МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕЛЕГРАМ-БОТУ ДЛЯ ВЗАЄМОДІЇ ЗІ СТУДЕНТАМИ ЗНУ»**

Виконав: студент 2 курсу, групи 8.1221-з
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
О. О. Сердюк
(ініціали та прізвище)

Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н.Решевська К.С..
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри інформаційних технологій в туризмі НУЗП, доцент, к.т.н. Морозов Д.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук , к.т.н., доцент

_____ Чопоров С.В.
(підпис)

« ____ » _____ 2022 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Сердюку Олексію Олеговичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розробка програмного забезпечення Телеграм-боту для взаємодії зі студентами ЗНУ

керівник роботи (проекту) доцент кафедри комп'ютерних наук, доцент к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)
Решевська К.С.
затверджені наказом ЗНУ від « 10 » травня 2022 року № 514-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка телеграм-боту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 11.05.2022

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	06.05.2022	
2.	Збір вихідних даних.	01.09.2022	
3.	Обробка літературних джерел	5.10.2022	
4.	Розробка першого та другого розділу.	18.10.2022	
5.	Розробка третього розділу.	05.11.2022	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи магістра.		

Студент _____ О.О. Сердюк
(підпис) (ініціали та прізвище)

Керівник роботи _____ К.С. Решевська
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота магістра: «Розробка програмного забезпечення Телеграм-боту для взаємодії зі студентами ЗНУ»: 57 с., 5 рис., 11 джерел.

CHAT-BOT, FLASK, POSTGRESQL, PYTHON, REST API, STUDENTS, TELEGRAM, TIMETABLE

Об'єкт дослідження – архітектура взаємодії клієнтської та серверної складових чат-ботів.

Предмет дослідження – REST API для Telegram-боту.

Мета роботи – розробка REST API для збереження даних розкладу начальних груп Запорізького національного університету на сервері для організації роботи Telegram-боту, що спілкується зі студентами.

В результаті виконання кваліфікаційної роботи вирішено наступні задачі: досліджено предметну область, проведено порівняння і аналіз наявних аналогів Інтернет-ботів, вивчено архітектуру REST API, спроєкторвано базу даних для збереження даних про розклад та користувачів, розроблено програмне забезпечення для зберігання даних про розклад навчального закладу з Excel-файлу у базі даних системи керування базами даних PostgreSQL та надання програмного інтерфейсу для організації роботи Telegram-боту.

SUMMARY

Bachelor’s qualifying paper: "Development of Telegram-bot software for interaction with students of ZNU": 51 pages, 24 figures, 11 sources.

CHAT-BOT, FLASK, POSTGRESQL, PYTHON, REST API, STUDENTS, TELEGRAM, TIMETABLE

The object of the study is the architecture of the mutual modality of the client and server warehouse chatbots.

The subject of research is the REST API for the Telegram bot.

The aim of the study is development of a REST API for saving data on the schedule of initial groups of the Zaporizhia National University on a server for organizing the work of a Telegram bot that communicates with students.

As a result of the qualification work, the following tasks were solved: the subject area was researched, a comparison and analysis of existing analogues of Internet bots was carried out, the REST API architecture was studied, a database was designed for saving data about the schedule and users, software was developed for storing data about the schedule of an educational institution with Excel file in the database of the PostgreSQL database management system and providing a software interface for organizing the work of the Telegram bot.

ВСТУП

У сучасному світі для спілкування, розваг, роботи та навчання повсюдно використовується Інтернет. Інтернет є не тільки джерелом різноманітної та корисної для користувачів інформації, але також є основною формою віртуального спілкування. Завдяки Інтернету тепер не потрібна постійна особиста присутність, багато питань вирішуються через засоби віртуальної комунікації. Багато в чому цьому сприяли системи миттєвого обміну повідомленнями – месенджери. Месенджери забезпечують користувачів великою кількістю зручних способів онлайн-спілкування як з родичами і друзями, так і з діловими партнерами. Проведення онлайн-опитувань і голосувань є одним з найбільш популярних засобів швидкої комунікації. Голосування та опитування дозволяють в інтерактивному режимі організувати процес опитування і збору інформації у певної групи людей.

Мета роботи – розробка REST API для збереження даних розкладу начальних груп Запорізького національного університету на сервері для організації роботи Telegram-боту, що спілкується зі студентами.

Об'єкт дослідження – архітектура взаємодії клієнтської та серверної складових чат-ботів.

Предмет дослідження – REST API для Telegram-боту.

Розроблений в рамках кваліфікаційної роботи REST API надасть можливість автоматизованого аналізу Excel-файлу із розкладом навчального процесу Запорізького національного університету для збереження їх у базі даних на вебсервері та передання цих даних до Telegram-боту.

Актуальність і практична значимість кваліфікаційної роботи обумовлена високою популярністю месенджерів і чат-ботів серед студентів. Також наявність засобу автоматизованої відповіді на питання пов’язані із розкладом навчального процесу значно зменшить час спілкування зі студентами куратора групи, особливо зараз, коли навчання відбувається он-лайн, а вчорашні абітурієнти і сьогоднішні

студенти губляться у потоці інформації стосовно організації навчального процесу.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

– дослідження предметної області;

– провести порівняльний аналіз наявних аналогів чат-ботів;

– вивчити архітектуру REST API;

– спроєктувати базу даних для збереження даних про розклад та користувачів;

– розробити програмне забезпечення для зберігання даних про розклад навчального закладу з Excel-файлу у базі даних системи керування базами даних PostgreSQL та надання програмного інтерфейсу для організації роботи Telegram-боту.

1 ІНТЕРНЕТ-БОТИ І ТЕХНОЛОГІЇ ЇХ РОЗРОБКИ

1.1 Дослідження предметної області

Telegram – це безкоштовний месенджер для смартфонів та персональних комп'ютерів під управлінням всіх найбільш поширених на сьогоднішній день операційних систем, що дозволяє обмінюватися не тільки текстовими повідомленнями, але і різними медіафайлами (картинки, музика, архіви, текстові документи) [1].

Функціонально Telegram схожий на інші месенджери, його головні переваги перед іншими месенджерами – швидкість, захищеність, зберігання даних у хмарі (віддалений сервер) та безкоштовність.

Свій трафік Telegram надійно зашифровує, усі обчислення виконуються на віддаленому сервері, а клієнтська частина тільки шифрує дані і відправляє їх на сервер. Для стійкої роботи було створено унікальний протокол MTProto, що суттєво підвищило безпеку та захист від несанкціонованої втрати інформації.

Боти [1] – спеціальна програма, що виконує автоматично і/або за заданим розкладом які-небудь дії через ті ж інтерфейси, що й звичайний користувач. Зазвичай боти призначаються для виконання роботи, одноманітної й повторюваної, з максимально можливою швидкістю (очевидно, набагато вищою за можливості людини). . Написані для платформи Telegram , вони призначені для виконання самих різних функцій: від отримання новин до пошуку інформації та навіть торгівлі акціями. Головне завдання бота є автоматична відповідь після введеної користувачем команди. При цьому, працюючи безпосередньо через інтерфейс Telegram, програма імітує дії живого користувача, за рахунок чого користування таким ботом набагато зручніше та зрозуміліше.

Існують наступні різновиди ботів:

– чат-боти являють собою найпростіший чат, що імітує спілкування на задану користувачем тематику;

– боти-інформатори – окремий вигляд ботів, головна ціль яких - інформування користувача про тих або інших подіях(новини, заходи, публікації тощо);

– ігрові боти – це боти, з якими можна грати в різні ігри;

– боти-помічники – це боти, розроблені різними онлайн- сервісами, як доповнення до основної веб-версії.

Алгоритм роботи бот-утиліт полягає у наступному: повідомлення, команди і запити, відправлені користувачами, передаються до програмного забезпечення, що зберігається на серверах розробників. Посередницький анонімний сервер Telegram обробляє шифрування та здійснює зворотній зв'язок між утилітою та користувачем.

Взаємодія між користувачем та ботом виглядає наступним чином: користувач віддає команду боту, бот передає команду на сервер, після цього програма на сервері обробляє отриманий від робота запит, потім сервер віддає відповідь боту, в результаті бот виводить відповідь на екран програми користувачеві. І цей цикл повторюється раз за разом, коли користувач взаємодіє з будь-яким Telegram -ботом.

Взаємодія із серверами відбувається за допомогою простого HTTPS-інтерфейсу, який є спрощеною версією API Telegram . Інакше цей інтерфейс можна назвати програмним каталогом або бот-алгоритмом. Нові бот-утиліти створюються за допомогою спеціальної утиліти @BotFather.

1.2 Огляд популярних Telegram-ботів

Перед початком будь якого проекту слід проаналізувати вже наявні боти. Проаналізувати їх призначення, реалізовані особливості того чи іншого бота, інтерфейс. Також буде дуже корисно порівняти боти, знайти їх переваги і недоліки.

Список та короткий опис наявних популярних ботів у Telegram:

1. Gmail Bot (@GmailBot) – офіційний поштовий клієнт Google у Telegram, який дозволяє відповідати на листи з Gmail не виходячи з Telegram. Дуже корисний з точки зору аналізу та функціоналу бот, який дозволяє подивитися на що ще здатні боти.

2. Stickers (@stickers) – офіційний бот Telegram спеціалізований на створенні стікерів. Потрібно вибрати ім’я стікерпаку, завантажити фото у форматі PNG 512*512 та визначити до якого емодзі прикріпити цей стікер.

3. YouTube (@Save_youbot) – цей бот дозволяє завантажити відео з сервісу YouTube будь якого об’єму прямо у Telegram.

4. Око бога (@crosserr_bot) – цей бот дозволяє проводити пошук по імені, номеру автомобіля, по номеру телефона, по поштовій адресі або навіть по ІПН та видає інформацію по цим даним.

5. Mailsearchbot (@Mailsearchbot) – бот шукає паролі, що відповідають адресам електронної пошти, логінам або номеру телефону. Безкоштовно бот показує лише частину виявлених паролів, а за повною версією відправляє на сторонній сайт, де відвідувачу з ходу пропонують сплатити передплату.

6. ЧастушкиБот (@ChastushkiBot) – принцип роботи цього бота полягає у тому, що користувач відправляє текстове повідомлення боту, а бот у свою чергу підставляє музику і зачитує його у виді частушки. Для пауз можна ставити коми та точки, якщо ви хочете поставити наголос на певну голосну, то можна поставити + перед цією голосною. У відповідь бот присилає файл формату .mp3, тому ви можете завантажити його на ваш пристрій та використати у своєму проєкті;

7. Готовий до всього (@Hotovyi_do_vsioho_bot) – цей бот дозволяє вам по пунктам зібратись на випадок надзвичайної ситуації та бути готовим до будь-якого варіанту розвитку подій. Ви можете знайти відповіді на найпоширеніші запитання щодо підготовки до надзвичайних ситуацій, а також поради, щодо дій у небезпечних ситуаціях. Даний бот відображає сучасність, актуальність та гнучкість Telegram-ботів. Швидко пристосовувався до подій в Україні з 24.02.2022 та під час карантину SARS-COV-2 (COVID-19).

8. Серіали БОТ (@Serialy2021_bot) – цей бот дозволяє шукати серіали та дивитися його в Telegram, бот має зручну навігацію, та дозволяє зберігати серії серіалу на свій пристрій. Концептуально схожий з духом додатку, який я реалізую при виконані дипломного проєкту.

9. BuddyMusic (@Buddy_music_bot) – цей бот дозволяє здійснювати пошук пісень, завантажувати пісні з бази даних бота. Пошук пісень здійснюється по назві самої пісні або по імені автора.

10. psy.for.peace (@Psy_for_peace_bot) – цей бот створений для пошуку спеціаліста у області психології. Тут користувач повинен написати своє ім’я, вік, місто та запит для психолога. Після обробки даних користувачеві пише спеціаліст, або, у разі, якщо у користувача закриті повідомлення від інших користувачів, котрі не у контактах, прийде повідомлення з контактами спеціаліста у Telegram-бот.

1.3 Висновки з першого розділу

Під час виконання першого розділу кваліфікаційної роботи було проведено пошук інформації про роботу ботів, Telegram-ботів та загальні відомості про них.

