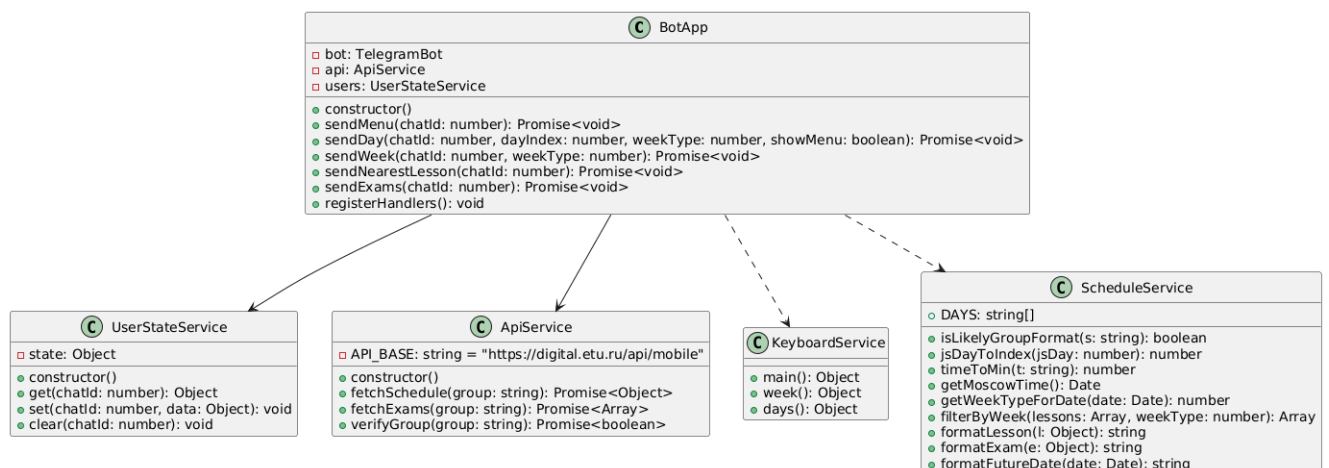


Рис. 2. Диаграмма классов

Спецификация классов.

1) Класс BotApp

Поля	Описание полей
bot: TelegramBot	экземпляр Telegram-бота
api: ApiService	сервис для работы с API ЛЭТИ
users: UserService	сервис хранения пользовательского состояния

Методы	Описание методов
constructor()	инициализирует бота, сервисы и регистрирует обработчики
registerHandlers()	регистрирует обработчики команд и текстовых сообщений
sendMainMenu(chatId: number): void	отправляет пользователю главное меню
handleGroupInput(chatId: number, text: string): void	обрабатывает ввод номера группы
sendTodaySchedule(chatId: number): void	выводит расписание на текущий день
sendTomorrowSchedule(chatId: number): void	выводит расписание на следующий день

sendWeekSchedule(chatId: number, weekType: number): void	выводит расписание на всю неделю
sendDaySchedule(chatId: number, dayIndex: number, weekType: number): void	выводит расписание на выбранный день недели
sendNearestLesson(chatId: number): void	выводит информацию о ближайшей паре
sendExamSchedule(chatId: number): void	выводит расписание экзаменов
changeGroup(chatId: number): void	сбрасывает сохранённую группу пользователя

2) Класс UserStateService

Поля	Описание полей
state: Map<number, Object>	структура данных, где ключом является chatId, а значением – объект состояния пользователя

Методы	Описание методов
get(chatId: number): Object	возвращает состояние пользователя
set(chatId: number, data: Object): void	сохраняет или обновляет состояние пользователя
clear(chatId: number): void	очищает состояние пользователя

hasGroup(chatId: number): boolean	проверяет, задан ли номер группы
setGroup(chatId: number, group: string): void	сохраняет номер группы
getGroup(chatId: number): string	возвращает номер группы пользователя

3) Класс ApiService

Поля	Описание полей
baseUrl: string	базовый URL API университета

Методы	Описание методов
fetchSchedule(group: string): Promise<Object>	получает расписание занятий для указанной группы
fetchExams(group: string): Promise<Array>	получает расписание экзаменов
verifyGroup(group: string): Promise<boolean>	проверяет существование учебной группы
request(endpoint: string, params: Object): Promise<any>	выполняет HTTP-запрос к API

4) Класс ScheduleService

Поля	Описание полей
DAYS: Array<string>	массив названий дней недели (понедельник–суббота)

Методы	Описание методов
getMoscowTime(): Date	возвращает текущее московское время
getWeekType(date: Date): number	определяет чётность недели
jsDayToIndex(jsDay: number): number	преобразует номер дня недели JavaScript в индекс расписания
filterLessonsByWeek(lessons: Array, weekType: number): Array	фильтрует занятия по типу недели
getNearestLesson(schedule: Object): Object null	находит ближайшее занятие
formatLesson(lesson: Object): string	форматирует информацию о занятии для вывода
formatDaySchedule(dayName: string, lessons: Array): string	форматирует расписание одного дня
formatExam(exam: Object): string	форматирует информацию об экзамене

5) Класс KeyboardApp

Методы	Описание методов
mainKeyboard(): Object	возвращает главное меню бота
weekTypeKeyboard(): Object	возвращает клавиатуру выбора чётной/нечётной недели
daySelectKeyboard(): Object	возвращает клавиатуру выбора дня недели
removeKeyboard(): Object	возвращает объект для скрытия клавиатуры

4. РЕАЛИЗАЦИЯ

4.1 Структура кода.

