ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Розробка та розгортання Discord-бота для відстеження розкладу занять у ХНУРЕ

Кваліфікаційна робота

Перший (бакалаврський рівень)

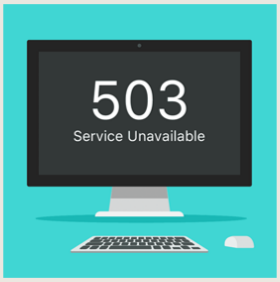
Автор:
ст. групи КІУКІ-21-2
Якущенко Д.О.

Керівник:
асист. Михайліченко І.В.

1

АКТУАЛЬНІСТЬ ТЕМИ

Відсутність повідомлень у існуючих програмах



Відсутність стабільного доступу до розкладу на ПК

Популярність Discord серед студентів

2

ІСНУЮЧІ РІШЕННЯ

1. cist.nure.ua

2. Nure Timetable (Android)

3. KNURE Timetable (iOS)

3

ІСНУЮЧІ РІШЕННЯ

Критерій	1	2	3
Платформа/середовище	iOS		Браузер
Мова інтерфейсу	Англійська, російська	Українська, англійська	Українська
Нагадування про пари	-	-	-
Потреба в установці	+	+	-
Швидкість доступу	Швидка	Швидка	Низька
Стабільність роботи	+	+	-

4

РОЗРОБКА DISCORD-БОТА З РОЗКЛАДОМ ЗАНЯТЬ

- Отримання розкладу та груп з cist.nure.ua
- Обробка та збереження інформації в базу даних
- Реалізація зручних команд для запиту розкладу
- Автоматичне нагадування про пари
- Збереження налаштувань користувача

5

ВИКОРИСТАНІ ТЕХНОЛОГІЇ

Node.js

Discord.js

SQLite

6

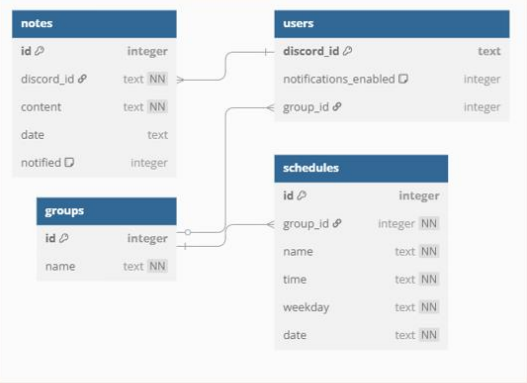
СТРУКТУРА ПРОЄКТУ

- bot.js — основний файл запуску бота
- parser.js — парсер груп з вебсайту
- schedule.js — парсер розкладу з вебсайту
- nure.db — база даних SQLite

- /commands — окрема папка з командами
  - допомога.js
  - розклад.js
  - нотатка.js
  - нотатки.js
  - повідомлення.js
  - видалити.js

7

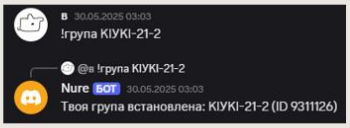
СТРУКТУРА БАЗИ ДАНИХ



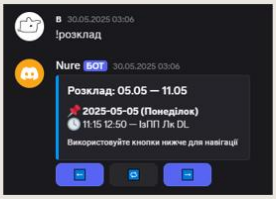
8

ФУНКЦІОНАЛ БОТА

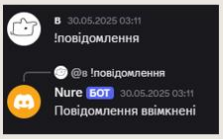
!група



!розклад



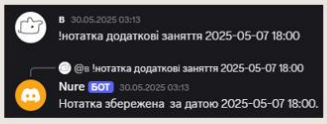
!повідомлення



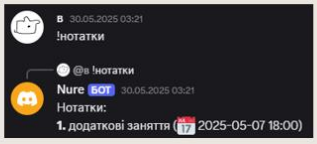
9

ФУНКЦІОНАЛ БОТА

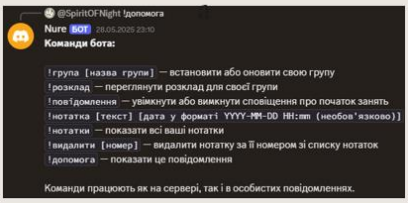
!нотатка



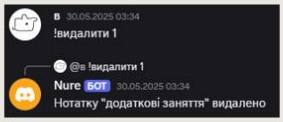
!нотатки



!допомог



!видалити



10

ВИСНОВКИ

- Проаналізовано актуальність та проблематику доступу до розкладу занять в університеті
- Проаналізовано існуючі засоби перегляду розкладу та визначено їх обмеження
- Розроблено Discord-бота, що надає розклад у зручній формі та підтримує функцію сповіщень
- Використано javascript, node.js, discord.js, SQLite
- Тестування функціоналу проведено на реальних прикладах розкладу