У цьому розділі були розглянуті основні поняття та призначення Telegram-ботів, створен список різноманітних призначень Telegram-застосунків.

Також була проведена класифікація Telegram-ботів та зроблено висновок, що класифікувати Telegram-ботів наразі є важливим кроком при їх дослідженні, бо ця гілка індустрії програмування дуже швидко розвивається, додаються нові технології та можливості.

Було проведено огляд серед інших телеграм ботів, на основі списку ботів було обрано технології виконання власного проєкту. Серед оглянутих ботів є схожі за призначенням проєкти, але немає аналога Telegram-боту, який надаватиме студентам інформацію про розклад навчального закладу.

2 REST API ДЛЯ ІНФОРМАЦІЙНОГО TELEGRAM-БОТУ

2.1 Програмний інтерфейс додатку

Інтерфейс прикладного програмування (API – Application Programming Interface) представляє собою набір правил і функцій, які дозволяють двом різним додаткам взаємодіяти один з одним. Подібні інтерфейси забезпечують інтеграцію додатків, що дає можливість розробникам створювати потужні цифрові рішення.

API виступає посередником між додатками, які надсилають запити та відповіді. Наприклад, реєстрація у додатку через існуючий обліковий запис користувача Twitter відбувається через API Twitter, який розробники інтегрували в додаток.

Хронологія розвитку API представлено на рисунку 2.1.

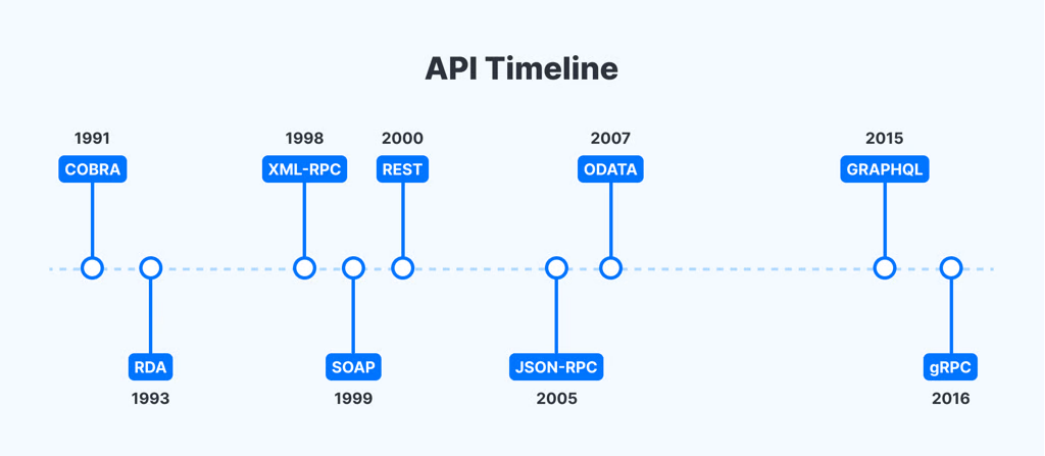

Рисунок 2.1 – Хронологія появи форматів API

Для відправлення і отримання запитів (функцій) в залежності від призначення API слідує різним протоколам і стандартам. Розрізняють наступні типи протоколів і архітектур:

1. XML-RPC – дозволяє виконувати обмін функціями між двома або більше мережами. XML-RPC використовує XML для опису запитів і відповідей, а також

за допомогою протоколів HTTP передає інформацію від клієнта на сервер.

2. JSON-RPC – це полегшений RPC, схожий на XML. Протокол закодований в JSON, дозволяє отримувати запит на сервер з можливістю асинхронних відповідей.

3. SOAP – простий протокол доступу до об'єктів — протокол для обміну структурованою інформацією при реалізації веб-сервісів у комп'ютерних мережах. SOAP використовує XML для автентифікації, авторизації та взаємодії процесів в операційних системах і дозволяє клієнтам викликати веб-сервіси та отримувати відповіді незалежно від мови та платформи.

4. REST API – репрезентативна передача стану (REST) — це архітектурний клієнт, який здійснює реалізацію і сервер незалежно один від одного. Сервіси в REST API взаємодіють по протоколу HTTP.

2.2 Архітектурний стиль взаємодії компонентів вебсервісу

REST або Representational state transfer – це архітектурний стиль проєктування API з використанням протоколу HTTP. Основна перевага REST – велика гнучкість.

REST API застосовується в тому випадку, якщо є необхідність надання даних із сервера веб-додатків або сайту.

Головними компонентами REST API є (рис.2.2):

Клієнт – клієнт або програма, яка запускається на стороні користувача (на його пристрої) та ініціює комунікацію.

Сервер – сервер, який надає API для якісного доступу до своїх даних і функцій.

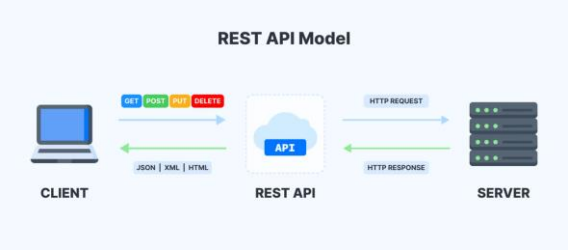

Рисунок 2.2 – Архітектура взаємодії клієнту і серверу

Ресурс – ресурс представляє собою будь-який вид контенту (відео, текст, картинка), який сервер може передавати клієнту. REST API взаємодіє за допомогою HTTP-запитів, виконуючи стандартні функції: створення, оновлення, читання, видалення записів на ресурсі. Існує чотири методи, які описують, що потрібно робити з ресурсом (рис 2.3):

– POST — створення ресурсу;

– GET — отримання ресурсу;

– PUT — оновлення ресурсу;

– DELETE — видалення ресурсу.

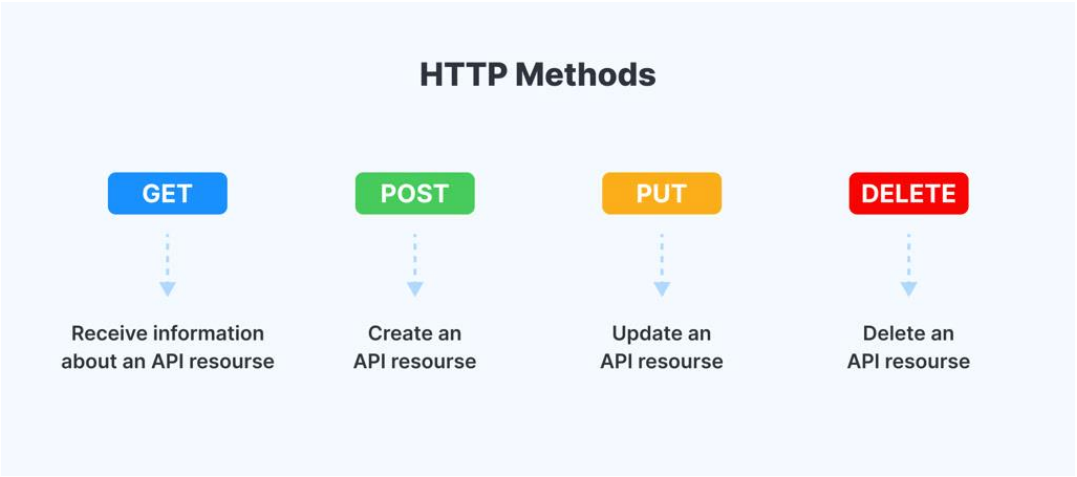

Рисунок 2.3 – Методи взаємодії клієнту із сервером

Технологію REST API застосовують у вебдодатку, де користувачі повинні надавати та отримувати дані на сервері та з нього. В даний час це найпоширеніший спосіб організації API. Він витіснив раніше популярні способи SOAP і WSDL.

У RESTful немає єдиного стандарту роботи: його називають «архітектурним стилем» для роботи з серверами. Такий підхід у 2000 році у своїй дисертації виклав програміст і дослідник Рой Філдінг, один із творців протоколу HTTP.

2.3 Принципи REST API

У RESTful є 7 принципів написання коду інтерфейсів.

1. Відокремлення клієнта від сервера (клієнт-сервер). Клієнт – це інтерфейс користувача сайту або додатку. В REST API код запитів залишається на стороні клієнта, а код для доступу до даних – на стороні серверу. Це покращує API організації, дозволяє легко переносити інтерфейс користувача на іншу платформу та дає можливість краще масштабувати серверне зберігання даних.

2. Відсутність запису стану клієнта (без стану). Сервер не повинен зберігати інформацію про стан (виконаних операцій) клієнта. Кожен запит від клієнта повинен містити тільки ту інформацію, яка потрібна для отримання даних від сервера.

3. Кешуємість (Casheable). У запиті даних повинно бути зазначено, що потрібно кешувати дані (зберігати в спеціальному буфері для частих запитів). Якщо така вказівка є, клієнт отримає право звернутися до цього буферу за необхідності.

4. Єдність інтерфейсу (Uniform Interface). Усі дані повинні запитуватися через одну URL-адресу стандартними протоколами, наприклад, HTTP. Це покращує архітектуру додатків і робить взаємодію з сервером більш корисною.

5. Багаторівневість системи (Layered System). Сервер RESTful може розташовуватися на різних рівнях, при цьому кожен сервер взаємодіє тільки з найближчими рівнями і не пов'язаний запитами з іншими.

6. Надання коду за запитом (Code on Demand). Сервери можуть відправляти код клієнта (наприклад, скрипт для запуску відео). Так загальний код додатку стає складнішим тільки за необхідності.

7. Початок від нуля (Починаючи зі стилю Null). Клієнт знає тільки одну точку входу на сервер. Подальші необхіності взаємодії забезпечуються сервером.

2.4 Стандарти REST API

Сам по собі RESTful не є стандартом або протоколом. Розробники керуються принципами REST API для створення ефективної роботи з серверами для своїх сайтів і програм. Принципи дозволяють створювати серверну архітектуру за допомогою інших протоколів: HTTP, URL, JSON і XML.

Стан ресурсу в певний момент часу називається представленням ресурсу. Воно складається з:

– даних;

– метаданих, описуваних даних;

– гіпермедіа посилань, які допомагають клієнтам перейти в наступний стан.

Ця інформація надається клієнту в будь-якому форматі: JavaScript (JSON), HTML, XLT, Python, PHP або простий текст.

JSON – найбільш популярний формат, тому що він легко читається машинами і людиною і не залежить від мови програмування.

2.5 Висновки з другого розділу

У другому розділі викладено архітектуру взаємодії клієнтської та серверної частини чат-боту. Проведено огляд існуючих архітектур та виявлено, що найпопулярнішою серед них є технологія взаємодії клієнту і серверу REST API.

Розглянуто принципи написання коду серверної частини за архітектурою REST API та наведено схему взаємодії серверу і клієнту.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Формулювання вимог до проєкту

Правильне висування вимог – один з ключових етапів формування проєкту. Поганий або неопрацьований план може зробити процес розробки неефективним, змінити кінцевий результат, віддаливши його від початковою цілі сервісу, зробити проект неможливим для виконання.

У формуванні вимог потрібно орієнтуватися на переваги користувачів. Виходячи з їх переваг і потреб, було виявлено перелік вимог, що поєднуються з висунутими задачами.

Найбільш поширеними потребами користувачів можуть бути:

– швидкий доступ до конфігурацій каналу;

– зручний інтерфейс;

– гнучкі налаштування;

– можливість повного доступу до функціоналу сервісу.

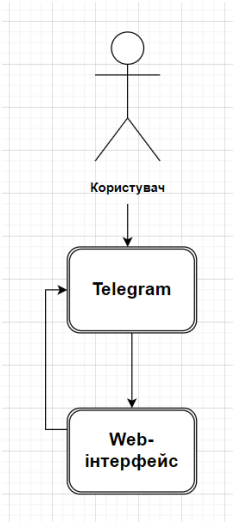

Рисунок 3.1 – Принцип функціонування боту

Дана схема наочно показує принцип функціонування чат-боту та його взаємозв'язок. Під зв'язком мається на увазі передача даних від одного блоку до іншого.

