МІНІСТЕРСТВО ОСВІТИ НАУКИ УКРАЇНИ

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО
«ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра економічної кібернетики та інженерії програмного забезпечення

ДО ЗАХИСТУ ДОПУЩЕНА

Завідувач кафедри,
д.е.н., доц.
_____ С.І. Левицький

# БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА
# ПЕРСОНАЛЬНИЙ ПОМІЧНИК УПРАВЛІННЯ РОЗКЛАДОМ ПрАТ ПВНЗ «ЗІЕІТ»

Виконав

ст. гр. ІПЗ – 118 _____ М.Д. Гнутенко

Керівник

к.т.н., доц. _____ О.А. Жеребцов

Запоріжжя

2022

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»
Кафедра економічної кібернетики та інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри,
д.е.н., доц.

_____ С.І. Левицький

17 січня 2022 р.

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ

студенту гр. ІПЗ-118 ,

спеціальності 121 - «Інженерія програмного забезпечення»

Гнутенко Максиму Денисовичу

1. Тема: Персональний помічник управління розкладом ПрАТ ПВНЗ «ЗІЕІТ» затверджена наказом № 06.1 – 50 від 15 січня 2022 р.

2. Термін здачі студентом закінченої роботи: 18 червня 2022 р.

3. Перелік питань, що підлягають розробці:

1. Провести огляд літератури, що присвячена тематиці досліджень.

2. Розглянути варіанти та технології щодо оптимізації механізмів управління розкладом.

3. Виконати огляд та порівняння аналогів систем з інформування розкладу.

4. Здійснити обґрунтований вибір використання стеку технологій для розробки проекту.

5. Розробити алгоритми парсингу вихідної інформації розкладу.

6. Здійснити проектування та програмування запропонованої системи.

7. Виконати операції життєвого циклу програмного забезпечення щодо створеного продукту.

8. Оформити звіт за результатами роботи.

4. Календарний графік підготовки випускної роботи молодшого спеціаліста.

| № етапу | Зміст | Терміни виконання | Готовність по графіку %, підпис керівника | Підпис керівника про повну готовність етапу, дата |
|---|---|---|---|---|
| 1 | Формулювання (корегування) теми випускної роботи молодшого спеціаліста та збір практичного матеріалу за темою випускної роботи | 17.01.22-18.02.22 | | |
| 2 | **І атестація** І розділ випускної роботи молодшого спеціаліста | 28.03.22-01.04.22 | | |
| 3 | **ІІ атестація** ІІ розділ випускної роботи молодшого спеціаліста | 26.04.22-29.04.22 | | |
| 4 | **ІІІ атестація** ІІІ розділ випускної роботи молодшого спеціаліста, висновки та рекомендації, додатки, реферат | 23.05.22-27.05.22 | | |
| 5 | Перевірка випускної роботи молодшого спеціаліста програмою «Антиплагіат» | 30.05.22-10.06.22 | | |
| 6 | Доопрацювання випускної роботи молодшого спеціаліста, підготовка презентації, отримання відгуку керівника і рецензії | 30.05.22-10.06.22 | | |
| 7 | **Попередній захист випускної роботи молодшого спеціаліста** | **14.06.22-18.06.22** | | |
| 8 | Подача випускної роботи молодшого спеціаліста на кафедру | за 3 дні до захисту | | |
| 9 | **Захист випускної роботи молодшого спеціаліста** | **20.06.22-24.06.22** | | |

Керівник _____ О.А. Жеребцов
(підпис) (ініціали, прізвище)

«____» _____ 2022 р.

Завдання отримав до виконання _____ М.Д. Гнутенко
(підпис студента) (ініціали, прізвище)

«____» _____ 2022 р.

# РЕФЕРАТ

Бакалаврська робота містить 104 сторінки, 61 рисунків, 2 таблиці, 30 використаних джерел.

Метою розробки є створення персонального помічника для сповіщення про розклад з персональними налаштуваннями.

Об‘єктом дослідження є сучасна система відображення розкладу занять.

Предметом дослідження є чат-бот для відображення персонального розкладу.

Здійснено детальний огляд предметної області, та сучасних аналогів, таких як чат-бот для перевірки розкладу від ЗВО НУПП, та модуль розкладу АСУНЗ Національного Медичного Університету. Виявлено, що розробка системи сповіщення про новий розклад є доцільною. Проект реалізовано у виді чат-боту для месенджеру Telegram, з використанням мови програмування Java та бібліотеки Apache POI.

Отриманий програмний продукт є простим у використанні, та гнучким у налаштуванні. Система сповіщення дозволяє своєчасно отримувати необхідну інформацію без регулярних запитів.

JAVA, APACHE POI, HIBERNATE, TELEGRAM, ПЕРСОНАЛЬНИЙ ПОМІЧНИК, РОЗКЛАД

ЗМІСТ

# ПЕРЕЛІК ТЕРМІНІВ

| Скорочення | Повна назва | Пояснення/переклад |
|---|---|---|
| БД | База даних | |
| ООП | Об'єктно-орієнтоване програмування | Одна з парадигм програмування |
| СУБД | Система управління базами даних | |
| IDE | Integrated development environment | Інтегроване середовище розробки |
| АСУНЗ | Автоматизована система управління навчальним закладом | |
| SQL | Structured Query Language | Формальна мова для взаємодії з реляційною базою даних |
| JVM | Java Virtual Machine | Середовище виконання програм, написаних на Java |
| JDBC | Java Database Connectivity | Стандарт підключення та взаємодії з базами даних |
| ORM | Object Relation Mapping | |
| API | Application Programming Interface | Опис способів взаємодії з певною системою, наприклад програмою або віддаленим сервером |
| \|FTP | File Transfer Protocol | Протокол передачі файлів у мережі |
| SSH | Secure Shell | Протокол віддаленого управління операційною системою через мережу інтернет |
| HTTPS | HyperText Transfer Protocol Secure | Захищений протокол передачі гіпертексту |
| JIT | Just In Time | Механізм компіляції виконуваного проміжного коду у машинний під час його виконання |