ДОДАТОК Б

Вихідний код

Б.1 Модель бота

```
const { Client, GatewayIntentBits, Partials } =
require('discord.js');
const fs = require('fs');
const path = require('path');
const sqlite3 = require('sqlite3').verbose();
const { parse, parseISO, isSameMinute, addMinutes } =
require('date-fns');

const CHECK_INTERVAL = 60 * 1000;

const client = new Client({
  intents: [
    GatewayIntentBits.Guilds,
    GatewayIntentBits.GuildMessages,
    GatewayIntentBits.DirectMessages,
    GatewayIntentBits.MessageContent,
  ],
  partials: [Partials.Channel],
});

client.commands = new Map();
const commandsPath = path.join(__dirname, 'commands');
const commandFiles = fs.readdirSync(commandsPath).filter(file =>
file.endsWith('.js'));

for (const file of commandFiles) {
  const command = require(`./commands/${file}`);
  client.commands.set(command.name, command);
}

client.once('ready', () => {
  console.log(`Бот ${client.user.tag} запущен`);

  setInterval(() => {
    checkForUpcomingLessons(client);
    checkForNotes(client);
  }, CHECK_INTERVAL);
});

client.on('messageCreate', async (message) => {
  if (message.author.bot || !message.content.startsWith('!'))
return;
```

```
  const args = message.content.slice(1).trim().split(/ +/);
  const commandName = args.shift().toLowerCase();
  const command = client.commands.get(commandName);

  if (!command) return;

  try {
    await command.execute(message, args);
  } catch (error) {
    console.error(error);
    message.reply('Помилка при виконанні команди');
  }
});

function checkForUpcomingLessons(client) {
  const db = new sqlite3.Database('./nure.db');
  const now = new Date();

  db.all(
    `SELECT u.discord_id, s.name, s.time, s.date
     FROM users u
     JOIN schedules s ON u.group_id = s.group_id
     WHERE u.notifications_enabled = 1`,
    async (err, rows) => {
      if (err) {
        console.error('DB error (lessons):', err);
        db.close();
        return;
      }

      for (const row of rows) {
        const [startTime] = row.time.split(' ');
        const lessonDateTime = parse(`${row.date} ${startTime}`,
'yyyy-MM-dd HH:mm', new Date());

        if (isSameMinute(lessonDateTime, now)) {
          try {
            const user = await
client.users.fetch(row.discord_id);
            await user.send(`Заняття: **${row.name}** о
${startTime}`);
          } catch (e) {
            console.error(`Не вдалося надіслати повідомлення
${row.discord_id}:`, e.message);
          }
        }
      }

      db.close();
    }
  );
}
```

```
function checkForNotes(client) {
  const db = new sqlite3.Database('./nure.db');
  const now = new Date();

  db.all(
    `SELECT id, discord_id, content, date
     FROM notes
     WHERE date IS NOT NULL AND notified = 0`,
    async (err, rows) => {
      if (err) {
        console.error('DB error (notes):', err);
        db.close();
        return;
      }

      for (const note of rows) {
        const noteDate = parseISO(note.date);
        if (isSameMinute(noteDate, now)) {
          try {
            const user = await
client.users.fetch(note.discord_id);
            await user.send(`Нагадування: "${note.content}"`);
            db.run('UPDATE notes SET notified = 1 WHERE id = ?',
[note.id]);
          } catch (e) {
            console.error(`Не вдалося нагадати
${note.discord_id}:`, e.message);
          }
        }
      }

      db.close();
    }
  );
}

client.login('TOKEN');
```