Рисунок 3.1 в загальних рисах відображає основну модель взаємодії та роботи різних елементів програми, але не є достатньо докладною і наочною для розробника. Для визначення конкретних підзадач на основі специфікації вимог потрібно більш докладно розписати кожну взаємодію блоків один з одним.

– користувач – Telegram: користувач запускає бота для налаштування каналу;

– Telegram – web -інтерфейс: для створення опитувань, налаштувань каналу, перегляду не анонімної статистики опитувань, користувач вибирає у роботі Telegram потрібну опцію, і його перенаправляє на web -інтерфейс для подальших дій.

– web-інтерфейс – Telegram: налаштування, виконані у web -інтерфейсі, відтворюються в каналі Telegram

3.2 Вибір середовища розробки

Для реалізації проєкту існує безліч різних рішень, кожне з яких має свої плюси і мінуси. В даній главі будуть розглянуті різні інструменти для реалізації програми, та обранінайбільш відповідні їх.

Python – високорівнева мова програмування загального призначення з динамічною типізацією і автоматичним управлінням пам'яттю. Мова є повністю об'єктно-орієнтованою. Сам ж мова відома як інтерпретована.

Python є мультипарадигмальною мовою програмування, підтримуючою імперативне, процедурне, структурне, об'єктно- орієнтоване та функціональне програмування. Завдання узагальненого програмування вирішуються за рахунок динамічної типізації. Аспектно–орієнтоване програмування частково

підтримується через декоратори, більш повноцінна підтримка забезпечується додатковими фреймворками, вибір яких буде розглянуто нижче.Такі методики як контактне і логічне програмування можна реалізувати з допомогою бібліотек або розширень [6].

Основні архітектурні риси: динамічна типізація, автоматичне управління пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопотокових обчислень з глобальною блокуванням інтерпретатора, високорівневі структури даних. Підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети.

Проаналізуємо переваги та недоліки цієї мови.

Переваги:

– орієнтованість мови на підвищення продуктивності розробника, читання коду і його якості

– забезпечення переносимості написаних на ньому програм;

– мінімалістичний синтаксис;

– стандартна бібліотека включає великий набір корисних функцій;

– велика кількість фреймворків .

Недоліки:

– найчастіше невисока швидкість роботи і високе споживання пам’яті;

– динамічна типізація може викликати помилки при неправильній роботі.

Порівняно з іншими мовами, наприклад C#, Python більш лаконічний при роботі, зокрема, з Telegram -ботами, що, разом з вищепереліченими факторами та некритичними для даної розробки мінусами робить Python найбільш вдалою мовою розробки проєкту.

Для роботи з web -середовищем буде потрібно вибрати відповідний фреймворк. Для порівняння були обрані три найбільш популярних для даних завдань фреймворки: Django і Flask.

Переваги Django :

– принцип "Все включено". Цей принцип говорить про те, що більшість інструментів для створення програми – частина фреймворку, а не поставляються

в вигляді окремих бібліотек [1].

– стандартизована структура. Django, як фреймворк, задає структуру проєкту. Вона допомагає розробникам розуміти, де і як додавати нову функціональність.

– завдяки однаковій для всіх проєктів структурі набагато простіше знайти вже готові рішення або отримати допомогу від інших програмістів.

– програми Django. Програми Django дозволяють розділити проєкт на кілька частин. Цей підхід дозволяє легко інтегрувати готові рішення [4].

– безпечний за замовчуванням. Django безпечний з коробки і включає механізми запобігання поширених таких атак, як SQL – ін'єкцій і підробки міжсайтових запитів.

Недоліки:

– Django розвивається повільно. Django є великим і монолітним фреймворком. Це дозволяє спільноті розробляти сотні універсальних модулів і додатків, але знижує швидкість розробки самого Django . Крім того, фреймворк повинен підтримувати зворотний сумісність, тому він розвивається щодо повільно;

– компоненти розвертаються спільно;

– фреймворк використовує шаблон маршрутизації з вказівкою URL;

– швидкість.

Не можна сказати, що фреймворк сам собою повільний. Однак, якщо неправильно спроєктувати архітектуру, то вона у поєднанні з Python (як сказано вище, не найшвидшою мовою програмування) призведе до повільної роботи сайту або програми.

Тепер розглянемо фреймворк Flask :

Переваги:

– легкий для розуміння;

– Flask має просту структуру і інтуїтивно зрозумілий синтаксису. Крім того, на відміну від інших фреймворків Flask дозволяє програмісту повністю контролювати процес розробки;

– гнучкий;

– лише деякі частини фреймворка недоступні для модифікацій, тому що вони вже максимально прості та оптимізовані. Програміст може самостійно редагувати велику частина інструментів фреймворка під свої потреби;

– разом з Flask програміст отримує шаблонізатор , які дозволяє використовувати один і той ж користувальницький інтерфейс на кількох сторінках [2];

– зручні інструменти для тестування.

В Flask інтегровані інструменти для тестування і налагодження. Програміст має в розпорядженні повноцінні unit тести, вбудований сервер розробки, відладчик та обробник запитів.

Недоліки:

– не підтримує асинхронність;

– Flask обробляє всі запити на один потік, тобто кожен запит блокує потік на час свого виконання, а потім передає черга наступному запит [8];

– нестача можливостей.

Flask – це мікро-веб-фреймворк, а значить, його функціонал обмежений, і не підійде для роботи над великими проєктами.

Зіставивши переваги і недоліки двох вищеперелічених фреймворків можна зробити висновок, що для створення невеликої програми, яким і є проєкт, що розробляється, більше підійде Flask , як більш легкий і гнучкий інструмент розробки, щодо Django .

Web-сервер – це технологія, що використовується для розміщення одного або кілька веб-сайтів. Для роботи з цими ресурсами використовується web - браузер в якості клієнта.

3.4 Проєктування бази даних

Для зберігання даних про розклад занять навчальних груп ЗНУ було розроблено семантичний аналізатор excel-файлу із розкладом навчання академічних груп. Діаграма структури БД представлена на рисунку 3.2.

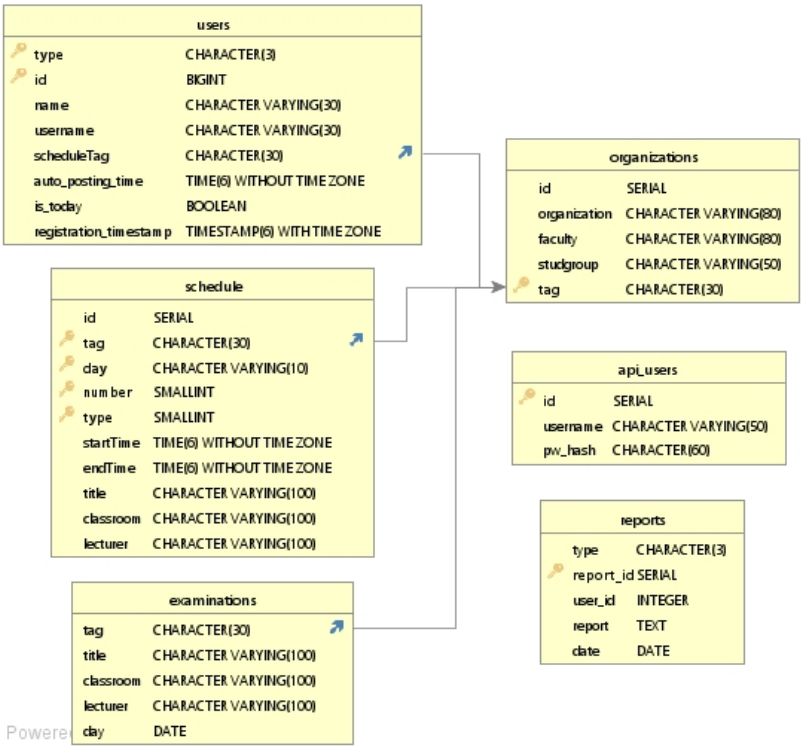

Рисунок 3.2 – Діаграма бази даних боту

Таблиці бази даних та їх властивості:

Users – зберігає інформацію про користувачів системи;

Schedules – зберігає інформацію про розклад;

Examinations – містить дані про іспити;

Organizations – відношення студенти-розклад.

SQL-запити на створення таблиць наведено у додатку А.

3.5 Структура серверної складової чат-боту

Серверна частина чат-боту на платформі Telegram розділена на модулі. Кожен модуль вирішує окремі задачі:

1. Модуль Telegram боту: [UniverScheduleBot](./bot) – бот, який показує розклад занять. Основний модуль проєкту, що здійснює обробку запитів користувача і формування відповідей на них.

2. Модуль для автоматичного відправлення розкладу: [autoposting](./autoposting)

3. Модуль для роботи з базою даних боту (додавання/зміна/видалення груп та файлів розкладу): [api_server].

4. Панель керування базою даних бота: [Bot Rozklad Control Panel]. Надає веб-інтерфейс для взаємодії з розкладом БД, а також дозволяє редагувати користувачів API бота (модуль [api_server].

Структура проєкту відображена у таблиці 3.1, а коди окремих модулів у додатках Б-Е.

Таблиця 3.1 –

Структура проєкту

Назва модулю	Складові модулю	Призначення складової модулю
Модуль для автоматичного відправлення розкладу (autoposting)	auto_posting_thread.py	основний скрипт модулю
	scheduleCreator.py	функції для генерації повідомлень з розкладом
	scheduledb.py	клас для роботи з базою даних
	helpers.py	допоміжні функції
	config.py	налаштування модулю

Продовження таблиці 3.1

Модуль боту для Telegram (bot)	UniverScheduleBot.py	основний скрипт модуля
	scheduleCreator.py	функції для генерації повідомлень з розкладом
	scheduledb.py	клас для роботи із базою даних
	helpers.py	допоміжні функції
	config.py	налаштування модулю

Вебінтерфейс серверної частини боту наведено у додатку Є.

3.6 Висновки з розділу 3

У третьому розділі кваліфікаційної роботи спроєктовано структуру бази даних серверної частини чат-боту та розроблено програмний код обробки Excel-файлу із розкладом студентів у навчальному закладі. Запрограмовано алгоритм обробки розкладу та алгоритм взаємодії із клієнтом.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено огляд існуючих чат-ботів, на основі якого були виділені ключові особливості для створюваного сервісу, що дозволяють отримати продукт, виділяється серед решти унікальним функціоналом.

Розроблено REST API чат – боту, який має функціонал спілкування із студентами вишу з приводу поточного розкладу занять навчального процесу в форматі Telegram каналів.

Для досягнення мети були розглянуті різні інструменти і обрані найбільш підходящі для реалізації розробки. Виконаний проєкт було реалізовано засобами мови програмування Python та фреймворку Flask.

Таким чином, розроблено повноцінний вебсервіс для використання його у якості чат-боту у Telegram-каналах при інформуванні студентів розкладу навчального процесу у будь-якому навчальному закладі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Вікіпедія. Telegram. URL: https://uk.wikipedia.org/wiki/Telegram. (дата звернення: 05.10.22)

1. Вікіпедія. Бот. URL: https://uk.wikipedia.org/wiki/ Робот_(програма)

2. Relan Kunal. Building REST APIs with Flask: Create Python Web Services with MySQL. Apress, 2019 — 132 p.

3. Peralta J.H. Microservice APIs: Using Python, Flask, FastAPI, OpenAPI and more. Manning, 2023. — 411 p.

4. Asif M. Python for Geeks: Build production-ready applications using advanced Python concepts and industry best practices.Packt Publishing, 2021. — 546 p.

5. Grinberg M. Flask Web Development, 2018 . 314 p.

6. Apache . URL:https://httpd.apache.org/.

7. Django . URL:https://www.djangoproject.com/

8. Python.URL:https://www.python.org/ .

9. Toledo L. Python-telegram-bot Documentation.URL: https://readthedocs.org/projects/python-telegram-bot / downloads / pdf / latest /.