| | | |
|---|---|---|
| DAO | Data Access Object | Архітектурний паттерн для абстрагування від безпосередньої роботи з ресурсами, наприклад базою даних. |
| REST | Representation State Transfer | Стиль проектування взаємодії систем у мережі |
| DI | Dependency Injection | Архітектурний паттерн |
| URL | Uniform Resource Locator | Унікальне посилання на ресурс у певній системі |
| Месенджер | Програма для організації дистанційного спілкування між двома або більшою кількістю користувачів через мережу інтернет, за допомогою текстових повідомлень з можливістю відправки медіа даних | |
| Чат-бот | Програма, взаємодія з якою виконується за допомогою існуючої платформи, наприклад месенджеру, через текстові команди. Зазвичай з точки зору месенджеру, бот – це штучно створений користувач | |
| Парсинг | Процес збору, та структуризації даних з попереднім синтаксичним аналізом, з метою подальшого використання програмою або користувачем | |
| Парсер | Програма або підпрограма, яка на вхід отримує певні дані, та після процесу парсингу віддає зібрані дані у зручному для розуміння програмою або користувачем виді | |
| Кросплатформеність | Здатність програмного забезпечення працювати на декількох апаратних платформах або операційних системах | |
| Растеризація | Процес рендерингу деякої моделі, результатом якого є растрове зображення | |
| Регулярний вираз | Формальна мова для пошуку підстрок у строках | |
| Аутентифікація | Процес перевірки справжності | |
| Авторизація | Процес надання прав на виконання певних дій користувачу | |
| Endpoint | Точка призначення для деякого запиту | |
| Webhook | Метод зміни поведінки веб додатку через зворотні визови так званих веб курків – віддалених серверів зі заздалегідь відомим API | |
| Long Polling | Процес отримання даних, який передбачає періодичні запити | |

# ВСТУП

На сьогоднішній час автоматизація внутрішніх процесів відбувається майже на будь-якому підприємстві. Це значно підвищує ефективність та зручність роботи для персоналу та клієнтів. Як приклад, майже всі заклади вищої освіти (ЗВО) використовують системи електронного відображення розкладу. Це позитивно впливає на інформованість як персоналу так і студентів.

Деякі ЗВО використовують функціональні системи повної інтеграції, які добре організовані та мають зручні підсистеми відображення розкладу та інформування персоналу та студентів. Незважаючи на це, багато вищих навчальних закладів використовують або застарілі, або незручні методи відображення та інформування стосовно розкладу. Але розклад це важлива частина організацій трудового ритму навчального закладу.

На даний момент, система відображення розкладу вищого навчального закладу ПрАТ ПВНЗ "ЗІЕІТ" полягає в публікації первісних Excel таблиць на офіційному сайті. Це досить зручний метод, але він має не тільки переваги, а і недоліки. Наприклад, не всі мобільні пристрої мають можливість переглядати такий формат файлів. Для цього багатьом користувачам необхідно завантажити спеціальну програму для перегляну Excel файлів.

Актуальність даної дипломної роботи полягає в удосконалені системи публікації розкладу та реалізації оперативного інформування викладачів та студентів за допомогою персонального помічника.

# РОЗДІЛ 1
# ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА АНАЛОГІВ

## 1.1 Стан системи публікації розкладу ЗІЕІТ

Розклад для інституту та коледжу ЗІЕІТ [1] складається з декількох частин:

- Розклад занять.
- Розклад зайнятості викладачів.
- Розклад консультацій.

Кожна частина - це один або декілька документів. Розклад занять, зайнятості та консультацій формується автоматично за допомогою системи «Деканат», у виді таблиць формату Excel (.xls або .xlsx). Після цього, при необхідності редагується навчальним відділом. Така система дозволяє дуже просто публікувати розклад по двом напрямам:

- Друкувати та вивішувати на дошці розкладу занять.
- Публікувати на офіційному сайті ЗІЕІТ.

Слід зауважити, що розклад зайнятості не публікується на сайті, а відправляється кожному викладачу індивідуально через приватні канали передачі даних.

На сайті розклад можна завантажити у первісному форматі - *xls* або *xlsx*. Достатньо знайти необхідний курс на натиснути на посилання «*завантажити*». Вигляд меню з розкладом показано на рисунку 1.1.

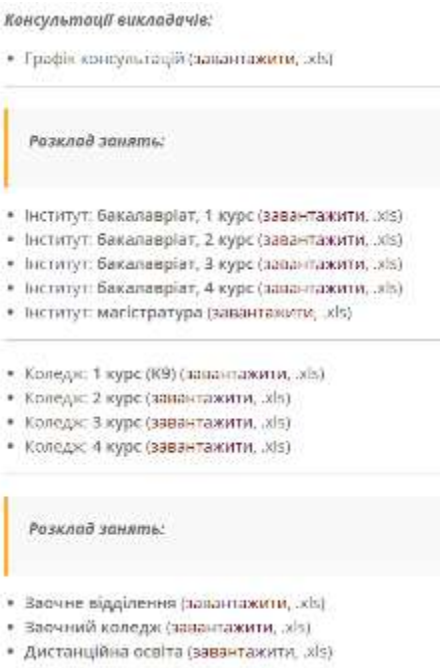


Рисунок 1.1 - Меню розкладу на сайті ЗІЕІТ

Інформування про оновлення розкладу в цій системі відсутнє. За виключенням інформування викладачів за допомогою приватних каналів обміну інформацією.