Код написан на языке JavaScript (Node.js v24.11.1). Используются следующие библиотеки:

- node-telegram-bot-api: взаимодействие с Telegram API через polling;
- axios: выполнение HTTP-запросов к API ЛЭТИ;
- express: простой HTTP-сервер для keep-alive на Render (отдельный GET /);
- dotenv: загрузка переменных окружения.

Код организован в один файл с выделением классов. Логика регистрации обработчиков и взаимодействия с пользователем реализована в методе registerHandlers() класса BotApp, где обрабатываются команды и нажатия кнопок.

Используются переменные окружения:

- BOT_TOKEN: токен Telegram-бота (обязательно);
- SEMESTER_START: (необязательно) дата начала семестра в формате YYYY-MM-DD для правильного вычисления чётности недель.

Чётность недели вычисляется двумя способами:

- 1) Если задана переменная SEMESTER_START, то вычисление ведётся как разница в днях между текущей датой и датой начала семестра, делённая на 7 -> индекс недели.
- 2) Если SEMESTER_START не задана, используется fallback на ISO-неделю: вычисляется номер ISO-недели и затем берётся его чётность.

Фильтрация занятий по неделе: занятия в API могут содержать поля week или weeks (строка), в которой встречается «1», «2», «1/2», «вся» и т. п. Фильтрация происходит так: если поле отсутствует, то пара считается для всех недель; если поле содержит «1/2» или вся, то пара выводится всегда; иначе ищем цифру 1 или 2 и сравниваем с требуемой чётностью.

Работа с API ЛЭТИ: метод ApiService.fetchSchedule(group) делает get-запрос к <https://digital.etu.ru/api/mobile/schedule?groupNumber=XXXX>. Ответ ожидается в виде объекта, где ключ – это номер группы, а значение – объект с полем days, содержащим занятия по дням. Функция обрабатывает случаи, когда ответ пуст, или когда структура отличается.

4.2 Описание интерфейса пользователя программы.

Пользовательский интерфейс – специальная клавиатура с кнопками (KeyboardService).

Основная последовательность взаимодействия:

- /start – запрос номера группы.
- Ввод группы – формат 4 цифры; проверка существования через ApiService.verifyGroup.
- Появление основной клавиатуры.
- При выборе «День недели» – показ клавиатуры дней; пользователь выбирает день – бот предлагает выбрать тип недели.
- При выборе «Вся неделя» – бот предлагает выбрать тип недели, затем выводит расписание на каждый день данной.
- При выборе «Экзамены» – вызывается fetchExams и результат форматируется и выводится.

После выполнения каждой команды бот повторно выводит главное меню.

4.3 Примеры работы.