10. Flask . URL:https://flask.palletsprojects.com/en/2.0.x/.

ДОДАТОК А

SQL-запити на створення таблиць бази даних

```
CREATE TABLE IF NOT EXISTS organizations
(
  id serial,
  organization character varying(80),
  faculty character varying(80),
  studgroup character varying(50),
  tag character(30) PRIMARY KEY
);

CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE INDEX IF NOT EXISTS trgm_idx ON organizations USING GIN (lower(organization || ' '
|| faculty || ' ' || studgroup) gin_trgm_ops);

CREATE TABLE IF NOT EXISTS schedule
(
  id serial,
  tag character(30),
  day character varying(10),
  "number" smallint,
  type smallint,
  "startTime" time without time zone,
  "endTime" time without time zone,
  title character varying(100),
  classroom character varying(100),
  lecturer character varying(100),
  PRIMARY KEY (tag, day, number, type),
  CONSTRAINT schedule_fkey FOREIGN KEY (tag)
  REFERENCES organizations (tag) MATCH SIMPLE
  ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS examinations
(
  tag character(30),
  title character varying(100),
  classroom character varying(100),
  lecturer character varying(100),
  day date,
  CONSTRAINT examinations_fkey FOREIGN KEY (tag)
  REFERENCES organizations (tag) MATCH SIMPLE
  ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS users
(
```

```
  type character(3),
  id bigint,
  name character varying(30),
  username character varying(30),
  "scheduleTag" character(30),
  auto_posting_time time without time zone,
  is_today boolean,
  registration_timestamp timestamp with time zone DEFAULT now(),
  PRIMARY KEY (type, id),
  CONSTRAINT users_fkey FOREIGN KEY ("scheduleTag")
      REFERENCES organizations (tag) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE SET NULL
);

CREATE OR REPLACE VIEW users_vw AS
 SELECT users.type,
    users.id,
    organizations.organization,
    organizations.faculty,
    organizations.studgroup,
    users.auto_posting_time,
    users.is_today,
    users.registration_timestamp
   FROM users
     JOIN organizations ON users."scheduleTag" = organizations.tag
 ORDER BY organizations.studgroup;

CREATE TABLE IF NOT EXISTS reports
(
  type character(3),
  report_id serial PRIMARY KEY,
  user_id integer,
  report text,
  date date
);

CREATE TABLE IF NOT EXISTS api_users
(
  id serial PRIMARY KEY,
  username character varying(50),
  pw_hash character(60)
)
```

ДОДАТОК Б

Головний скрипт модулю автоматичної відправки розкладу занять

```
import logging
import threading
from datetime import datetime, timedelta
from time import sleep

import telebot

from config import config
from helpers import daysOfWeek, get_date_keyboard, get_week_type
from scheduleCreator import create_schedule_text
from scheduledb import ScheduleDB
from statistic import track

bot = telebot.AsyncTeleBot(config["TOKEN"])

logging.basicConfig(format='%(asctime)-15s [ %(levelname)s ] uid=%(userid)s %(message)s',
                    filemode='a',
                    filename=config["LOG_DIR_PATH"] + "log-
{0}.log".format(datetime.now().strftime("%Y-%m-%d")),
                    level="INFO")
logger = logging.getLogger('bot-logger')

def auto_posting(current_time, day, week_type, is_today=True):
    # Выборка пользователей из базы
    with ScheduleDB(config) as db:
        users = db.find_users_where(auto_posting_time=current_time, is_today=is_today)

    if users is None:
        return None
    try:
        for user in users:
            cid = user[0]
            tag = user[1]

            schedule = create_schedule_text(tag, day[0], week_type)
            if len(schedule[0]) <= 14:
                continue
            bot.send_message(cid, schedule, reply_markup=get_date_keyboard())

            # Статистика
            if config['STATISTIC_TOKEN'] != '':
                track(config['STATISTIC_TOKEN'], cid, current_time, 'auto_posting')
            else:
                logger.info('auto_posting. Time: {0}'.format(current_time), extra={'userid': cid})
    except BaseException as e:
```

```
        logger.warning('auto_posting: {0}'.format(str(e)), extra={'userid': 0})

def today_schedule(current_time):
    today = datetime.now()
    week_type = get_week_type(today)

    if datetime.weekday(today) == 6:
        today += timedelta(days=1)
        week_type = (week_type + 1) % 2

    day = [daysOfWeek[datetime.weekday(today)]]

    auto_posting(current_time, day, week_type)

def tomorrow_schedule(current_time):
    tomorrow = datetime.now()
    tomorrow += timedelta(days=1)
    week_type = get_week_type(tomorrow)

    # Вибірка користувачів з бази, у яких встановлено відправлення розпису на завтрашній день,
     Якщо сьогодні неділя, то розклад буде відправлятися на понеділок.
    if datetime.weekday(tomorrow) == 6:
        tomorrow += timedelta(days=1)
        week_type = (week_type + 1) % 2

    day = [daysOfWeek[datetime.weekday(tomorrow)]]

    auto_posting(current_time, day, week_type, is_today=False)

if __name__ == "__main__":
    while True:
        # Отправка расписания на сегодня
        threading.Thread(target=today_schedule(datetime.now().time().strftime("%H:%M:00"))).start()

        # Отправка расписания на завтра
        threading.Thread(target=tomorrow_schedule(datetime.now().time().strftime("%H:%M:00"))).sta
rt()

        # Обчислюємо різницю в секундах, між початком хвилини та часом завершення потоку
        time_delta = datetime.now() - datetime.now().replace(second=0, microsecond=0)
        # Потік засинає на час, що дорівнює кількості секунд до наступної хвилини.
        sleep(60 - time_delta.seconds)
```

ДОДАТОК В

Функції для створення повідомлення з розкладом

```
# -*- coding: utf-8 -*-
from functools import lru_cache

import scheduledb
from config import config
from helpers import daysOfWeek_rus

def print_type(raw_type, week_type=-1):
    # Якщо задано тип тижня (числитель/знаменник), тобто week_type не дорівнює значенням за замовчуванням,
     # то додаткова інформація про тип тижня не виводиться
    if week_type != -1:
        return ""

    raw_type = int(raw_type)
    if raw_type == 0:
        return "числ"
    elif raw_type == 1:
        return "знам"
    elif raw_type == 2:
        return ""

@lru_cache(maxsize=128)
def create_schedule_text(tag, day, week_type=-1):
    result = []
    schedule = ""
    try:
        with scheduledb.ScheduleDB(config) as db:
            data = db.get_schedule(tag, day, week_type)

        schedule += ">{0}:\n".format(daysOfWeek_rus[day])
        index = 0
        while index < len(data):
            row = data[index]

            title = ' '.join(str(row[1]).split())
            classroom = ' '.join(str(row[2]).split())

            schedule += str(row[0]) + " пара:\n"
            # Цей блок необхідний висновку тих занять, де заняття по чисельнику і знаменнику різняться
            if index != len(data) - 1:
                # Порівнюється порядковий номер заняття даного та наступного рядка і якщо вони рівні,
```

```
# то вони виводяться разом
            if data[index + 1][0] == row[0]:
                schedule += '{0} {1} {2}\n'.format(title, classroom, print_type(row[3], week_type))

                index += 1
                row = data[index]
                title = ' '.join(str(row[1]).split())
                classroom = ' '.join(str(row[2]).split())

                schedule += '{0} {1} {2}\n'.format(title, classroom, print_type(row[3], week_type))
            else:
                schedule += '{0} {1} {2}\n'.format(title, classroom, print_type(row[3], week_type))
        else:
            schedule += '{0} {1} {2}\n'.format(title, classroom, print_type(row[3], week_type))

        schedule += "----------\n"
        index += 1
    result.append(schedule)
  except:
    pass
  finally:
    return result

def create_schedule_xls(tag, day):
  pass

def create_schedule_pdf(tag, day):
  pass
```