Це стан системи публікації розкладу ЗІЕІТ. Така система не нова, та використовується достатньо давно багатьма ЗВО. Такий висновок був зроблений на підставі простого дослідження систем розкладу різних навчальних закладів. Про це у наступному підрозділі.

## 1.2 Механізми електронного відображення розкладу різних ЗВО

На стан 2020 року в Україні нараховується більше 280 вищих навчальних закладів [2]. Кожен з цих навчальних закладів самостійно вирішує, який спосіб електронного відображення розкладу використовувати.

Багато навчальних закладів публікують первісні Excel таблиці на офіційному сайті. Цей спосіб використовує і ЗІЕІТ. Він має багато переваг, але має і недоліки. Наприклад, така система потребує розділення розкладу на частини –

розклад студентів та розклад викладачів. Розклад зайнятості викладачів зазвичай дуже нечитабельний, особливо для сильно навантажених тижнів.

Інші використовують сторонні системи управління розкладом, такі як АСУНЗ (Автоматизована система управління навчальним закладом) [3]. Такі системи дозволяють формувати розклад та відображувати його одразу на сайті, або в мобільному додатку, з можливістю вибирати як розклад студентів, так і розклад викладачів.

Інші навчальні заклади мають систему авторизації, тому оцінити їх систему публікації розкладу сторонньому спостерігачу неможливо.

У якості прикладів було відібрано декілька крупних ЗВО Запоріжжя та інших міст України:

- Запорізький Національній Університет.
- Запорізький Державний Медичний Університет.
- Київський Європейський Університет.

Так наприклад, Запорізький національний університет публікує розклад за схожим принципом, який використовується в ЗІЕІТ- публікує Excel таблиці з розкладом на сайті [4]. Студент має завантажити потрібний файл з розкладом та відкрити через програму зчитування таблиць формату .xls. На рисунку 1.2 приведений знімок розділу з розкладом на їх офіційному сайті.

Сам розклад має структуру, яка показана на рисунку 1.3. На рисунку приведена його частина, бо вся таблиця не поміститься у формат Word документу. Ця структура побудована за схожим принципом, який використовується в ЗІЕІТ. Зверху розташовані коди груп, зліва день тижня та номер пари. На їх перетині знаходяться самі клітини з парами, в яких зазначені ім‘я, тип пари, та ПІБ викладача.

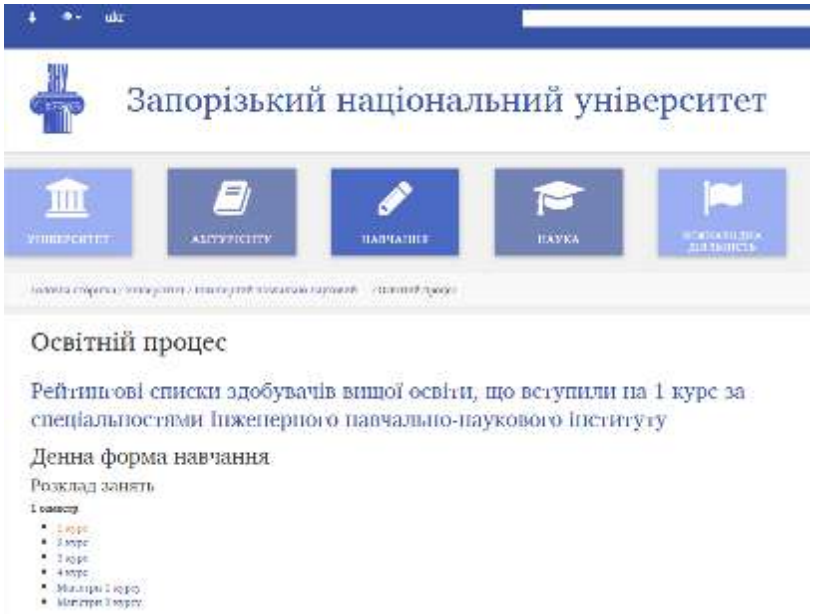


Рисунок. 1.2 - Сторінка з розкладом занять на офіціальному сайті ЗНУ

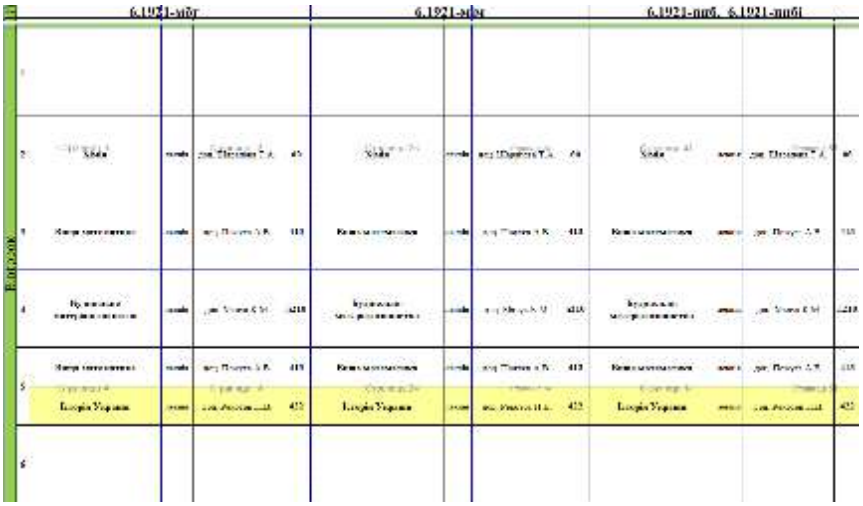

Рисунок. 1.3 - Частина розкладу занять інженерного факультету ЗНУ

А ось приклад розкладу Запорізького Державного Медичного Університету [5], на рисунку 1.4. Це також Excel таблиця, яку можна завантажити на сайті цього ЗВО. Його структура дещо відрізняється від приведеної вище. Тут немає ділення на групи, а є лише день тижня, номер та час пари. Також, в клітинах з інформацією про пару, не зазначається ПІБ викладача.