Б.2 Парсер груп

```
const puppeteer = require('puppeteer');
const Database = require('better-sqlite3');

const db = new Database('nure.db');

const createTable = db.prepare(`
  CREATE TABLE IF NOT EXISTS groups (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL
  )
```

```
`);
createTable.run();

const insertGroup = db.prepare(`
  INSERT OR IGNORE INTO groups (id, name) VALUES (?, ?)
`);

async function scrapeGroups() {
  try {
    const browser = await puppeteer.launch({ headless: true });
    const page = await browser.newPage();

    await page.goto('https://cist.nure.ua/ias/app/tt/f?p=778:2',
{
      waitUntil: 'domcontentloaded',
    });

    await
page.waitForSelector('a[onclick*="IAS_Change_Groups("]');

    const faculties = await page.evaluate(() => {
      const facultiesArray = [];
      const facultyLinks =
document.querySelectorAll('a[onclick*="IAS_Change_Groups("]');

      facultyLinks.forEach(faculty => {
        facultiesArray.push({
          id:
faculty.getAttribute('onclick').match(/IAS_Change_Groups\((\d+)\
)/)[1],
          name: faculty.textContent.trim()
        });
      });

      return facultiesArray;
    });

    for (let i = 0; i < faculties.length; i++) {
      const faculty = faculties[i];
      const facultyUrl =
`https://cist.nure.ua/ias/app/tt/WEB_IAS_TT_AJX_GROUPS?p_id_fac=
${faculty.id}`;

      await page.goto(facultyUrl);
      await
page.waitForSelector('a[onclick*="IAS_ADD_Group_in_List("]');

      const groups = await page.evaluate(() => {
        const groupsArray = [];
        const groupLinks =
document.querySelectorAll('a[onclick*="IAS_ADD_Group_in_List("]'
);
```

```
        groupLinks.forEach(group => {
          const match =
group.getAttribute('onclick').match(/IAS_ADD_Group_in_List\('(.+
?)',(\d+)\)/);
          if (match) {
            groupsArray.push({
              name: match[1],
              id: parseInt(match[2])
            });
          }
        });

        return groupsArray;
      });

      for (const group of groups) {
        insertGroup.run(group.id, group.name);
      }

      console.log(`Групи для факультета ${faculty.name}:`,
groups.length);
    }

    await browser.close();
    console.log('Усі групи завантажені в базу данних');
  } catch (error) {
    console.error('Помилка при парсингу:', error.message);
  }
}

scrapeGroups();
```