ДОДАТОК Г

Клас для роботи з базою даних

```
import hashlib
import logging
import psycopg2
from datetime import datetime

organization_field_length = 15
faculty_field_length = 10
group_field_length = 5

class ScheduleDB:
    def __init__(self, config):
        self.con = psycopg2.connect(
            dbname=config["DB_NAME"],
            user=config["DB_USER"],
            password=config["DB_PASSWORD"],
            host=config["DB_HOST"])
        self.cur = self.con.cursor()

        logging.basicConfig(format='%(asctime)-15s [ %(levelname)s ] %(message)s',
                            filemode='a',
                            filename=config["LOG_DIR_PATH"] + "log-{0}.log".format(datetime.now().strftime("%Y-
%m")))
        self.logger = logging.getLogger('db-logger')

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        self.con.commit()
        self.con.close()

    @staticmethod
    def create_tag(organization, faculty, group):
        org_hash = hashlib.sha256(organization.encode('utf-8')).hexdigest()
        faculty_hash = hashlib.sha256(faculty.encode('utf-8')).hexdigest()
        group_hash = hashlib.sha256(group.encode('utf-8')).hexdigest()
        return org_hash[:organization_field_length] + \
               faculty_hash[:faculty_field_length] + \
               group_hash[:group_field_length]

    def add_lesson(self, tag, day, number, week_type, time_start, time_end, title, classroom, lecturer):
        try:
            self.cur.execute('INSERT INTO schedule(tag, day, "number", type, "startTime", "endTime", \
                              title, classroom, lecturer) VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s);',
                              (tag, day, number, week_type, time_start, time_end, title, classroom, lecturer))
            self.con.commit()
            return True
        except BaseException as e:
            self.logger.warning('Add to schedule failed. Error: {0}. Data:\
```

```
                    tag={1},\
                    day={2},\
                    number={3},\
                    week_type={4},\
                    time_start={5},\
                    time_end={6},\
                    title={7},\
                    classroom={8},\
                    lecturer={9}'.format(
                str(e), tag, day, number, week_type, time_start, time_end, title, classroom, lecturer))
            return False

    def add_exam(self, tag, title, classroom, lecturer, day):
        try:
            self.cur.execute("INSERT INTO examinations(tag, title, classroom, lecturer, day)
VALUES(%s,%s,%s,%s,%s);",
                             (tag, title, classroom, lecturer, day))
            self.con.commit()
            return True
        except BaseException as e:
            self.logger.warning("Add exam failed. Error: {0}. Data:\
                    tag={1},\
                    title={2},\
                    classroom={3},\
                    lecturer={4},\
                    day={5}".format(str(e), tag, title, classroom, lecturer, day))
            return False

    def add_organization(self, organization, faculty, group):
        tag = self.create_tag(organization, faculty, group)
        try:
            self.cur.execute("INSERT INTO organizations(organization, faculty, studgroup, tag)
VALUES(%s,%s,%s,%s);",
                             (organization, faculty, group, tag))
            self.con.commit()
            return tag
        except BaseException as e:
            self.logger.warning("Add organization failed. Error: {0}. Data:\
                    organization={1},\
                    faculty={2},\
                    group={3},\
                    tag={4}".format(str(e), organization, faculty, group, tag))
            return None

    def add_report(self, cid, report):
        try:
            self.cur.execute('INSERT INTO reports (type, user_id, report, date) VALUES(%s, %s, %s, %s)',
                             ('tg', cid, report, datetime.now().strftime("%Y-%m-%d %H:%M:%S")))
            self.con.commit()
            return True
        except BaseException as e:
            self.logger.warning('Add report failed. Error: {0}. Data: cid={1}, report={2}'.format(str(e), cid, report))
            return False

    def add_user(self, cid, name, username, tag):
```

```
        try:
            self.cur.execute('INSERT INTO users VALUES(%s,%s,%s,%s,%s,null,null)', ('tg', cid, name,
username, tag))
            self.con.commit()
            return True
        except BaseException as e:
            self.logger.warning('Add user failed. Error: {0}. Data: cid={1}, name={2}, username={3},
tag={4}'.format(
                str(e), cid, name, username, tag))
            raise e

    def update_user(self, cid, name, username, tag):
        try:
            self.cur.execute('UPDATE users SET "scheduleTag" = (%s) WHERE id = (%s) AND type = (%s)',
(tag, cid, 'tg'))
            self.con.commit()
            return True
        except BaseException as e:
            self.logger.warning('Update user failed. Error: {0}. Data: cid={1}, name={2}, username={3},
tag={4}'.format(
                str(e), cid, name, username, tag))
            raise e

    def find_user(self, cid):
        try:
            self.cur.execute('SELECT "scheduleTag" FROM users WHERE id = (%s) AND type = (%s)', (cid, 'tg'))
            return self.cur.fetchone()
        except BaseException as e:
            self.logger.warning('Select user failed. Error: {0}. Data: cid={1}'.format(str(e), cid))
            raise e

    def find_users_where(self, auto_posting_time=None, is_today=None):
        try:
            if auto_posting_time is not None and is_today is not None:
                self.cur.execute('SELECT id, "scheduleTag" FROM users \
                                  WHERE auto_posting_time = %s AND is_today = %s  AND type = (%s)',
                                 (auto_posting_time, is_today, 'tg'))
                return self.cur.fetchall()
            elif auto_posting_time is not None:
                self.cur.execute('SELECT id, "scheduleTag" FROM users \
                                  WHERE auto_posting_time = %s AND type = (%s)',
                                 (auto_posting_time, 'tg'))
                return self.cur.fetchall()
            elif is_today is not None:
                self.cur.execute('SELECT id, "scheduleTag" FROM users WHERE is_today = %s AND type =
(%s)',
                                 (is_today, 'tg'))
                return self.cur.fetchall()
            else:
                self.cur.execute('SELECT id, "scheduleTag" FROM users WHERE type = (%s)', ['tg'])
                return self.cur.fetchall()
        except BaseException as e:
            self.logger.warning('Select users failed. Error: {0}. auto_posting_time={1}'.format(
                str(e), auto_posting_time))
            raise e
```

```
    def get_exams(self, tag):
        exams = []
        try:
            self.cur.execute("SELECT day, title, classroom, lecturer FROM examinations \
                        WHERE tag = (%s) ORDER BY day", [str(tag)])
            exams = self.cur.fetchall()
        except BaseException as e:
            self.logger.warning('Select exams failed. Error: {0}. Data: tag={1}'.format(str(e), tag))
            raise e
        finally:
            return exams

    def get_schedule(self, tag, day, week_type=-1):
        data = []
        try:
            if week_type != -1:
                self.cur.execute('SELECT number,title,classroom,type FROM schedule \
                        WHERE tag = (%s) AND day = (%s) AND (type = 2 OR type = %s) \
                        ORDER BY number, type ASC', (tag, day, week_type))
            else:
                self.cur.execute('SELECT number,title,classroom,type FROM schedule \
                        WHERE tag = (%s) AND day = (%s) ORDER BY number, type ASC', (tag, day))
            data = self.cur.fetchall()
        except BaseException as e:
            self.logger.warning('Select schedule failed. Error: {0}. Data: tag={1}, day={2},
week_type={3}'.format(
                str(e), tag, day, week_type))
            raise Exception
        finally:
            return data

    def get_organizations(self, tag=""):
        organizations = []
        try:
            self.cur.execute("SELECT DISTINCT ON (organization) organization, tag \
            FROM organizations WHERE tag LIKE %s ORDER BY organization;", [tag + '%'])
            organizations = self.cur.fetchall()
        except BaseException as e:
            self.logger.warning('Select schedule failed. Error: {0}. Data: tag={1}'.format(str(e), tag))
            raise e
        finally:
            return organizations

    def get_faculty(self, tag=""):
        faculties = []
        try:
            self.cur.execute("SELECT DISTINCT ON (faculty) faculty, tag \
            FROM organizations WHERE tag LIKE %s ORDER BY faculty;", [tag + '%'])
            faculties = self.cur.fetchall()
        except BaseException as e:
            self.logger.warning('Select schedule failed. Error: {0}. Data: tag={1}'.format(str(e), tag))
            raise e
        finally:
            return faculties
```

```
    def get_group(self, tag=""):
        group = []
        try:
            self.cur.execute("SELECT DISTINCT ON (studGroup) studGroup, tag \
            FROM organizations WHERE tag LIKE %s ORDER BY studGroup;",
                             [tag + '%'])
            group = self.cur.fetchall()
        except BaseException as e:
            self.logger.warning('Select group failed. Error: {0}. Data: tag={1}'.format(str(e), [tag]))
            raise e
        finally:
            return group

    def set_auto_post_time(self, cid, time, is_today):
        try:
            self.cur.execute('UPDATE users SET auto_posting_time = %s, is_today = %s \
            WHERE id = %s AND type = (%s)',
                             (time, is_today, cid, 'tg'))
            self.con.commit()
            return True
        except BaseException as e:
            self.logger.warning('Set auto post time failed. Error: {0}. Data: cid={1},
auto_posting_time={2}'.format(
                str(e), cid, time))
            raise e

    def clear_tables(self):
        try:
            self.cur.execute('TRUNCATE users;')
            self.cur.execute('TRUNCATE organizations CASCADE;')
            self.cur.execute('TRUNCATE reports;')
            self.con.commit()

            old_isolation_level = self.con.isolation_level
            self.con.set_isolation_level(0)

            self.cur.execute('VACUUM')
            self.con.commit()

            self.con.set_isolation_level(old_isolation_level)

            return True
        except BaseException as e:
            self.logger.warning('clear tables failed. Error: {0}.'.format(
                str(e)))
            raise e
```

ДОДАТОК Ґ

Допоміжні функції

```
rom telebot import types
from datetime import datetime
from config import config

daysOfWeek = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

ScheduleType = {
   "Понеділок": daysOfWeek[0],
   "Вівторок": daysOfWeek[1],
   "Середа": daysOfWeek[2],
   "Четвер": daysOfWeek[3],
   "П’ятниця": daysOfWeek[4],
   "Субота": daysOfWeek[5],
   "Неділя": daysOfWeek[6],
   "Сьогодні": "Today",
   "Завтра": "Tomorrow",
   "Весь тиждень": daysOfWeek
}

daysOfWeek_rus = {
   daysOfWeek[0]: " Понеділок ",
   daysOfWeek[1]: " Вівторок ",
   daysOfWeek[2]: " Середа ",
   daysOfWeek[3]: "Четвер",
   daysOfWeek[4]: " П’ятниця ",
   daysOfWeek[5]: "Субота",
   daysOfWeek[6]: " Неділя ",
}
def get_date_keyboard():
   now = datetime.now()
   date_select = types.ReplyKeyboardMarkup(row_width=2, resize_keyboard=True,
one_time_keyboard=False)

   # Якщо зараз грудень, січень чи травень, червень, то виводиться кнопка іспити
   if now.month == 1 or now.month == 12 or now.month == 5 or now.month == 6:
      date_select.row('Экзамены')

   date_select.row("Сьогодні")
   date_select.row("Завтра")
   date_select.row("Весь тиждень")
   date_select.row("Понеділок", "Вівторок")
   date_select.row("Середа", "Четвер")
   date_select.row("П’ятница", "Субота")

   return date_select

def get_week_type(day):
   return (day.isocalendar()[1] + int(config["WEEK_TYPE"])) % 2
```