Рисунок. 1.4 - Приклад розкладу ЗДМУ

Ще один вищий навчальний заклад - Київський Європейський Університет [6]. Його система відображення розкладу також полягає в публікації Excel таблиць на офіційному сайті. Сам вид таблиці приведений на рисунку 1.5. Це розклад занять першого курсу економічного факультету. Його структура схожа зі структурою розкладу ЗІЕІТ за невеликими виключеннями.

Рисунок. 1.5 - Приклад розкладу Київського Європейського Університету

А ось приклад використання системи АУНЗ, про яку було написано раніше. Роботу цієї системи можна побачити на прикладі Національного медичного університету імені О.О. Богомольця [7]. В лівому меню можна вибрати тип розкладу – студента, викладача, аудиторії, тощо. На рисунку 1.6 можна

побачити приклад відображення розкладу студентів певного курсу та певної групи, завдяки цієї системи.

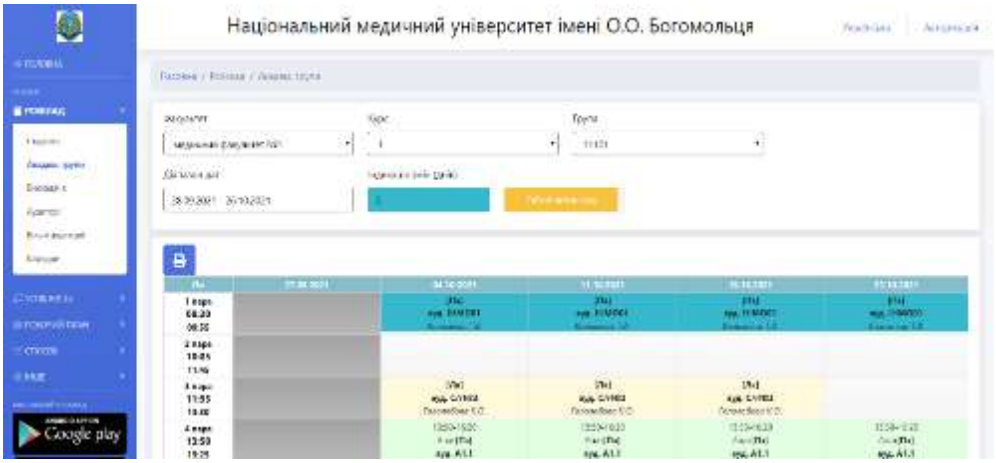


Рисунок. 1.6 - Приклад відображення розкладу студентів

А на рисунку 1.7 можна побачити відображення розкладу викладача, за допомогою системи АУНЗ. Також опціонально можна вибрати діапазон дат та інші параметри.


Рисунок. 1.7 - Приклад відображення розкладу викладача

Остання система, яку можна привести – це Telegram чат-бот для перегляду розкладу від Національного Університету "Полтавська Політехніка Імені Юрія Кондратюка" [8]. Приклад роботи з ним видно на рисунку 1.8.

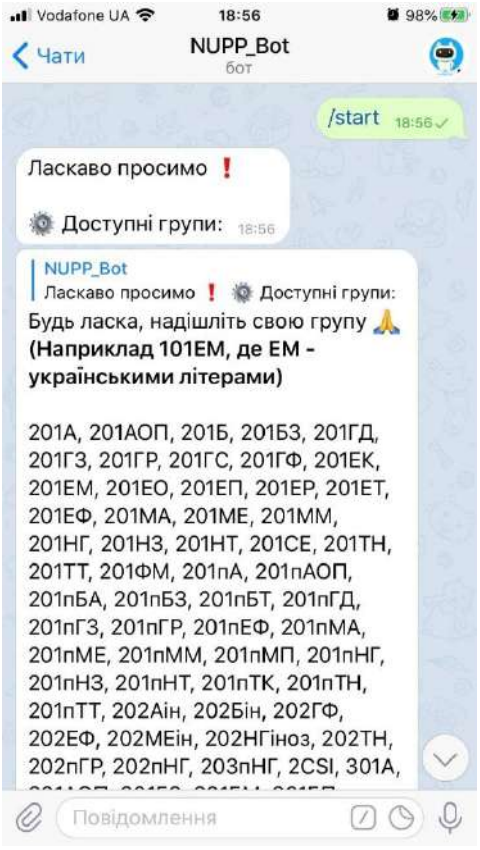


Рисунок 1.8 – Чат-бот від НУПП

Бот надає можливість перегляду розкладу різних груп. Також він має цікаву функцію нагадування про початок пари заздалегідь.

Це різні системи електронного відображення розкладу вищих навчальних закладів України. Кожна система має свої переваги та недоліки, тому вибрати кращу буде важно. Якщо будь яка система відображення розкладу виконує поставлені цілі та задовольняє персонал та студентів, можна вважати її вибір правильним.