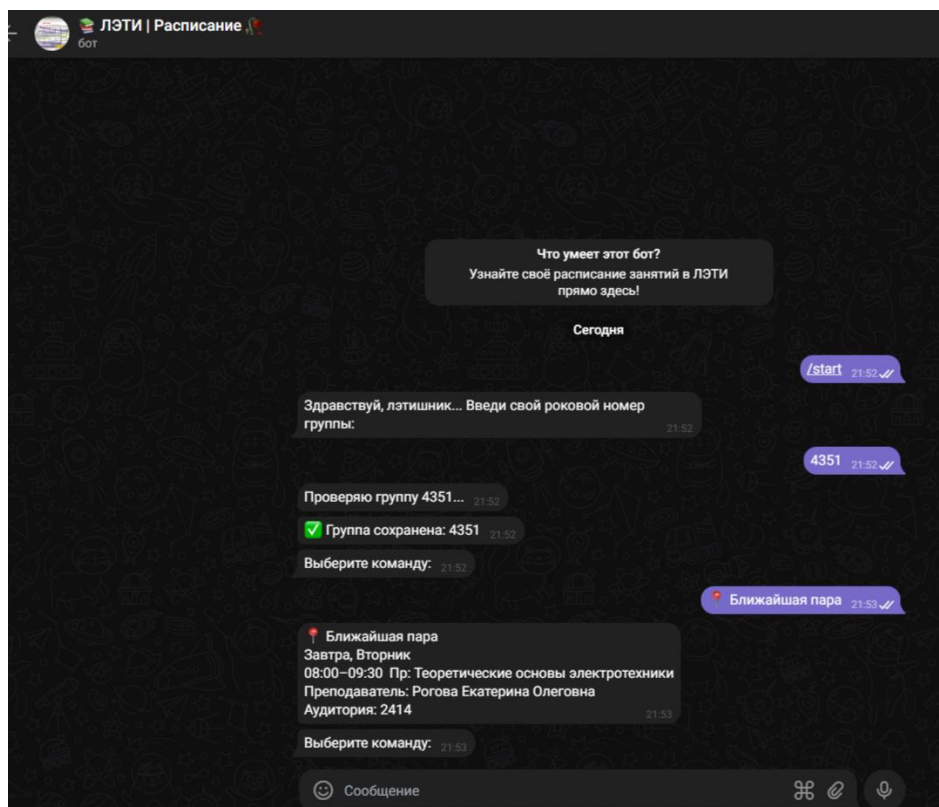


Рис. 3. Начало работы и команда ближайшей пары

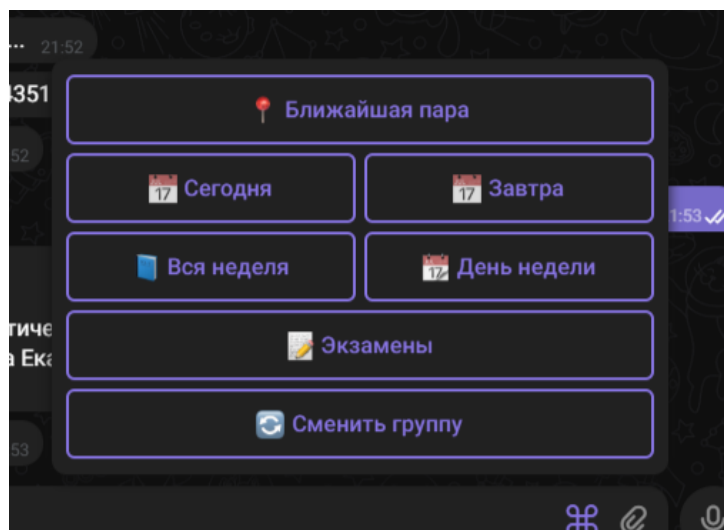


Рис. 4. Основная клавиатура

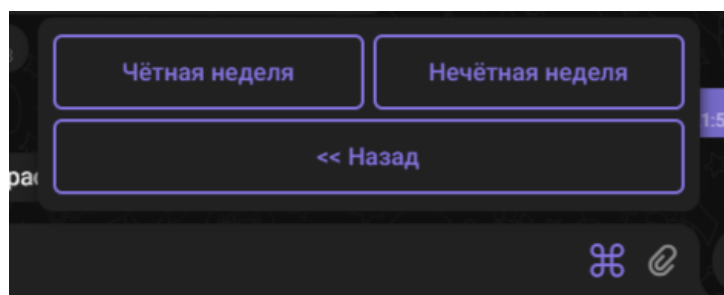


Рис. 5. Подменю при выборе всей недели

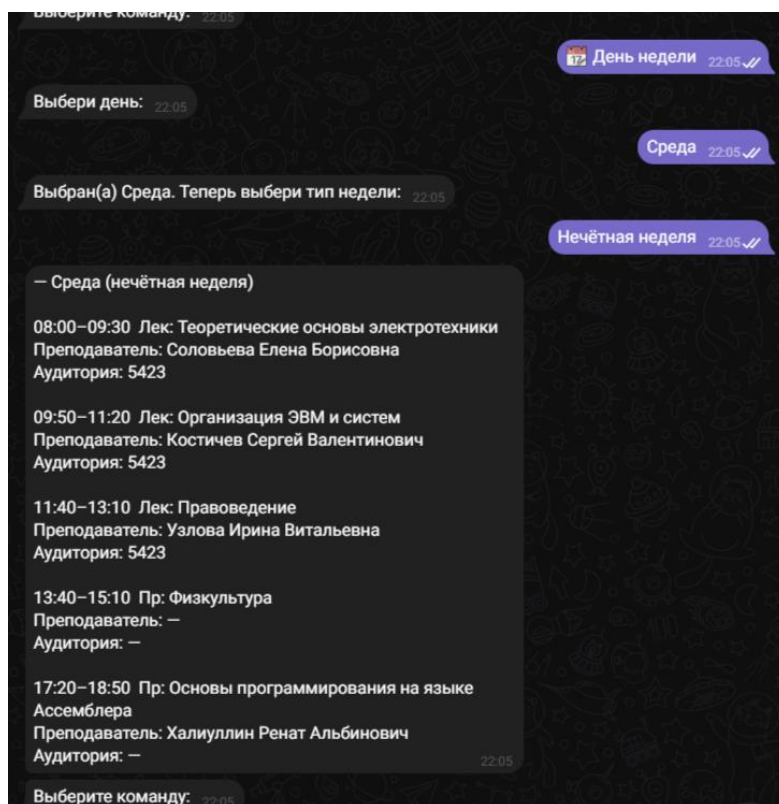


Рис. 6. Выбор дня

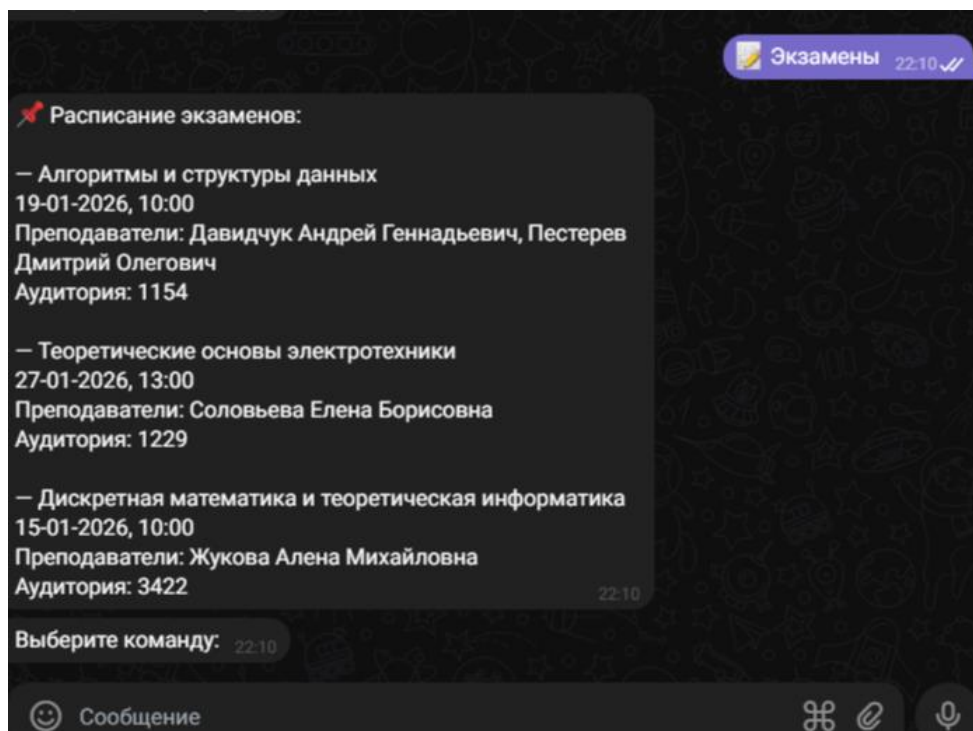


Рис. 7. Расписание экзаменов

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы был разработан бот в мессенджере Telegram, предоставляющий расписание занятий и экзаменов вуза ЛЭТИ. Он решает практическую задачу – упрощает доступ к информации о занятиях, учитывает чётность недель, типы занятий, поддерживает выбор дня и интеграцию с публичным API вуза. Бот протестирован и готов к использованию.

Результаты были выложены на GitHub: <https://github.com/JoyKkk/OOPBotik/>

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ЦНОП СПбГЭТУ «ЛЭТИ». URL: <https://digital.etu.ru/api/docs-public/> (дата обращения: 16.12.2025).
2. Telegram APIs. URL: <https://core.telegram.org/bots/samples> (дата обращения: 16.12.2025).
3. GitHub, документация по node-telegram-bot-api. URL: <https://github.com/yagor/node-telegram-bot-api> (дата обращения: 16.12.2025).
4. Axios, документация. URL: <https://axios-http.com/docs/intro> (дата обращения: 16.12.2025).