ДОДАТОК Д

Основний скрипт модулю боту для Telegram

```
# -*- coding: utf-8 -*-
import logging
import re
from datetime import datetime, time, timedelta

import flask
from flask import request, jsonify

import telebot
from telebot import types

from config import config
from helpers import daysOfWeek, ScheduleType, get_date_keyboard, get_week_type
from scheduleCreator import create_schedule_text
from scheduledb import ScheduleDB, organization_field_length, faculty_field_length

# Статистика
from statistic import track

WEBHOOK_URL_BASE = "https://{}:{}".format(config["WEBHOOK_HOST"],
config["WEBHOOK_PORT"])
WEBHOOK_URL_PATH = "/{}/".format(config["TOKEN"])

bot = telebot.AsyncTeleBot(config["TOKEN"])
app = flask.Flask(__name__)

logger = telebot.logger
telebot.logger.setLevel(logging.INFO)

commands = {  # Опис команд, що використовується в команді "help"
    'start': 'Стартове повідомлення та пропозиція зареєструватися',
     'help': 'Інформація про роботу та список доступних команд',
     'registration': 'Вибір ВНЗ, факультету та групи для виведення розкладу',
     'send_report <повідомлення>': 'Надіслати інформацію про помилку або щось ще',
     'auto_posting_on <ЧЧ:ММ>': 'Включення та вибір часу для автоматичного відправлення розкладу в
діалог',
     'auto_posting_off': 'Вимкнення автоматичного відправлення розкладу'
    }

# ----------------------------------
#  BOT HANDLERS
# ----------------------------------

# handle the "/registration" command
@bot.message_handler(commands=['registration'])
def command_registration(m):
    cid = m.chat.id

    # Процедура регистрации проходит в четрые этапа:
```

```
    # 1 етап: вибір навчального закладу <--
    # 2 етап: вибір факультету
    # 3 етап: вибір групи
    # 4 етап: додавання даних про належність користувача до навчального закладу у БД
    try:
        # Статистика
        if config['STATISTIC_TOKEN'] != '':
            track(config['STATISTIC_TOKEN'], cid, 'stage 1', 'registration-stage-1')

        keyboard = types.InlineKeyboardMarkup()

        with ScheduleDB(config) as db:
            result = db.get_organizations()
        for row in result:
            callback_button = types.InlineKeyboardButton(
                text=str(row[0]),
                callback_data="reg:stage 2:{0}".format(str(row[1])[:organization_field_length]))
            keyboard.add(callback_button)

        bot.send_message(cid, "Виберіть університет:", reply_markup=keyboard)
    except BaseException as e:
        logger.warning('Registration problem: {0}'.format(str(e)))
        bot.send_message(cid, "Сталося щось дивне, спробуйте почати спочатку, ввівши команду
/registration")

# handle the "/start" command
@bot.message_handler(commands=['start'])
def command_start(m):
    # Статистика
    # if config['STATISTIC_TOKEN'] != '':
    #     track(config['STATISTIC_TOKEN'], m.chat.id, m.text, 'start')
    # else:
    #     logger.info('start')

    cid = m.chat.id
    command_help(m)

    try:
        with ScheduleDB(config) as db:
            user = db.find_user(cid)
        if user and user[0] is not None:
            bot.send_message(cid, "Ви вже додані до бази даних", reply_markup=get_date_keyboard())
        else:
            bot.send_message(cid, "Вас ще немає у базі даних, тому пройдіть просту процедуру реєстрації")
            command_registration(m)
    except BaseException as e:
        logger.warning('command start: {0}'.format(str(e)))
        bot.send_message(cid, "Сталося щось дивне, спробуйте ввести команду заново",
                         reply_markup=get_date_keyboard())

# help page
@bot.message_handler(commands=['help'])
def command_help(m):
    # Статистика
    # if config['STATISTIC_TOKEN'] != '':
```

```
    #     track(config['STATISTIC_TOKEN'], m.chat.id, m.text, 'help')
    # else:
    #     logger.info('help')

    cid = m.chat.id
    help_text = "Доступні такі команди: \n"
    for key in commands:
        help_text += "/" + key + ": "
        help_text += commands[key] + "\n"
    bot.send_message(cid, help_text, reply_markup=get_date_keyboard())

    help_text = ('Опис кнопок:\nКнопка "Сьогодні", як це не дивно виводить розклад на сьогоднішній
день, '
                 'причому з урахуванням типу тижня ( чисельник / знаменник ), але є один нюанс: якщо
сьогодні неділя '
                 'або час більше 21:30, то виводиться розклад наступного дня\n')
    bot.send_message(cid, help_text, reply_markup=get_date_keyboard())

    guide_url = 'https://github.com/ajax3101/UniverScheduleBot/wiki/Guide'

    help_text = 'Більш детальну інструкцію та опис команд  \
    ви можете подивитися за посиланням: {}'.format(guide_url)
    bot.send_message(cid, help_text, reply_markup=get_date_keyboard())

# send_report handler
@bot.message_handler(commands=['send_report'])
def command_send_report(m):
    # # Статистика
    # if config['STATISTIC_TOKEN'] != '':
    #     track(config['STATISTIC_TOKEN'], m.chat.id, m.text, 'report')
    # else:
    #     logger.info('report')

    cid = m.chat.id
    data = m.text.split("/send_report")

    if data[1] != '':
        report = data[1]
        with ScheduleDB(config) as db:
            if db.add_report(cid, report):
                bot.send_message(cid, "Повідомлення прийнято")
            else:
                bot.send_message(cid, "Сталося щось дивне, спробуйте ввести команду заново",
                                 reply_markup=get_date_keyboard())
    else:
        bot.send_message(
            cid,
            "Сталося щось дивне, спробуйте ввести команду заново Ви відправили порожній рядок.
Приклад: /send_report <повідомлення>",
            reply_markup=get_date_keyboard())

# handle the "/auto_posting_on" command
@bot.message_handler(commands=['auto_posting_on'])
def command_auto_posting_on(m):
    # Статистика
```

```
    # if config['STATISTIC_TOKEN'] != '':
    #     track(config['STATISTIC_TOKEN'], m.chat.id, m.text, 'auto_posting_on')
    # else:
    #     logger.info('auto_posting_on')

    cid = m.chat.id

    try:
        data = m.text.split("/auto_posting_on")[1].strip()
        if re.match(data, r'\d{1,2}:\d\d'):
            raise BaseException
    except:
        bot.send_message(cid, "Ви надіслали порожній рядок або рядок неправильного формату.
Правильний формат ЧЧ:ММ",
                         reply_markup=get_date_keyboard())
        return None

    try:
        db = ScheduleDB(config)
        user = db.find_user(cid)
        if user and user[0] is not None:
            keyboard = types.InlineKeyboardMarkup()
            callback_button = types.InlineKeyboardButton(
                text="На Сегодня",
                callback_data="ap:{0}:1".format(data))
            keyboard.add(callback_button)
            callback_button = types.InlineKeyboardButton(
                text="На Завтра",
                callback_data="ap:{0}:0".format(data))
            keyboard.add(callback_button)

            bot.send_message(cid, "Виберіть день, на який приходитиме розклад:", reply_markup=keyboard)
        else:
            bot.send_message(cid, "Вас ще немає у базі даних, тому пройдіть просту процедуру реєстрації")
            command_registration(m)
    except BaseException as e:
        logger.warning('command auto_posting_on: {0}'.format(str(e)))
        bot.send_message(cid, “Сталося щось дивне, спробуйте ввести команду заново”)

@bot.message_handler(commands=['auto_posting_off'])
def command_auto_posting_off(m):
    # # Статистика
    # if config['STATISTIC_TOKEN'] != '':
    #     track(config['STATISTIC_TOKEN'], m.chat.id, m.text, 'auto_posting_off')
    # else:
    #     logger.info('auto_posting_off')

    cid = m.chat.id

    try:
        db = ScheduleDB(config)
        user = db.find_user(cid)
        if not user:
            bot.send_message(cid, " Вас ще немає у базі даних, тому пройдіть просту процедуру реєстрації")
            command_registration(m)
```

```
            return

        if db.set_auto_post_time(cid, None, None):
            bot.send_message(cid, " Автоматичне відправлення розкладу успішно вимкнено ")
        else:
            bot.send_message(cid, " Сталося щось дивне, спробуйте ввести команду заново ",
                             reply_markup=get_date_keyboard())

    except BaseException as e:
        logger.warning('command auto_posting_off: {0}'.format(str(e)))
        bot.send_message(cid, " Сталося щось дивне, спробуйте ввести команду заново ")

# exams message handler
@bot.message_handler(func=lambda message: 'Екзамени' in (message.text if message.text is not None else "),
content_types=['text'])
def exams(m):
    cid = m.chat.id

    # Статистика
    track(config['STATISTIC_TOKEN'], cid, m.text, 'exams')

    # Якщо користувача немає в базі, то виведе пропозицію зареєструватися
    try:
        with ScheduleDB(config) as db:
            user = db.find_user(cid)
        if not user or user[0] is None:
            message = " Вас ще немає у базі даних, тому пройдіть просту процедуру реєстрації:\n"
            message += 'Введіть команду(без лапок):\n\nреєстрація "назва вузу " "факультет" "группа"\n\n'
            message += 'Якщо ви припуститесь помилки, то просто наберіть команду заново.\n'

            bot.send_message(cid, message, reply_markup=get_date_keyboard())
    except BaseException as e:
        bot.send_message(cid, Сталося щось дивне, спробуйте ввести команду заново ',
                         reply_markup=get_date_keyboard())

    try:
        with ScheduleDB(config) as db:
            exams_list = db.get_exams(user[0])

        message = "
        for exam in exams_list:
            message += exam[0].strftime('%d.%m.%Y') + ":\n"

            title = ' '.join(str(exam[1]).split())
            lecturer = ' '.join(str(exam[2]).split())
            classroom = ' '.join(str(exam[3]).split())

            message += title + ' | ' + lecturer + ' | ' + classroom + "\n"
            message += "----------\n"
        if len(message) == 0:
            message = 'Схоже розкладу іспитів для вашої групи немає в базі '

    except BaseException as e:
        message = " Сталося щось дивне, спробуйте ввести команду заново "
```

```
    bot.send_message(cid, message, reply_markup=get_date_keyboard())

# text message handler
@bot.message_handler(func=lambda message: True, content_types=['text'])
def response_msg(m):
    cid = m.chat.id
    if m.text in ScheduleType:
        # Статистика
        if config['STATISTIC_TOKEN'] != '':
            track(config['STATISTIC_TOKEN'], m.chat.id, m.text, 'schedule')
        else:
            logger.info('message: {0}'.format(m.text))

        # За умовчанням week_type дорівнює -1 і при такому значенні будуть виводиться всі заняття,
         # і для парних і непарних тижнів
        week_type = -1

        if m.text == "Весь тиждень":
            days = ScheduleType[m.text]
        elif m.text == "Сегодня":
            today = datetime.now()
            # Якщо запитується розклад на сьогоднішній день,
             # то week_type дорівнює залишку від розподілу на 2 номери тижня в році, тобто він визначає
парна вона або непарна
            week_type = get_week_type(today)

            # Якщо сьогодні неділя, то показується розклад на понеділок наступного тижня
             # Також у цьому випадку, як week_type використовується тип наступного тижня
            if datetime.weekday(today) == 6:
                today += timedelta(days=1)
                week_type = (week_type + 1) % 2

            days = [daysOfWeek[datetime.weekday(today)]]
        elif m.text == 'Завтра':
            tomorrow = datetime.now()
            tomorrow += timedelta(days=1)
            # Якщо запитується розклад на сьогоднішній день,
             # то week_type дорівнює залишку від розподілу на 2 номери тижня в році, тобто він визначає
парна вона або непарна
            week_type = get_week_type(tomorrow)

            # Якщо сьогодні неділя, то показується розклад на понеділок наступного тижня
             # Також у цьому випадку, як week_type використовується тип наступного тижня
            if datetime.weekday(tomorrow) == 6:
                tomorrow += timedelta(days=1)
                week_type = (week_type + 1) % 2

            days = [daysOfWeek[datetime.weekday(tomorrow)]]
        else:
            days = [ScheduleType[m.text]]

        for day in days:
            try:
                with ScheduleDB(config) as db:
                    user = db.find_user(cid)
```

```
            if user and user[0] is not None:
                result = create_schedule_text(user[0], day, week_type)
                for schedule in result:
                    bot.send_message(cid, schedule, reply_markup=get_date_keyboard())
            else:
                bot.send_message(cid, " Вас ще немає у базі даних, тому пройдіть просту процедуру
реєстрації ")
                command_registration(m)
        except BaseException as e:
            logger.warning('response_msg: {0}'.format(str(e)))
            bot.send_message(cid, “Сталося щось дивне, спробуйте ввести команду заново”)
    else:
        # Статистика
        if config['STATISTIC_TOKEN'] != '':
            track(config['STATISTIC_TOKEN'], m.chat.id, m.text, 'unknown')
        else:
            logger.info('unknown message: {0}'.format(m.text))

        bot.send_message(cid, "Невідома  команда", reply_markup=get_date_keyboard())

# ------------------------------------
#  BOT CALLBACKS
# ------------------------------------

@bot.callback_query_handler(func=lambda call: "reg:stage 2:" in call.data)
def callback_registration(call):
    cid = call.message.chat.id

    # Парсинг повідомлення, що вказує стадію реєстрації
    # reg : stage : tag
    callback_data = re.split(r':', call.data)

    # Процедура реєстрації проходить у чотири етапи:
     # 1 етап: вибір навчального закладу
     # 2 етап: вибір факультету <--
     # 3 етап: вибір групи
     # 4 етап: додавання даних про належність користувача до навчального закладу до БД
    try:
        # Статистика
        if config['STATISTIC_TOKEN'] != '':
            track(config['STATISTIC_TOKEN'], cid, 'stage 2', 'registration-stage-2')

        keyboard = types.InlineKeyboardMarkup()

        organization_id = callback_data[2]

        with ScheduleDB(config) as db:
            result = db.get_faculty(organization_id)

        for row in result:
            callback_button = types.InlineKeyboardButton(
                text=str(row[0]),
                callback_data="reg:stage 3:{0}".format(
                    str(row[1])[:organization_field_length + faculty_field_length]))
            keyboard.add(callback_button)
```

```
        bot.send_message(cid, "Оберіть факультет:", reply_markup=keyboard)
    except BaseException as e:
        logger.warning('Registration problem: {0}'.format(str(e)))
        bot.send_message(cid, " Сталося щось дивне, спробуйте почати спочатку, ввівши команду
/registration")

@bot.callback_query_handler(func=lambda call: "reg:stage 3:" in call.data)
def callback_registration(call):
    cid = call.message.chat.id

    # Парсинг повідомлення, що вказує стадію реєстрації
    # reg : stage : tag
    callback_data = re.split(r':', call.data)

    # Процедура реєстрації проходить у чотири етапи:
     # 1 етап: вибір навчального закладу
     # 2 етап: вибір факультету
     # 3 етап: вибір групи <--
     # 4 етап: додавання даних про належність користувача до навчального закладу до БД
    try:
        # Статистика
        if config['STATISTIC_TOKEN'] != '':
            track(config['STATISTIC_TOKEN'], cid, 'stage 3', 'registration-stage-3')

        keyboard = types.InlineKeyboardMarkup()

        faculty_id = callback_data[2]

        with ScheduleDB(config) as db:
            result = db.get_group(faculty_id)

        for row in result:
            callback_button = types.InlineKeyboardButton(
                text=str(row[0]),
                callback_data="reg:stage 4:{0}".format(str(row[1])))
            keyboard.add(callback_button)

        bot.send_message(cid, "Выберите группу:", reply_markup=keyboard)
    except BaseException as e:
        logger.warning('Registration problem: {0}'.format(str(e)))
        bot.send_message(cid, "Сталося щось дивне, спробуйте почати спочатку, ввівши
команду/registration")

@bot.callback_query_handler(func=lambda call: "reg:stage 4:" in call.data)
def callback_registration(call):
    cid = call.message.chat.id

    # Парсинг повідомлення, що вказує стадію реєстрації
    # reg : stage : tag
    callback_data = re.split(r':', call.data)

    # Процедура регистрации проходит в четрые этапа:
    # 1 этап: выбор учебного заведения
    # 2 этап: выбор факультета
```

```
    # 3 этап: выбор группы
    # 4 этап: добавление данных о принадлежности пользователя к учебному заведению в БД <--
    try:
        # Статистика
        if config['STATISTIC_TOKEN'] != '':
            track(config['STATISTIC_TOKEN'], cid, 'stage 4', 'registration-stage-4')

        group_id = callback_data[2]

        db = ScheduleDB(config)

        row = db.get_group(group_id)
        user = db.find_user(cid)

        if user:
            db.update_user(cid, call.message.chat.first_name, call.message.chat.username, str(row[0][1]))
        else:
            db.add_user(cid, call.message.chat.first_name, call.message.chat.username, str(row[0][1]))

        bot.send_message(cid, " Чудово, ви зареєструвалися, ваша група:" + row[0][0] +
                         "\ nЯкщо ви помилилися, то просто введіть команду /registration і змініте дані ",
                         reply_markup=get_date_keyboard())
        bot.send_message(cid, " Тепер ви можете налаштувати автоматичне відправлення розкладу в
заданий час,"
                              " Ввівши команду /auto_posting_on <час>, "
                              "де <час> має мати формат ЧЧ:ММ")

    except BaseException as e:
        logger.warning('Registration problem: {0}'.format(str(e)))
        bot.send_message(cid, "Сталося щось дивне, спробуйте почати спочатку, ввівши
команду/registration")

@bot.callback_query_handler(func=lambda call: "ap:" in call.data)
def callback_auto_posting(call):
    cid = call.message.chat.id

    try:
        callback_data = re.split(r':', call.data)

        # Перевірка на відповідність введених користувачем даних прийнятому формату
        if len(callback_data) != 4:
            bot.send_message(cid,
                             " Ви надіслали порожній рядок або рядок неправильного формату. Правильний формат
ЧЧ:ММ ",
                             reply_markup=get_date_keyboard())
            return

        hour = ''.join(filter(lambda x: x.isdigit(), callback_data[1]))
        minutes = ''.join(filter(lambda x: x.isdigit(), callback_data[2]))
        is_today = callback_data[3]

        # Перевірка на відповідність введених користувачем даних прийнятому формату
        if not hour.isdigit() or not minutes.isdigit():
            bot.send_message(cid,
                             " Ви надіслали порожній рядок або рядок неправильного формату. Правильний формат
```

```
ЧЧ:ММ ",
                        reply_markup=get_date_keyboard())
            return

        with ScheduleDB(config) as db:
            if db.set_auto_post_time(cid, (hour + ":" + minutes + ":" + "00").rjust(8, '0'), is_today):
                bot.send_message(cid, "Час встановлено", reply_markup=get_date_keyboard())
            else:
                bot.send_message(cid, “Сталося щось дивне, спробуйте ввести команду заново”,
                                 reply_markup=get_date_keyboard())
    except BaseException as e:
        logger.warning('callback_auto_posting: {0}'.format(str(e)))
        bot.send_message(cid, “Сталося щось дивне, спробуйте ввести команду заново”)

# ----------------------------------
#  FLASK ROUTES
# ----------------------------------

# Empty webserver index, return nothing, just http 200
@app.route('/', methods=['GET', 'HEAD'])
def index():
    return ''

# Убирает вебхук
@app.route("/remove_webhook", methods=["GET", "HEAD"])
def remove_webhook():
    bot.remove_webhook()
    return "ok", 200

# Сбрасывает вебхук
@app.route("/reset_webhook", methods=["GET", "HEAD"])
def reset_webhook():
    bot.remove_webhook()
    bot.set_webhook(url=WEBHOOK_URL_BASE+WEBHOOK_URL_PATH,
certificate=open(config["WEBHOOK_SSL_CERT"], 'r'))
    return "ok", 200

# Обработка запросов к вебхуку
@app.route(WEBHOOK_URL_PATH, methods=['POST'])
def webhook():
    if flask.request.headers.get('content-type') == 'application/json':
        json_string = flask.request.get_data().decode('utf-8')
        update = telebot.types.Update.de_json(json_string)
        bot.process_new_updates([update])
        return ''
    else:
        flask.abort(403)

# Добавление организации в базу данных
@app.route("/api/organization", methods=["POST"])
def add_organization():
    if not request.is_json:
        return "incorrect request", 403

    try:
```

```
        content = request.get_json()

        if content['key'] != config["TOKEN"]:
            return "invalid api key", 403

        data = content['data']

        answer = {
            'ok': [],
            'failed': []}

        with ScheduleDB(config) as db:
            for org_data in data:
                # Обязательные параметры запроса
                organization = org_data['organization']
                group = org_data['group']

                # Необов'язкові параметри
                faculty = org_data['faculty'] if 'faculty' in org_data else ''

                tag = db.add_organization(organization, faculty, group)
                json_data = {
                    'tag': tag,
                    'data': org_data
                }

                if tag is not None:
                    answer['ok'].append(json_data)
                else:
                    answer['failed'].append(json_data)

        return jsonify(answer), 200
    except KeyError as e:
        return 'key not found: {}'.format(str(e)), 403

# Додавання розкладу до бази даних
@app.route("/api/schedule", methods=["POST"])
def add_schedule():
    if not request.is_json:
        return "incorrect request", 403

    try:
        content = request.get_json()

        if content['key'] != config["TOKEN"]:
            return "invalid api key", 403

        data = content['data']

        answer = {'failed': []}
        with ScheduleDB(config) as db:
            for lecture in data:
                # Обов'язкові параметри запиту
                tag = lecture['tag']
                day = lecture['day']
```

```
            number = lecture['number']
            week_type = lecture['week_type']
            title = lecture['title']
            classroom = lecture['classroom']

            # Необязательные параметры
            time_start = lecture['time_start'] if 'time_start' in lecture else None
            time_end = lecture['time_end'] if 'time_end' in lecture else None
            lecturer = lecture['lecturer'] if 'lecturer' in lecture else None

            if not db.add_lesson(tag, day, number, week_type, time_start, time_end, title, classroom, lecturer):
                answer['failed'].append(lecture)
        return jsonify(answer), 200
    except KeyError as e:
        return 'key not found: {}'.format(str(e)), 403

if __name__ == '__main__':
    # Start flask server
    app.run(
        host=config["WEBHOOK_LISTEN"],
        port=config["WEBHOOK_PORT"],
        ssl_context=(config['WEBHOOK_SSL_CERT'], config['WEBHOOK_SSL_PRIV']),
        debug=True)
```