## 1.3 Механізми автоматичного сповіщення користувачів

Механізм сповіщення призначений для оперативного інформування користувача про змінення стану певної системи, в нашому випадку - розкладу.

Існує багато способів сповіщення користувачів. Найменш затратним та швидким можна вважати сповіщення через мережу інтернет. Нижче приведені основні способи масового сповіщення користувачів через інтернет:

- електронна пошта;
- спеціалізовані додатки;
- месенджери;
- повідомлення від веб додатку через систему сповіщення браузера.

Як і з механізмами відображення розкладу, кожен спосіб має переваги та недоліки, та призначений для різних задач.

Одна з задач дипломної роботи - знайти оптимальний спосіб сповіщення користувачів про новий розклад. Для цього треба відмітити переваги та недоліки кожного з представлених методів. Ці переваги та недоліки виділені у таблиці 1.1.

Таблиця 1.1 - Переваги та недоліки різних методів сповіщення

| **Метод** | **Переваги** | **Недоліки** |
|---|---|---|
| Електронна пошта | Висока доступність<br>Можливість відправити з будь якого додатку | Необхідність постійно перевіряти повідомлення на наявність нових<br>Відсутність швидких запитів до системи |
| Спеціалізовані додатки | Повідомлення одразу видно у меню повідомлень<br>Можливість реалізувати додатковий функціонал | Необхідність встановити на телефон/пк<br>Необхідність писати окремий кросплатформений додаток |
| Месенджери | Висока доступність<br>Повідомлення одразу видно у меню повідомлень<br>Можливість реалізувати додатковий функціонал за допомогою чатботу | Необхідність встановити (але більшість користувачів вже використовують певний месенджер) |
| Повідомлення від веб додатку через систему сповіщення браузера | Висока доступність<br>Можливість реалізувати додатковий функціонал | Не всі браузери мають цю функцію або реалізують її так як треба<br>Необхідність вручну активувати цю функцію<br>Повідомлення можуть мати обмежений об‘єм інформації |

Слід зауважити, що деякі недоліки, такі як необхідність встановлювати окремі додатки, нівелюються корисністю додатку та залежить від цілей, для яких він використовується. Наприклад такі додатки як месенджер встановлені майже на всіх мобільних пристроях, бо вони корисні та виконують багато функцій. Але може бути недоцільно використовувати окремий додаток тільки для повідомлень. Так ми перейшли до аргументації вибору системи автоматичного сповіщення користувачів про оновлений розклад.

## 1.4 Аргументація вибору системи автоматичного сповіщення та відображення розкладу

У попередньому підрозділі були розглянуті переваги та недоліки різних систем сповіщення користувачів через мережу інтернет. Як вже було зазначено, деякі переваги та деякі недоліки можуть грати ключове значення при виборі цієї системи для певних задач.

Найголовнішими критеріями обрання технічних засобів для доставки інформаційних повідомлень учасникам є швидкість, доступність, кросплатформеність, легкість та зручність у використанні. Одна з задач дипломної роботи – вибрати найбільш оптимальну систему для сповіщення про новий розклад. Після невеликого дослідження був вибраний метод передачі інформації через месенджер, а саме через чат-бот для месенджеру «Telegram». Цей підрозділ виділений для аргументації цього вибору.

Для вибору оптимального методу, визначимо вимоги до системи сповіщення. Система сповіщення про новий розклад повинна:

- Завжди відображати повідомлення про новий розклад. Виключенням може бути лише відсутність підключення до мережі інтернет у користувача.
- Бути доступною для більшості людей. Це включає можливість отримувати сповіщення на різних мобільних або стаціонарних пристроях.

- Мати можливість включати та виключати повідомлення без відключення від мережі інтернет.
- Відображати розклад у найбільш доступному для швидкого перегляду форматі для більшості користувачів – растровому зображенні.

Тепер, коли вимоги для системи складені, видно, що месенджер найбільш вдало підходить до поставлених вимог. На це вказують властивості всіх популярних месенджерів:

- Повідомлення будуть завжди відображатись, за виключенням ситуацій, коли користувач вручну відключив цю можливість у додатку.
- Месенджери доступні для всіх користувачів, та стоять на більшості мобільних пристроїв та персональних комп‘ютерах.
- Завдяки простим командам, які часто доступні як кнопки, можна активувати та деактивувати повідомлення.
- Усі популярні месенджери підтримують текстові та голосові повідомлення, відправку зображень та інших медіа файлів різних форматів.

На даний момент в Україні лідирують декілька месенджерів: Telegram [9] , Viber [10], WhatsApp [11], Facebook Messenger [12].

За даними на 2020 рік, в Україні месенджером «Telegram» користується приблизно 6 млн користувачів. Це значить, що принаймні у кожного сьомого українця встановлений цей месенджер. В 2021 році статистики ще немає, але вважаючи постійний ріст кількості користувачів цього додатку у світі, можна зрозуміти, що кількість користувачів в Україні стала більше.

Незважаючи на те, що існують і інші не менш популярні месенджери, «Telegram» сильно виділяється від своїх конкурентів не в останню чергу завдяки можливості створення чат-ботів. На даний момент це самий популярний месенджер з можливістю створити чат-бота будь яким користувачем, без реєстрації компанії або інших недоступних для більшості розробників дій.

Перевагою використання чат-ботів можна вважати: миттєвість доставки повідомлення, доступність 24/7, швидке інформування в автоматичному режимі, кросплатформеність, відсутність необхідності встановлення додаткового програмного забезпечення для більшості користувачів, можливість заміни людини експертною системою або штучним інтелектом непомітно для кінцевого користувача.