ПРИЛОЖЕНИЕ

ЛИСТИНГ ФАЙЛА RASP.JS

```
const TelegramBot = require('node-telegram-bot-api');
const axios = require('axios');
const express = require('express');
require('dotenv').config();

const API_BASE = 'https://digital.etu.ru/api/mobile';
const token = process.env.BOT_TOKEN;

if (!token) {
  console.error('Токен бота не задан в .env');
  process.exit(1);
}

/* ===== SERVICES ===== */
class UserStateService {
  constructor() {
    this.state = {};
  }

  get(chatId) {
    return this.state[chatId] || {};
  }

  set(chatId, data) {
    if (!this.state[chatId]) this.state[chatId] = {};
    Object.assign(this.state[chatId], data);
  }

  clear(chatId) {
    delete this.state[chatId];
  }
}

class ApiService {
  async fetchSchedule(group) {
    try {
      const res = await axios.get(`${API_BASE}/schedule`, {
        params: { groupNumber: group },
        timeout: 10000
      });
    }

    if (!res.data || Object.keys(res.data).length === 0) {
      throw new Error('Пустой ответ от API');
    }

    if (res.data[group]) return res.data[group];
    const firstKey = Object.keys(res.data)[0];
```

```

        return res.data[firstKey];
    } catch (err) {
        const msg = err.response?.status ? `HTTP ${err.response.status}` :
err.message;
        throw new Error(`Не удалось получить расписание: ${msg}`);
    }
}

async fetchExams(group) {
    try {
        const res = await axios.get(`${API_BASE}/exam`, {
            params: { groupNumber: group },
            timeout: 10000
        });

        if (!res.data || Object.keys(res.data).length === 0) return [];
        if (Array.isArray(res.data[group])) return res.data[group];

        const firstKey = Object.keys(res.data)[0];
        return Array.isArray(res.data[firstKey]) ? res.data[firstKey] : [];
    } catch (err) {
        console.error('fetchExams error:', err.message);
        return [];
    }
}

async verifyGroup(group) {
    try {
        await this.fetchSchedule(group);
        return true;
    } catch {
        return false;
    }
}
}

class KeyboardService {
    static main() {
        return {
            reply_markup: {
                resize_keyboard: true,
                keyboard: [
                    ['🔍 Ближайшая пара'],
                    ['📅 Сегодня', '📅 Завтра'],
                    ['📅 Вся неделя', '📅 31 День недели'],
                    ['📝 Экзамены'],
                    ['🔄 Сменить группу']
                ]
            }
        }
    }
};

```

```

}

static week() {
  return {
    reply_markup: {
      resize_keyboard: true,
      keyboard: [
        ['Чётная неделя', 'Нечётная неделя'],
        ['<< Назад']
      ]
    }
  };
}

static days() {
  return {
    reply_markup: {
      resize_keyboard: true,
      keyboard: [
        ['Понедельник', 'Вторник', 'Среда'],
        ['Четверг', 'Пятница', 'Суббота'],
        ['<< Назад']
      ]
    }
  };
}

}

class ScheduleService {
  static DAYS = ['Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница',
    'Суббота', 'Воскресенье'];

  static isLikelyGroupFormat(s) {
    return /^d{4}$/.test(s);
  }

  static jsDayToIndex(jsDay) {
    return (jsDay + 6) % 7;
  }

  static timeToMin(t) {
    if (!t || typeof t !== 'string') return 0;
    const parts = t.split(':').map(x => parseInt(x, 10));
    if (parts.length < 2 || isNaN(parts[0])) return 0;
    return parts[0] * 60 + (isNaN(parts[1]) ? 0 : parts[1]);
  }

  static getMoscowTime() {
    const now = new Date();
    return new Date(now.getTime() + 3 * 60 * 60 * 1000);
  }
}

```

```

}

static getWeekTypeForDate(date) {
  if (process.env.SEMESTER_START) {
    const start = new Date(process.env.SEMESTER_START + 'T00:00:00');
    if (!isNaN(start.getTime())) {
      const diffDays = Math.floor((date - start) / 86400000);
      if (diffDays >= 0) {
        const weekIndex = Math.floor(diffDays / 7);
        return (weekIndex % 2 === 0) ? 1 : 2;
      }
    }
  }
}

const tmp = new Date(date.getTime());
tmp.setHours(0, 0, 0, 0);
tmp.setDate(tmp.getDate() + 3 - ((tmp.getDay() + 6) % 7));
const week1 = new Date(tmp.getFullYear(), 0, 4);
const weekNumber = 1 + Math.round(((tmp.getTime() - week1.getTime()) / 86400000
- 3 + ((week1.getDay() + 6) % 7)) / 7);
return (weekNumber % 2 === 1) ? 1 : 2;
}

static filterByWeek(lessons, weekType) {
  if (!Array.isArray(lessons)) return [];
  return lessons.filter(l => {
    if (!l.week && !l.weeks) return true;
    const wRaw = (l.week || l.weeks || '').toString().toLowerCase();
    if (!wRaw) return true;
    if (wRaw.includes('вср') || wRaw.includes('all') || wRaw.includes('1/2'))
return true;
    const found = wRaw.match(/[12]/);
    if (!found) return true;
    return found[0] === String(weekType);
  });
}

static formatLesson(l) {
  const time = `${l.start_time || '?:??'}-${l.end_time || '?:??'}`;
  const type = l.subjectType ? `${l.subjectType}: ` : '';
  const name = l.name || l.subject || '-';

  const teachers = [l.teacher, l.second_teacher].filter(t => t && t.trim());

  const teacherText = teachers.length ? teachers.join(', ') : '-';
  const room = l.room ? `Аудитория: ${l.room}` : `Аудитория: -`;

  return `${time} ${type}${name}\nПреподаватели: ${teacherText}\n${room}`;
}