ДОДАТОК Е

Розбір Excel-файлу та збереження даних у базу даних

```
# -*- coding: utf-8 -*-
import pandas as pd
from app.parser_helpers import (parse_day, parse_time, parse_lesson_number,
                                parse_title, parse_lecturer, parse_classroom)
from app.parser_helpers import parse_date, parse_exam_data

class Parser:
    def __init__(self, file_name):
        self.file_name = file_name

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        pass

    def __parse_schedule(self, group_table):
        data = []               # Масив для збереження інформації про заняття
        for i in range(len(group_table)-1):
            # Взяття даних з комірок і значень індексів
            index = group_table.index[i][0]
            lesson_info = group_table.index[i][1]

            lesson_data = group_table[i]

            # Одне заняття описується двома рядками таблиці
            # type описує яка це частина опису заняття
            type = i % 2

            # Якщо заняття кожну неділю, то другої частини опису заняття буде її дублікат
            if type == 1 and lecture_type == 'all':
                continue

            # Тут визначається тип заняття – чи буде воно кожен тиждень, а по лише по парним, або
непарним тижням (тип odd та even)
            if type == 0:
                if group_table[i] == group_table[i+1]:
                    lecture_type = 'all'
                else:
                    lecture_type = 'odd'
            else:
                lecture_type = 'even'

            #
            if pd.isna(lesson_data):
                continue

            day = parse_day(index)
```

```
            time_start, time_end = parse_time(lesson_info)
            lesson_number = parse_lesson_number(lesson_info)

            week_type = lecture_type

            title = parse_title(lesson_data)
            lecturer = parse_lecturer(lesson_data)
            classroom = parse_classroom(lesson_data)

            lesson_structure = {
                'day': day,
                'number': lesson_number,
                'week_type': week_type,
                'title': title,
                'classroom': classroom,
                'lecturer': lecturer,
                'time_start': time_start,
                'time_end': time_end,
            }
            data.append(lesson_structure)
        return data

    def __parse_exams(self, group_table):
        data = []  # Масив для збреження інформації про екземпляри
        for i in range(len(group_table) - 1):
            exam_index = group_table.index[i]#parse_date(group_table.index[i])
            exam_info = group_table[i]#parse_exam_data(group_table[i])

            if pd.isna(exam_info):
                continue

            date = parse_date(exam_index)
            exam_data = parse_exam_data(exam_info)

            lecturer = exam_data[0] if len(exam_data) >= 1 else ''
            title = exam_data[1] if len(exam_data) >= 2 else ''
            classroom = exam_data[2] if len(exam_data) >= 3 else ''

            exam_structure = {
                'day': date,
                'title': title,
                'classroom': classroom,
                'lecturer': lecturer
            }
            data.append(exam_structure)
        return data

    def parse_schedule_from_excel(self, header_row=0):
        json_data = {}

        xls = pd.ExcelFile(self.file_name, engine='openpyxl')
        for sheet in xls.sheet_names:
            df = pd.read_excel(self.file_name, sheet, engine='openpyxl', header=header_row, index_col=[0, 1])
            df = df.loc[:, ~df.columns.str.contains('^Unnamed')]
```

```
            json_data[sheet] = {}
            for group in df.columns.values:
                group_table = df[group]

                # Парсінг занять
                json_data[sheet][group] = self.__parse_schedule(group_table)
        return json_data

    def parse_exams_from_excel(self, header_row=0):
        json_data = {}

        xls = pd.ExcelFile(self.file_name, engine='openpyxl')
        for sheet in xls.sheet_names:
            df = pd.read_excel(self.file_name, sheet, engine='openpyxl', header=header_row, index_col=0)
            df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

            json_data[sheet] = {}
            for group in df.columns.values:
                group_table = df[group]

                # Парсинг экзаменов
                json_data[sheet][group] = self.__parse_exams(group_table)
        return json_data
```