Саме цьому в якості системи сповіщення, відображення, та управління підписками на розклад, був вибраний чат-бот на платформі «Telegram».

## 1.5 Проблеми отримання даних про розклад

Весь розклад ЗІЕІТ генерується автоматично системою деканат у формат Excel таблиць. Дані для генерації цих таблиць знаходяться в базі даних системи деканат. На перший погляд найбільш придатним варіантом для отримання даних про розклад є робота з зовнішнім API системи деканат, або з їх базою даних. Але це не так по деяким причинам.

Перша причина полягає у тому, що система деканат ЗІЕІТ не має ніякого зовнішнього API.

Друга причина полягає у проблемах з актуальністю даних, які можуть знаходитись в базі даних системи деканат. Ця проблема виходить з того, що вихідні генеровані таблиці частіше редагуються учбовим відділом для внесення змін у порядок та дані про пари. Після таких змін єдиним актуальним джерелом даних про розклад є фінальні редаговані таблиці.

По цим причинам єдиний спосіб отримання актуальних даних про розклад – парсинг таблиць розкладу, які публікуються на офіційному сайті ЗІЕІТ.

## 1.6 Висновки розділу

Було здійснено огляд поточної системи публікації розкладу ЗІЕІТ та інших ЗВО. Встановлено потребу у створенні системи швидкого сповіщення про новий розклад зі зручним переглядом та персональними налаштуваннями користувача.

Також було здійснено огляд різних систем сповіщення користувачів та встановлено доцільність використання месенджерів.

# РОЗДІЛ 2
# МЕТОДОЛОГІЯ ТА ЗАСОБИ РОЗРОБКИ

## 2.1 Парсинг

Парсинг (частіше називається синтаксичним аналізом [13]) – це процес розбору та структуризації природної або формальної мови. Як правило, результатом синтаксичного аналізу є синтаксична будова речення, представлена або у вигляді дерева залежностей, або у вигляді дерева складових, або у вигляді деякого поєднання першого та другого способів уявлення. Парсинг часто застосовується разом з лексичним аналізом.

У сучасній розробці «парсингом» часто називають збір даних зі вже підготовлених структур, які пройшли етапи лексичного аналізу та побудову абстрактного синтаксичного дерева. До такого «парсингу» часто відносять парсинг HTML документів. Це зумовлено тим, що для більшості формальних мов вже існують парсери, тому проводити лексичний та синтаксичний аналіз мов по типу XML та HTML частіше всього не треба.

Парсинг Excel таблиць схожий за цим принципом. Це зумовлено тим, що формат XLSX, який використовується з 2007 року – це ZIP архів з файлами у форматі XML, які містять розмітку осередків. Більш детальна інформація про формат XLSX знаходиться у наступному підрозділі.

Парсинг таблиць, можна назвати парсингом частково структурованих даних, при умові, що структура та порядок осередків таблиці незмінні. Це зумовлено незначними періодичними змінами формату даних в осередках таблиці. Наприклад, через людський фактор, ПІБ викладача, або номер аудиторії може містити, або не містити деякі елементи, такі як символи пробілу, крапки, тощо. Зважаючи на це, необхідно проводити додатковий синтаксичний аналіз

контенту таблиць, для вичленення окремих даних, таких як ПІБ викладача або номер аудиторії, з урахуванням та виправленням можливих помилок.

Універсального парсеру для будь якої таблиці не існує. Для різних типів таблиць потрібен власний підхід с власними типами даних. Використання техніки машинного навчання може бути універсальним підходом для різних типів таблиць. Але дослідження [14], які проводилися для впровадження машинного навчання у документообіг показали, що такі алгоритми є доцільними лише для строго структурованих даних, для отримання точних результатів. Так як розклад занять на даний момент не є строго структурованим, є великий ризик отримання помилок у вихідних документах, а помилки у розкладі є недопустимими. Тому машинне навчання, як універсальний метод розбору даних таблиць розкладу, на даний момент не можна вважати доцільним.

## 2.2 Рендеринг

Рендеринг [15] - термін у комп'ютерній графіці, що означає процес отримання зображення за моделлю за допомогою комп'ютерної програми. Де модель - це опис будь-яких об'єктів представлений суворо обмеженою мовою, або у вигляді структури даних. Такий опис може містити геометричні дані, положення точки спостерігача, інформацію про освітлення, ступінь наявності якоїсь речовини, напруженість фізичного поля та ін. У контексті рендерингу розкладу, моделлю є віртуальна таблиця, створена за допомогою даних, отриманих у процесі парсингу.

На даний момент розроблено безліч алгоритмів візуалізації. Існуюче програмне забезпечення може використовувати кілька алгоритмів для отримання кінцевого зображення. Для рендерингу таблиць добре підходить растеризація [16]. Кінцевим результатом растеризації є растрові зображення. Рен-

деринг таблиці дозволяє переглядати її на більшому діапазоні пристроїв, особливо мобільних, ніж перегляд у специфічному форматі, такому як xls, csv, або інших.

## 2.3 Таблиці Excel

Одні з основних форматів, які Excel підтримує для редагування – це XLS [17] та XLSX. Другий був представлений разом з Microsoft Excel 2007. Повна специфікація існує на сайті Microsoft та займає приблизно 380 сторінок [18]. Тому у цьому підрозділі буде вказана лише базова структура цих форматів.

Обидва файла це ZIP архів, який містить файли з інформацією про листи та їх розмітку. У той час, як XLS містить бінарні файли, XLSX зберігає інформацію у форматі XML.