```



```

static formatExam(e) {
  const subj = e.name || '-';
  const date = e.date || (e.timestamp ? new Date(e.timestamp *
1000).toLocaleDateString() : '-');
  const time = e.start_time || (e.timestamp ? new Date(e.timestamp *
1000).toLocaleTimeString().slice(0, 5) : '-');
  const teachers = [e.teacher, e.secondTeacher].filter(Boolean).join(', ') ||
(Array.isArray(e.teachers) ? e.teachers.join(', ') : '-');
  const room = e.room || '-';
  return `— ${subj}\n${date}, ${time}\nПреподаватели: ${teachers}\nАудитория:
${room}`;
}

static formatFutureDate(date) {
  const today = new Date();
  today.setHours(0, 0, 0, 0);
  const target = new Date(date);
  target.setHours(0, 0, 0, 0);

  const diffTime = target - today;
  const diffDays = Math.ceil(diffTime / (1000 * 60 * 60 * 24));

  if (diffDays === 1) return 'Завтра';
  if (diffDays === 2) return 'Послезавтра';

  const day = target.getDate().toString().padStart(2, '0');
  const month = (target.getMonth() + 1).toString().padStart(2, '0');
  return `${day}.${month}`;
}
}

// ===== Бот =====
class BotApp {
  constructor() {
    this.bot = new TelegramBot(token, { polling: true });
    this.api = new ApiService();
    this.users = new UserStateService();

    this.registerHandlers();
  }

  async sendMenu(chatId) {
    await this.bot.sendMessage(chatId, 'Выберите команду:',
KeyboardService.main());
  }

  async sendDay(chatId, dayIndex, weekType, showMenu = true) {
    const state = this.users.get(chatId);

    if (!state.group) {

```

```

    await this.bot.sendMessage(chatId, 'Сначала укажи номер группы');
    if (showMenu) await this.sendMenu(chatId);
    return;
  }

  try {
    const sched = await this.api.fetchSchedule(state.group);
    const day = sched.days && sched.days[String(dayIndex)];

    if (!day || !day.lessons || day.lessons.length === 0) {
      await this.bot.sendMessage(chatId, `– ${ScheduleService.DAYS[dayIndex]}:
пар нет`);
      if (showMenu) await this.sendMenu(chatId);
      return;
    }

    const lessons = ScheduleService.filterByWeek(day.lessons, weekType);
    if (!lessons.length) {
      await this.bot.sendMessage(chatId, `– ${ScheduleService.DAYS[dayIndex]}:
пар нет (для выбранной недели)`);
      if (showMenu) await this.sendMenu(chatId);
      return;
    }

    const text = `– ${ScheduleService.DAYS[dayIndex]} (${weekType === 1 ?
'нечётная' : 'чётная'} неделя)\n\n` +
      lessons.map(ScheduleService.formatLesson).join('\n\n');

    await this.bot.sendMessage(chatId, text);
    delete state.selectedDay;

    if (showMenu) await this.sendMenu(chatId);
  } catch (e) {
    console.error('sendDay error', e);
    await this.bot.sendMessage(chatId, 'Ошибка получения данных расписания.');
```

```

    if (showMenu) await this.sendMenu(chatId);
  }
}

async sendWeek(chatId, weekType) {
  const state = this.users.get(chatId);

  if (!state.group) {
    await this.bot.sendMessage(chatId, 'Сначала укажи номер группы');
    await this.sendMenu(chatId);
    return;
  }

  try {
    for (let i = 0; i < 6; i++) {

```

```

        await this.sendDay(chatId, i, weekType, false);
    }
    await this.sendMenu(chatId);
} catch (e) {
    console.error('sendWeek error', e);
    await this.bot.sendMessage(chatId, 'Ошибка получения расписания на неделю.');
```

```

    await this.sendMenu(chatId);
}
}

async sendNearestLesson(chatId) {
    const state = this.users.get(chatId);

    if (!state.group) {
        await this.bot.sendMessage(chatId, 'Сначала укажи номер группы');
        await this.sendMenu(chatId);
        return;
    }

    try {
        const sched = await this.api.fetchSchedule(state.group);
        const nowMoscow = ScheduleService.getMoscowTime();
        const nowHours = nowMoscow.getHours();
        const nowMinutes = nowMoscow.getMinutes();
        const nowTotalMinutes = nowHours * 60 + nowMinutes;

        for (let dayOffset = 0; dayOffset < 7; dayOffset++) {
            const targetDate = new Date(nowMoscow);
            targetDate.setDate(nowMoscow.getDate() + dayOffset);
            targetDate.setHours(0, 0, 0, 0);

            const dayIndex = ScheduleService.jsDayToIndex(targetDate.getDay());
            const weekType = ScheduleService.getWeekTypeForDate(targetDate);

            const day = sched.days && sched.days[String(dayIndex)];
            if (!day || !day.lessons || day.lessons.length === 0) continue;

            const lessons = ScheduleService.filterByWeek(day.lessons, weekType)
                .sort((a, b) => ScheduleService.timeToMin(a.start_time) -
ScheduleService.timeToMin(b.start_time));

            if (!lessons.length) continue;

            for (const lesson of lessons) {
                const startTime = lesson.start_time || "00:00";
                const endTime = lesson.end_time || "23:59";

                const [startHour, startMinute] = startTime.split(':').map(Number);
                const [endHour, endMinute] = endTime.split(':').map(Number);