ДОДАТОК Є

Вебінтерфейс для роботи із чат-ботом

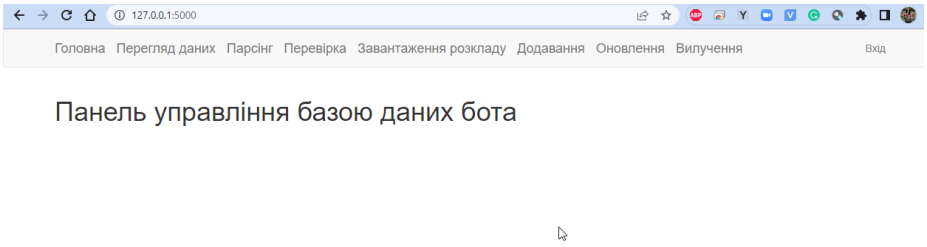

Рисунок Є.1 – Панель управління базою даних на сервері

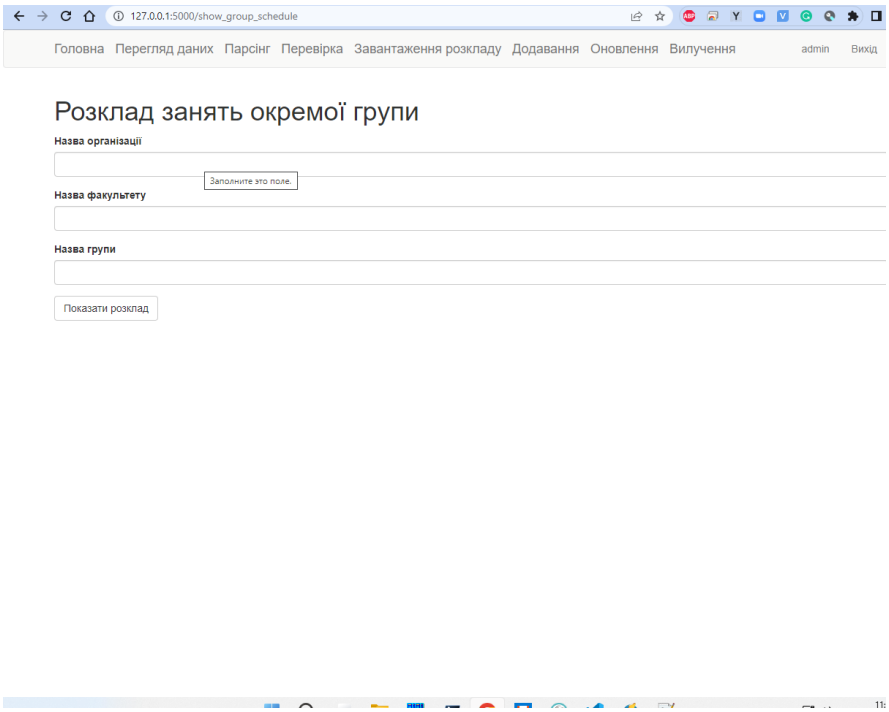

Рисунок Є.2 – Вебформа отримання розкладу окремої групи

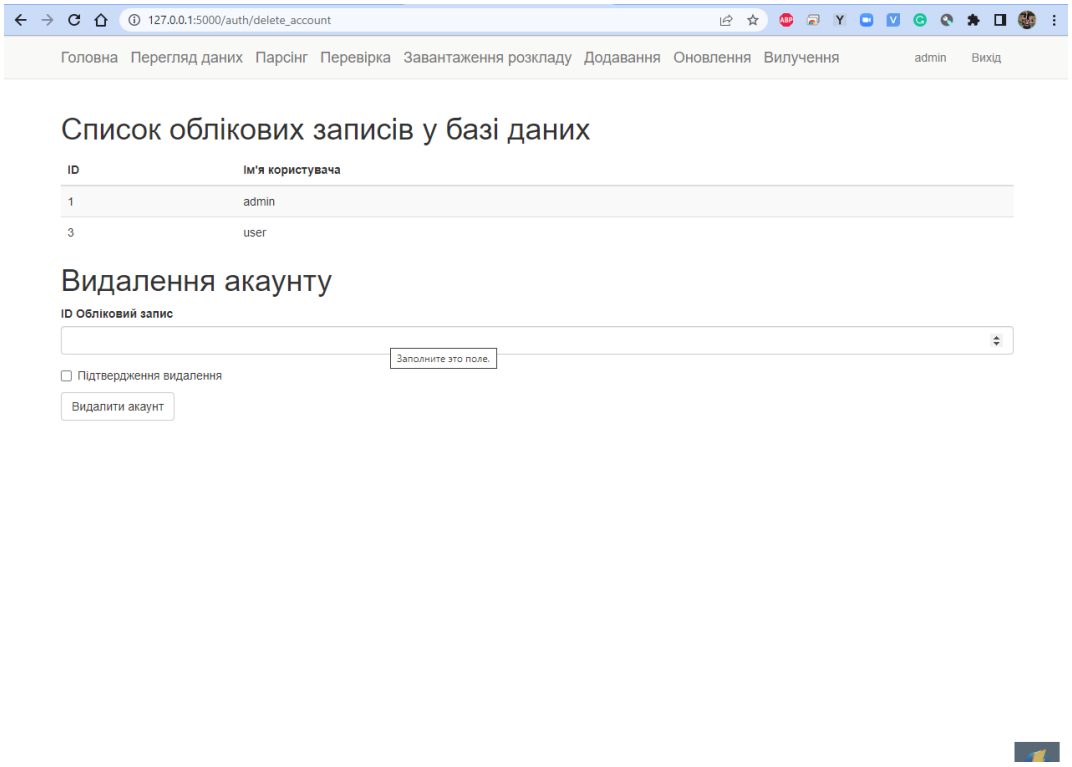

Рисунок Є.3 – Панель управління акаунтами

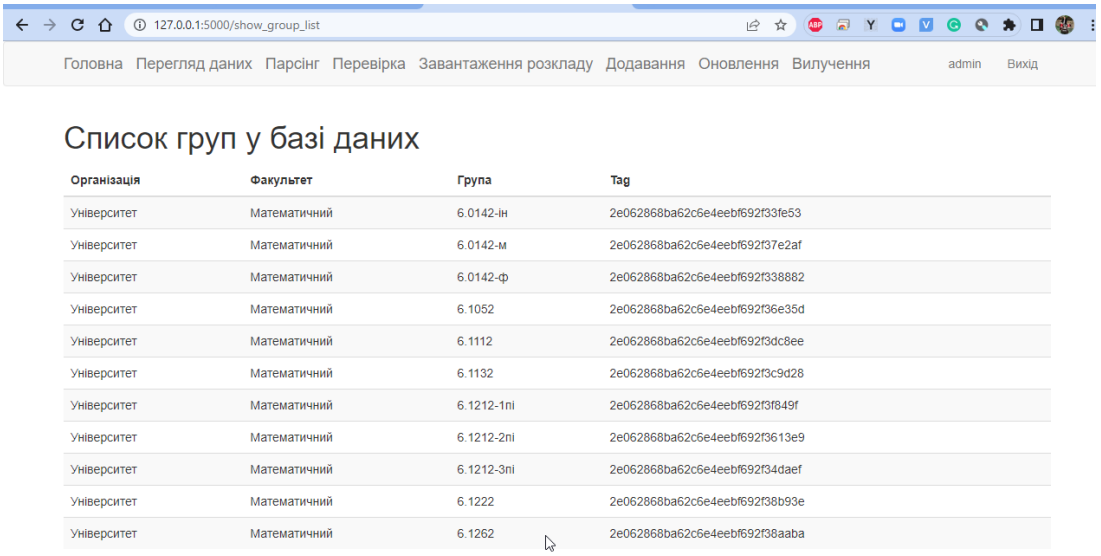

Рисунок Є.4 – Перегляд академічних груп у базі даних

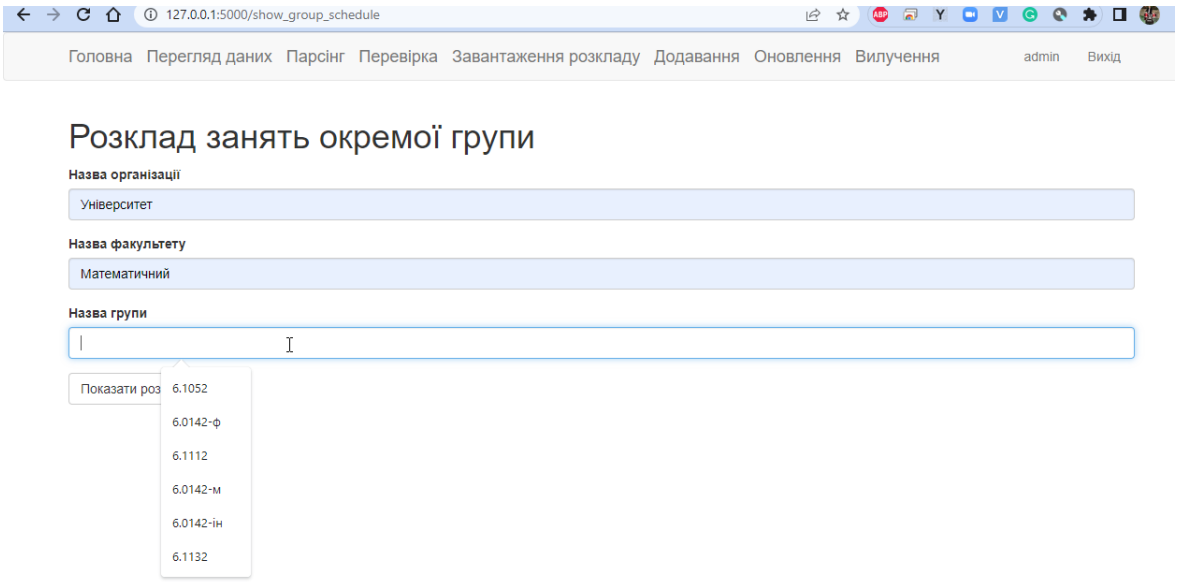

Рисунок Є.5 – Пошук розкладу занять окремої групи

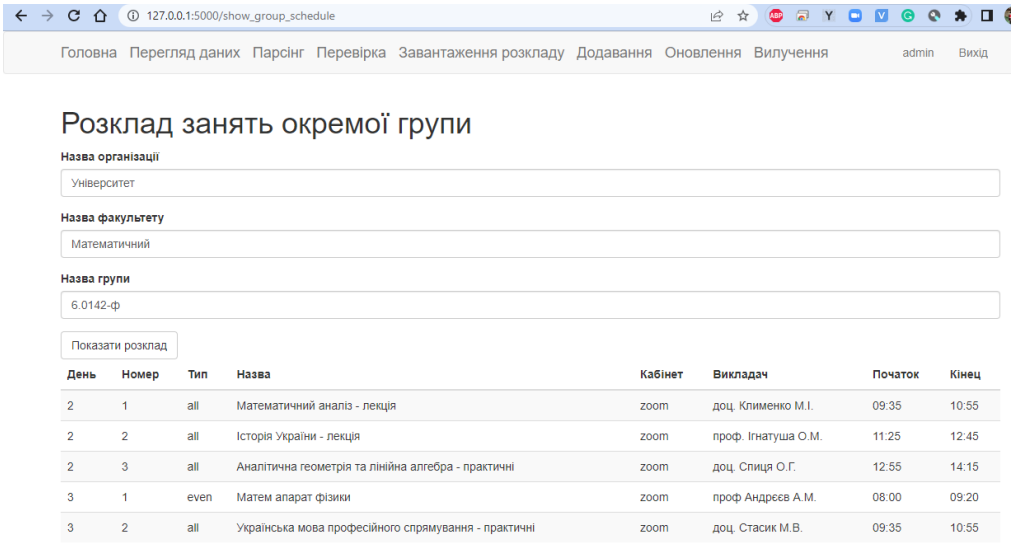

Рисунок Є.6 – Розклад занять окремої групи

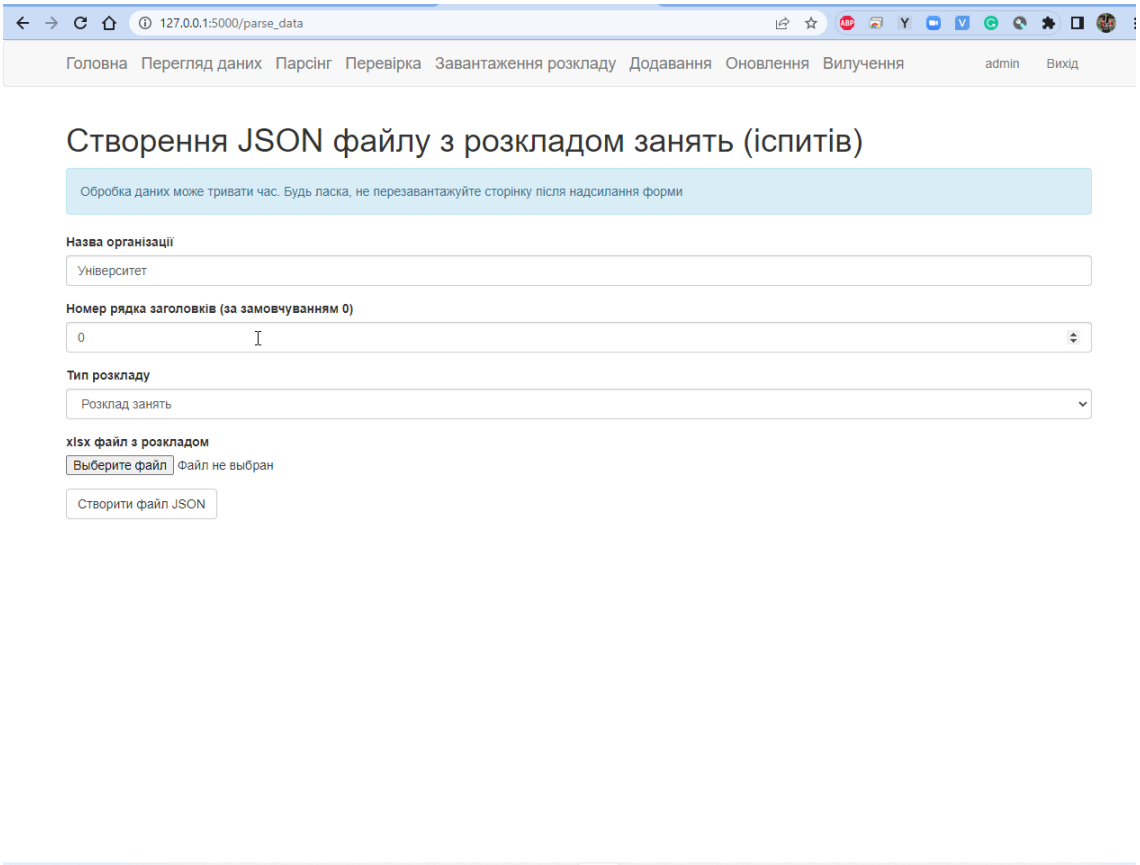

Рисунок Є.6 – Створення JSON – файлу з розкладом групи