Порожня книга, розпакована до її файлів, містить такі складові файли та папки.

- **[Content_Types].xml** - Це єдиний файл, який знаходиться на базовому рівні, розпакованого zip. У ньому перелічено типи вмісту для частин у пакеті. Усі посилання на файли XML, що входять до пакета, містяться в цьому файлі XML.
- **_rels** - Це папка Relationships, яка містить один файл XML, у якому зберігаються відносини на рівні пакета. Посилання на ключові частини файлів Xlsx містяться в цьому файлі як URI. Ці URI визначають тип зв’язку кожної ключової частини з пакетом. Це включає зв’язок з основним офісним документом, розташованим як xl/workbook.xml, та іншими частинами в docProps як основними та розширеними властивостями.
- **docProps** - Ця папка містить загальні властивості документа. Вони включають в себе набір основних властивостей, набір розширених властивостей або властивостей конкретної програми та попередній перегляд мініатюр документа. Порожня книга містить у цій папці два файли, а саме app.xml і

core.xml. Файл core.xml містить таку інформацію, як автор, дата створення, збереження та зміни. App.xml містить інформацію про вміст файлу.

- **xl** - Це основна папка, яка містить усі відомості про вміст книги.

Для кожного аркуша Excel, що міститься в книзі, є один файл XML. Ці файли XML можна знайти в папці xl/worksheets. Вся інформація, що міститься на аркуші, організована в різні розділи XML-файлу.

## 2.4 Регулярні вирази

Регулярні вирази – це формальна мова для пошуку підстрок у строках та маніпуляцій з ними. Для пошуку використовується рядок-зразок (pattern), що складається з символів та мета символів і задає правило пошуку.

Шаблон регулярного виразу складається із звичайних символів, наприклад `abc`, або комбінацій звичайних та спеціальних символів, наприклад `ab*c` або `Chapter (\d+)\.\d*`. Останній приклад включає дужки, які використовуються для позначення групи. Шаблон усередині групи обробляється як єдине ціле.

У разі коли пошук відповідності вимагає чогось більшого, ніж пряме зіставлення, наприклад знаходження послідовності символів, або знаходження пробілу, шаблон включає спеціальні символи.

Регулярні вирази широко використовуються для визначення правил формальних мов в різних трансляторах. Регулярні вирази добре підходять для аналізу та вичлення даних в осередках таблиць розкладу.

## 2.5 Архітектура «Клієнт-Сервер»

Архітектура клієнт-сервер – це обчислювальна модель, в якій сервер розміщує, постачає та керує більшістю ресурсів і послуг, які споживає клієнт. Цей тип архітектури має один або кілька клієнтських комп’ютерів, під’єднаних до центрального сервера через локальну мережу або мережу Інтернет. Ця система

спільно використовує обчислювальні ресурси. Архітектура клієнт/сервер також відома як модель мережевих обчислень або мережа клієнт/сервер, оскільки всі запити та послуги доставляються через мережу [19]. Приклад структури клієнт-сервер показана на рисунку 2.1.

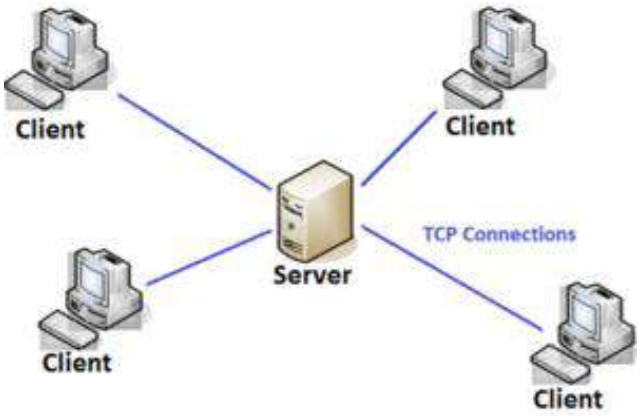


Рисунок 2.1 – Архітектура Клієнт-Сервер

Кожен сервер у такій архітектурі надає деякий інтерфейс, за яким клієнти можуть надсилати запити до цього серверу. У разі вхідного запиту сервер обробляє його, та відсилає відповідь, на яку чекає клієнт.

## 2.6 Архітектурні принципи REST

REST (абревіатура Representation State Transfer) – це набор правил, або архітектурний стиль взаємодії компонентів додатку у мережі. Такі правила дозволяють розробляти масштабовані системи, які можуть легко обмінюватись даними між собою. REST не обмежує архітектора у виборі технології передачі даних, але вводить 5 обов‘язкових правил для такої системи [20]:

1. Однорідність інтерфейсу.
2. Архітектура Клієнт-Сервер. REST передбачає саме таку архітектуру. Це забезпечує розділення проблем, що допомагає компонентам клієнта і сервера розвиватися незалежно.

3. Відсутність стану. Кожен запит від клієнта до сервера повинен містити всю інформацію, необхідну для розуміння та виконання запиту. Стан може зберігати клієнт, але не сервер
4. Кешування. Клієнт за необхідності може кешувати результат деяких запитів для підвищення швидкості роботи системи.
5. Шари. Багатошарова система дозволяє структурована так, що кожен компонент не може бачити за межами свого шару. Це дозволяє масштабувати систему без риску зламати щось в іншому місті.

REST не передбачає, у якому виді будуть передаватись дані. У сучасних системах дані частіше передаються у виді JSON строк, ріже у виді XML. У більш навантажених системах часто використовують Protobuf, який дозволяє описувати дані на рідній мові програмування, та серіалізувати їх у максимально стислий масив байтів.