```

```

const startTotalMinutes = startHour * 60 + startMinute;
const endTotalMinutes = endHour * 60 + endMinute;

if (dayOffset === 0) {
  if (endTotalMinutes < nowTotalMinutes) {
    continue;
  }

  if (startTotalMinutes <= nowTotalMinutes && nowTotalMinutes <
endTotalMinutes) {
    const minutesPassed = nowTotalMinutes - startTotalMinutes;
    const totalDuration = endTotalMinutes - startTotalMinutes;
    const minutesLeft = endTotalMinutes - nowTotalMinutes;

    const text = `🕒 Текущая пара
(сейчас)\n${ScheduleService.formatFutureDate(targetDate)},
${ScheduleService.DAYS[dayIndex]}\n${startTime}-${endTime} ${lesson.subjectType ?
lesson.subjectType : ''}: ${lesson.name || lesson.subject || 'Без
названия'}\nПреподаватель: ${lesson.teacher || '-'}\nАудитория: ${lesson.room || '-
'}`;

    await this.bot.sendMessage(chatId, text);
    return await this.sendMenu(chatId);
  }

  if (startTotalMinutes > nowTotalMinutes) {
    const minutesLeft = startTotalMinutes - nowTotalMinutes;
    const text = `🕒 Ближайшая пара\nСегодня,
${ScheduleService.DAYS[dayIndex]}\n${startTime}-${endTime} (через ${minutesLeft}
мин.)\n${lesson.subjectType ? lesson.subjectType : ''}: ${lesson.name ||
lesson.subject || 'Без названия'}\nПреподаватель: ${lesson.teacher || '-
'}\nАудитория: ${lesson.room || '-'}`;

    await this.bot.sendMessage(chatId, text);
    return await this.sendMenu(chatId);
  }
} else {
  const text = `🕒 Ближайшая пара
\n${ScheduleService.formatFutureDate(targetDate)},
${ScheduleService.DAYS[dayIndex]}\n${startTime}-${endTime} ${lesson.subjectType ?
lesson.subjectType : ''}: ${lesson.name || lesson.subject || 'Без
названия'}\nПреподаватель: ${lesson.teacher || '-'}\nАудитория: ${lesson.room || '-
'}`;

  await this.bot.sendMessage(chatId, text);
  return await this.sendMenu(chatId);
}
}
}

await this.bot.sendMessage(chatId, 'Пар не найдено в ближайшую неделю');
await this.sendMenu(chatId);
} catch (error) {

```

```

        console.error('Ошибка поиска ближайшей пары:', error);
        await this.bot.sendMessage(chatId, 'Ошибка при поиске ближайшей пары.
Попробуйте позже.');
```

```

        await this.sendMenu(chatId);
    }
}

async sendExams(chatId) {
    const state = this.users.get(chatId);

    if (!state.group) {
        await this.bot.sendMessage(chatId, 'Сначала укажи номер группы');
        return this.sendMenu(chatId);
    }

    // Получаем экзамены
    let exams = await this.api.fetchExams(state.group);

    // Сортируем по дате/времени: сначала по timestamp, иначе пытаемся составить из
    date + start_time
    exams.sort((a, b) => {
        const getTs = (e) => {
            if (!e) return 0;
            if (e.timestamp && !isNaN(Number(e.timestamp))) return Number(e.timestamp)
* 1000;
            // попробуем распарсить date + start_time
            if (e.date && e.start_time) {
                // e.date может быть 'YYYY-MM-DD' или 'DD.MM.YYYY' - пробуем оба варианта
                let d = Date.parse(e.date);
                if (isNaN(d) && typeof e.date === 'string' && e.date.includes('.')) {
                    // dd.mm.yyyy -> yyyy-mm-dd
                    const parts = e.date.split('.');
                    if (parts.length === 3) {
                        const dd = parts[0].padStart(2, '0');
                        const mm = parts[1].padStart(2, '0');
                        const yyyy = parts[2];
                        const iso = `${yyyy}-${mm}-${dd}T${e.start_time}`;
                        const parsed = Date.parse(iso);
                        if (!isNaN(parsed)) return parsed;
                    }
                } else if (!isNaN(d)) {
                    // если дата парсится напрямую, добавим время
                    const iso = new Date(d);
                    const [h,m] = (e.start_time || '00:00').split(':').map(Number);
                    iso.setHours(h||0, m||0, 0, 0);
                    return iso.getTime();
                }
            }
        }
        return getTs(a) - getTs(b);
    });
    return exams;
};