Б.3 Парсер розкладу

```
const puppeteer = require("puppeteer");
const sqlite3 = require("sqlite3").verbose();
const { format, startOfMonth, endOfMonth, startOfWeek, addDays,
subMonths, addMonths, parse } = require("date-fns");

const groupId = process.argv[2];

if (!groupId) {
  console.error("Не вказан groupId");
  process.exit(1);
}

function getWeekRanges(startDate, endDate) {
  const weeks = [];
  let current = startOfWeek(startDate, { weekStartsOn: 1 });

  while (current <= endDate) {
```

```
    const weekStart = format(current, "dd.MM.yyyy");
    const weekEnd = format(addDays(current, 6), "dd.MM.yyyy");
    weeks.push({ start: weekStart, end: weekEnd });
    current = addDays(current, 7);
  }

  return weeks;
}

async function parseWeekSchedule(startDate, endDate, page) {
  const url =
`https://cist.nure.ua/ias/app/tt/f?p=778:201:845728355434122::::
P201_FIRST_DATE,P201_LAST_DATE,P201_GROUP,P201_POTOK:${startDate
},${endDate},${groupId},0:`;
  await page.goto(url, { waitUntil: "networkidle2" });

  try {
    await page.waitForSelector("table.MainTT", { timeout: 10000
});
  } catch {
    console.warn(`Нема розкладу на тиждень ${startDate} —
${endDate}`);
    return [];
  }

  const schedule = await page.evaluate(() => {
    const rows =
Array.from(document.querySelectorAll("table.MainTT tr"));
    const data = [];
    let currentDay = null;
    let currentDate = null;

    for (let i = 0; i < rows.length; i++) {
      const cells = rows[i].querySelectorAll("td");

      if (cells.length === 2) {
        const maybeDay = cells[0].innerText.trim();
        const maybeDate = cells[1].innerText.trim();
        if (maybeDay.match(/[a-яA-ЯҐЄІЇ]/) &&
maybeDate.match(/\d{2}\.\d{2}\.\d{4}/)) {
          currentDay = maybeDay;
          currentDate = maybeDate;
        }
      }

      if (cells.length === 3 && currentDay && currentDate) {
        const time = cells[1].innerText.trim();
        const title = cells[2].innerText.trim();
        if (title) {
          data.push({
            name: title,
            time: time,
            weekday: currentDay,
```

```
            date: currentDate
          });
        }
      }
    }

    return data;
  });

  return schedule.map(item => {
    const parsed = parse(item.date, "dd.MM.yyyy", new Date());
    return {
      name: item.name,
      time: item.time,
      weekday: item.weekday,
      date: format(parsed, "yyyy-MM-dd")
    };
  });
}

(async () => {
  const now = new Date();
  const start = startOfMonth(subMonths(now, 1));
  const end = endOfMonth(addMonths(now, 1));

  const weeks = getWeekRanges(start, end);

  const db = new sqlite3.Database('./nure.db', (err) => {
    if (err) {
      console.error('Помилка при підключені до бази данних',
err.message);
    } else {
      console.log('База данних підключена');
    }
  });

  db.run(`
    CREATE TABLE IF NOT EXISTS schedules (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      group_id INTEGER,
      name TEXT,
      time TEXT,
      weekday TEXT,
      date TEXT,
      FOREIGN KEY(group_id) REFERENCES groups(id)
    )
  `);

  const browser = await puppeteer.launch({
    headless: "new",
    args: ["--no-sandbox", "--disable-setuid-sandbox"]
  });
```

```
  const page = await browser.newPage();

  for (const week of weeks) {
    console.log(`Парсинг: ${week.start} — ${week.end}`);
    const weekData = await parseWeekSchedule(week.start,
week.end, page);

    for (const item of weekData) {
    const { name, time, weekday, date } = item;
    db.get(
      `SELECT id FROM schedules WHERE group_id = ? AND date = ?
AND time = ?`,
      [groupId, date, time],
      (err, row) => {
        if (err) {
          console.error("Помилка перевірки існуючого запису:",
err.message);
        } else if (!row) {
          db.run(
            `INSERT INTO schedules (name, time, weekday, date,
group_id) VALUES (?, ?, ?, ?, ?)`,
            [name, time, weekday, date, groupId],
            (err) => {
              if (err) {
                console.error("Помилка вставки:", err.message);
              }
            }
          );
        } else {
          console.log(`Запис вже існує: ${groupId}, ${date},
${time}`);
        }
      }
    );
    }
    }

  await browser.close();
  db.close();
  console.log("Розклад для групи завантажено в базу данних");
})();
```

Б.4 Команда "видалити"

```
const sqlite3 = require('sqlite3').verbose();

module.exports = {
  name: 'видалити',
  description: 'Видалити свою нотатку за номером',

  async execute(message, args) {
```

```
    if (args.length !== 1 || isNaN(args[0])) {
      return message.reply('Введи номер нотатки для
видалення`');
    }

    const index = parseInt(args[0]);
    const userId = message.author.id;

    const db = new sqlite3.Database('./nure.db');

    db.all(
      'SELECT id, content FROM notes WHERE discord_id = ? ORDER
BY date IS NULL, date ASC',
      [userId],
      (err, rows) => {
        if (err) {
          console.error('DB error:', err);
          message.reply('Помилка бази даних.');
          db.close();
          return;
        }

        if (index < 1 || index > rows.length) {
          message.reply('Такої нотатки не існує!');
          db.close();
          return;
        }

        const noteToDelete = rows[index - 1];
        db.run('DELETE FROM notes WHERE id = ?',
[noteToDelete.id], function (err) {
          if (err) {
            console.error('Delete error:', err);
            message.reply('Не вдалося видалити нотатку');
          } else {
            message.reply(`Нотатку "${noteToDelete.content}"
видалено`);
          }
          db.close();
        });
      }
    );
  }
};
```

Б.5 Команда "група"

```
const sqlite3 = require('sqlite3').verbose();
const db = new sqlite3.Database('nure.db');
const { exec } = require('child_process');
```

```
module.exports = {
name: 'група',
async execute(message, args) {
const groupName = args.join(' ').trim();
if (!groupName) {
    return message.reply('Вкажи назву групи: !група ``назва
групи``');
  }

  db.get('SELECT id FROM groups WHERE name = ?', [groupName],
(err, row) => {
    if (err) {
      console.error('DB error:', err);
      return message.reply('Помилка доступу до бази даних');
    }

    if (!row) {
      return message.reply(`Група "${groupName}" не знайдена`);
    }

    const groupId = row.id;
    const userId = message.author.id;

    db.run(
      'INSERT INTO users (discord_id, group_id) VALUES (?, ?) ON
CONFLICT(discord_id) DO UPDATE SET group_id =
excluded.group_id',
      [userId, groupId],
      (err) => {
        if (err) {
          console.error('DB insert error:', err);
          return message.reply('Не вдалося зберегти групу.');
        }

        message.reply(`Твоя група встановлена: ${groupName} (ID
${groupId})`);


        exec(`node schedule.js ${groupId}`, (error, stdout,
stderr) => {
          if (error) {
            console.error(`Error while starting schedule.js:
${error.message}`);
            return;
          }
          if (stderr) {
            console.error(`stderr: ${stderr}`);
          }
          console.log(`schedule.js output:\n${stdout}`);
        });
      }
    );
  });
```

```
}
};
```

Б.6 Команда "допомога"

```
module.exports = {
  name: 'допомога',
  description: 'Показати список доступних команд',

  async execute(message) {
    const help = `
**Команди бота:**

\`!група [назва групи]\` — встановити або оновити свою групу
\`!розклад\` — переглянути розклад для своєї групи
\`!повідомлення\` — увімкнути або вимкнути сповіщення про
початок занять
\`!нотатка [текст] [дата у форматі YYYY-MM-DD HH:mm
(необов'язково)]\` — створити нову нотатку. Якщо вказати дату,
то бот нагадає про неї
\`!нотатки\` — показати всі ваші нотатки
\`!видалити [номер]\` — видалити нотатку за її номером зі списку
нотаток
\`!допомога\` — показати це повідомлення

Команди працюють як на сервері, так і в особистих повідомленнях.
    `;
    message.reply(help);
  }
};
```

Б.7 Команда "нотатка"

```
const sqlite3 = require('sqlite3').verbose();

module.exports = {
  name: 'нотатка',
  async execute(message, args) {
    const db = new sqlite3.Database('./nure.db');
    const discordId = message.author.id;
    const input = args.join(' ');
    if (!input) return message.reply('Напиши текст для додання
нотатки!');

    const dateMatch = input.match(/(\d{4}-\d{2}-\d{2}
\d{2}:\d{2})$/);
    let date = null;
    let content = input;
```

```
    if (dateMatch) {
      date = dateMatch[1];
      content = input.replace(date, '').trim();
    }

    db.run(
      'INSERT INTO notes (discord_id, content, date) VALUES (?,
?, ?)',
      [discordId, content, date],
      function (err) {
        if (err) {
          console.error(err);
          message.reply('Не вдалося зберігти нотатку до бд');
        } else {
          message.reply(`Нотатка збережена ${date ? ` за датою
${date}` : ''}.`);
        }
      }
    );

    db.close();
  }
};
```

Б.8 Команда "нотатки"

```
const sqlite3 = require('sqlite3').verbose();

module.exports = {
  name: 'нотатки',
  async execute(message) {
    const db = new sqlite3.Database('./nure.db');
    const discordId = message.author.id;

    db.all(
      'SELECT content, date FROM notes WHERE discord_id = ?
ORDER BY date IS NULL, date',
      [discordId],
      (err, rows) => {
        if (err) {
          console.error(err);
          message.reply('❌ Ошибка при получении заметок.');
        } else if (rows.length === 0) {
          message.reply('Нотаток немає');
        } else {
          const formatted = rows
            .map((r, i) => `**${i + 1}.** ${r.content}${r.date ?
` (🗓️ ${r.date})` : ''}`)
            .join('\n');
```

```
          message.reply(`Нотатки:\n${formatted}`);
        }
        db.close();
      }
    );
  }
};
```

Б.9 Команда "повідомлення"

```
const sqlite3 = require('sqlite3').verbose();

module.exports = {
  name: 'повідомлення',
  description: 'Ввімкнути/вимкнути повідомлення про початок
занять',

  async execute(message, args) {
    const db = new sqlite3.Database('./nure.db');
    const userId = message.author.id;

    db.get('SELECT notifications_enabled FROM users WHERE
discord_id = ?', [userId], (err, row) => {
      if (err || !row) {
        return message.reply('Треба встановити групу `!група`');
      }

      const newValue = row.notifications_enabled ? 0 : 1;

      db.run('UPDATE users SET notifications_enabled = ? WHERE
discord_id = ?', [newValue, userId], err => {
        if (err) {
          return message.reply('Не вдалося оновити
налаштування.');
        }

        message.reply(newValue
          ? 'Повідомлення ввімкнені'
          : 'Повідомлення вимкнені');
      });
    });
  }
};
```