```

```

        return getTs(a) - getTs(b);
    });

    if (!exams || exams.length === 0) {
        await this.bot.sendMessage(chatId, 'Расписание экзаменов не найдено для этой группы.');
```

группы.');

```

        return this.sendMenu(chatId);
    }

    const text = '📅 Расписание экзаменов:\n\n' +
exams.map(ScheduleService.formatExam).join('\n\n');
    await this.bot.sendMessage(chatId, text);
    await this.sendMenu(chatId);
}

registerHandlers() {
    this.bot.onText(/\/start/, (msg) => {
        const chatId = msg.chat.id;
        this.users.clear(chatId);
        this.bot.sendMessage(chatId, 'Здравствуй, лэтишник... Введи свой роковой номер группы:');
```

номер группы:');

```

    });

    this.bot.on('message', async (msg) => {
        const chatId = msg.chat.id;
        const text = msg.text && msg.text.trim();
        if (!text) return;

        if (text.startsWith('/')) return;

        const state = this.users.get(chatId);

        if (text === '🔄 Сменить группу') {
            this.users.clear(chatId);
            await this.bot.sendMessage(chatId, 'Введи новый номер группы:');
            return;
        }

        if (!state.group) {
            if (!ScheduleService.isLikelyGroupFormat(text)) {
                await this.bot.sendMessage(chatId, 'Неверный формат номера группы. Номер должен содержать 4 цифры. Введите снова:');
```

должен содержать 4 цифры. Введите снова:');

```

                return;
            }

            await this.bot.sendMessage(chatId, `Проверяю группу ${text}...`);
            const exists = await this.api.verifyGroup(text);
            if (!exists) {

```

```

        await this.bot.sendMessage(chatId, `Группа "${text}" не найдена. Проверь
номер и введи ещё раз:`);
        return;
    }

    this.users.set(chatId, { group: text });
    await this.bot.sendMessage(chatId, `✅ Группа сохранена: ${text}`);
    return this.sendMenu(chatId);
}

try {
    if (text === '📅 День недели') {
        return this.bot.sendMessage(chatId, 'Выбери день:',
KeyboardService.days());
    }

    if (text === '📅 Вся неделя') {
        this.users.set(chatId, { selectedDay: undefined });
        return this.bot.sendMessage(chatId, 'Выбери тип недели для расписания на
всю неделю:', KeyboardService.week());
    }

    if (text === '📄 Экзамены') {
        return this.sendExams(chatId);
    }

    if (text === '🕒 Ближайшая пара') {
        return this.sendNearestLesson(chatId);
    }

    if (text === '📅 Сегодня') {
        const today = new Date();
        return this.sendDay(
            chatId,
            ScheduleService.jsDayToIndex(today.getDay()),
            ScheduleService.getWeekTypeForDate(today)
        );
    }

    if (text === '📅 Завтра') {
        const tomorrow = new Date();
        tomorrow.setDate(tomorrow.getDate() + 1);
        return this.sendDay(
            chatId,
            ScheduleService.jsDayToIndex(tomorrow.getDay()),
            ScheduleService.getWeekTypeForDate(tomorrow)
        );
    }

    if (ScheduleService.DAYS.includes(text)) {

```

```

        this.users.set(chatId, { selectedDay: ScheduleService.DAYS.indexOf(text)
    });
    return this.bot.sendMessage(chatId, `Выбран(а) ${text}. Теперь выбери тип
недели:`, KeyboardService.week());
}

if (text === 'Чётная неделя' || text === 'Нечётная неделя') {
    const weekType = text.startsWith('Чёт') ? 2 : 1;
    const currentState = this.users.get(chatId);

    if (typeof currentState.selectedDay === 'number') {
        return this.sendDay(chatId, currentState.selectedDay, weekType, true);
    } else {
        return this.sendWeek(chatId, weekType);
    }
}

if (text === '<< Назад') {
    return this.sendMenu(chatId);
}

await this.bot.sendMessage(chatId, 'Неизвестная команда. Выбери действие:',
KeyboardService.main());
} catch (e) {
    console.error('message handler error', e);
    await this.bot.sendMessage(chatId, 'Произошла ошибка. Попробуйте снова.');
```

```

    await this.sendMenu(chatId);
}
});
}
}

const app = express();
const PORT = process.env.PORT || 3000;

app.get('/', (req, res) => res.send('Ботик работает как свинские часы'));
app.listen(PORT, () => console.log(`HTTP server listening on port ${PORT}`));

// ===== Запуск бота =====
new BotApp();
console.log('Успешный запуск бота!!!');
```