Б.10 Команда "розклад"

```
const { ActionRowBuilder, ButtonBuilder, ButtonStyle,
EmbedBuilder } = require('discord.js');
```

```
const sqlite3 = require('sqlite3').verbose();
const { format, startOfWeek, addDays, subWeeks, addWeeks } =
require('date-fns');
const { exec } = require('child_process');

module.exports = {
  name: 'розклад',
  description: 'Показати розклад із кнопками навігації та
оновлення',

  async execute(message, args, client) {
    const db = new sqlite3.Database('./nure.db');

    db.get('SELECT group_id FROM users WHERE discord_id = ?',
[message.author.id], async (err, row) => {
      if (err || !row) {
        return message.reply('Спочатку задай групу за допомогою
команди `!група`.');
      }

      const groupId = row.group_id;
      let weekOffset = 0;
      let lastMessage = null;

      const showSchedule = async (channel, userId) => {
        const baseDate = new Date();
        const monday = startOfWeek(addWeeks(baseDate,
weekOffset), { weekStartsOn: 1 });
        const sunday = addDays(monday, 6);
        const startStr = format(monday, 'yyyy-MM-dd');
        const endStr = format(sunday, 'yyyy-MM-dd');

        db.all(
          'SELECT * FROM schedules WHERE group_id = ? AND date
BETWEEN ? AND ? ORDER BY date, time',
          [groupId, startStr, endStr],
          async (err, rows) => {
            if (err) {
              console.error(err);
              return channel.send('Помилка при отриманні
розкладу.');
            }

            const embed = new EmbedBuilder()
              .setTitle(`Розклад: ${format(monday, 'dd.MM')} —
${format(sunday, 'dd.MM')}`)
              .setColor(0x0099ff)
              .setFooter({ text: 'Використовуйте кнопки нижче
для навігації' });

            embed.setDescription(
              rows.length === 0
                ? 'Немає занять на цьому тижні.'
```

```
                : rows.map(r => `📌 **${r.date}
(${r.weekday})**\n🕓 ${r.time} — ${r.name}`).join('\n\n')
            );

            const row = new ActionRowBuilder().addComponents(
              new
ButtonBuilder().setCustomId('prev').setLabel('⬅️').setStyle(Butt
onStyle.Primary),
              new
ButtonBuilder().setCustomId('update').setLabel('🔁').setStyle(Bu
ttonStyle.Secondary),
              new
ButtonBuilder().setCustomId('next').setLabel('➡️').setStyle(Butt
onStyle.Primary)
            );

            if (lastMessage) {
              try {
                await lastMessage.delete();
              } catch (e) {
                console.warn('Не вдалося видалити попереднє
повідомлення:', e.message);
              }
            }

            lastMessage = await channel.send({ embeds: [embed],
components: [row] });

            const collector =
lastMessage.createMessageComponentCollector({
              time: 5 * 60 * 1000
            });

            collector.on('collect', async interaction => {
              if (interaction.user.id !== userId) {
                return interaction.reply({ content: 'Це не твій
розклад.', ephemeral: true });
              }

              await interaction.deferUpdate();

              if (interaction.customId === 'prev') {
                weekOffset--;
                showSchedule(channel, userId);
              } else if (interaction.customId === 'next') {
                weekOffset++;
                showSchedule(channel, userId);
              } else if (interaction.customId === 'update') {
                await interaction.editReply({ content:
'Оновлення розкладу…', embeds: [], components: [] });

                exec(`node schedule.js ${groupId}`, (error,
```

```
stdout, stderr) => {
                  if (error) {
                    console.error(`schedule.js error:
${error.message}`);
                    return interaction.editReply('Не вдалося
поновити розклад.');
                  }

                  if (stderr) console.warn(stderr);
                  console.log(stdout);
                  showSchedule(channel, userId);
                });
              }
            });
          }
        );
      };

      showSchedule(message.channel, message.author.id);
    });
  }
};
```