Також, REST не передбачає, який протокол передачі даних буде використовуватись. Але на практиці частіше всього використовується HTTP, як один з найпопулярніших прикладів протоколу типу клієнт-сервер. Для бінарних даних також використовують протокол HTTP/2, так як він є більш оптимізованим варіантом ніж HTTP/1.1, так я є бінарним, та підтримує потоки даних.

## 2.7 Чат-боти

Чат-бот, або віртуальний співбесідник – це програма, управління якою здійснюється через чат іншої програми, наприклад месенджеру, соц. мережі, тощо. Управління чат-ботом може здійснюватися різними методами, такими як текстові команди, віртуальна клавіатура, запити на звичайній мові, голосові запити, тощо.

Сьогодні чат боти застосовують для самих різних потреб як великі компанії, так і для індивідуальних цілей. Для прикладу, на рисунку 2.2 приведений скриншот роботи офіційного чат боту онлайн банку Монобанк [21]. Такий чат бот значно економить час користувачів.

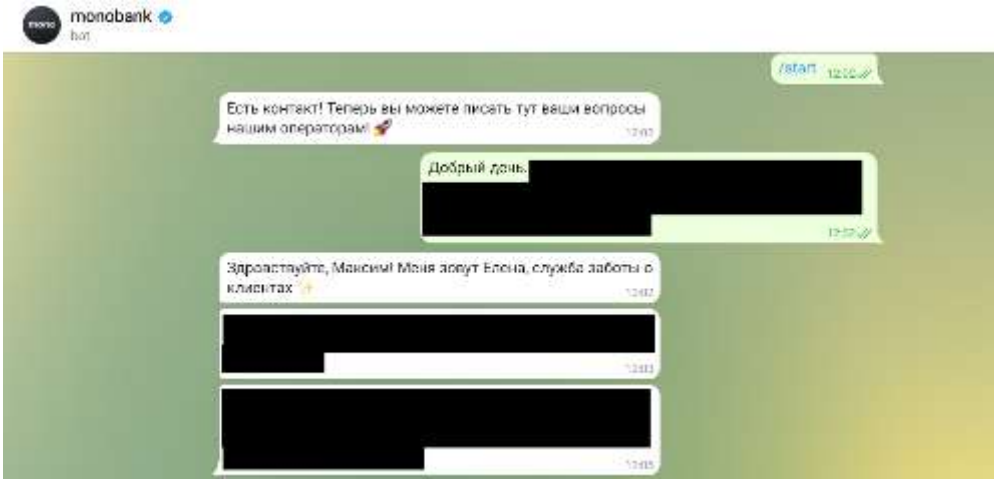


Рисунок 2.2 - Чат-бот онлайн банку Монобанк

Чат боти можна поділити на ті що навчаються, та «заскриптовані». Боти, що навчаються використовують машинне навчання для покращення результатів своїх відгуків на запити користувачів. Прикладом таких чат ботів є популярний проект «A.L.I.C.E» від компанії «Яндекс». Заскриптовані чат боти зазвичай використовують в своєї основі машину станів, яка є кінцевим автоматом, а значить відповіді таких ботів будуть завжди передбачуваними та більш точними. Такі боти використовуються для надання послуг певного вузького спектру. Прикладом такого боту є офіційний бот Telegram «BotFather» для регістрації власних чатботів. Його робота приведена на рисунку 2.3. Управління цим та подібними ботами зазвичай здійснюється за допомогою команд та віртуальної клавіатури.

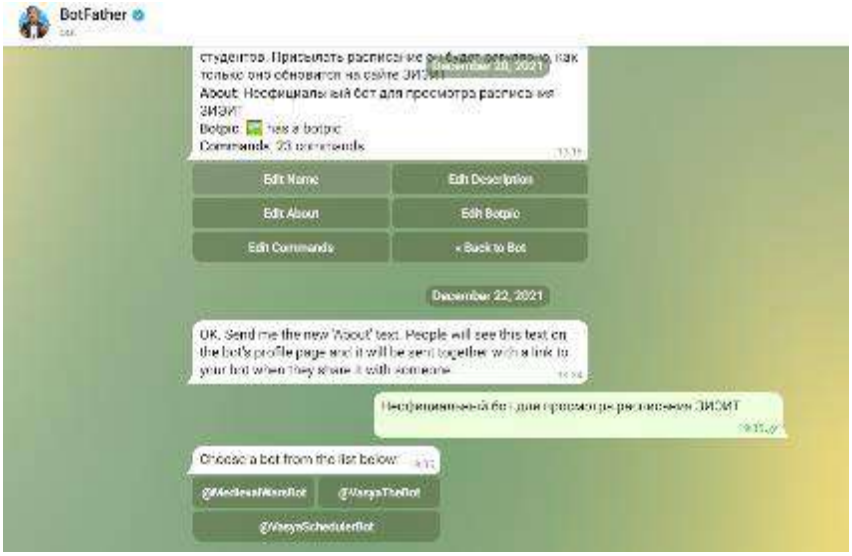


Рисунок 2.3 - Чат бот BotFather

## 2.7 Інструментарій для розробки

### 2.7.1 Мова програмування Java

Java – це строго типизована мова загального призначення, розроблена компанією Sun Microsystems, та пізніше придбана компанією Oracle. Програми Java зазвичай транслюються у спеціальний байт-код, тому вони можуть працювати на будь-якій комп'ютерній архітектурі, для якої існує реалізація віртуальної Java-машини.

Незважаючи на те, що програми, написані на Java, компілюються не у машинний код, а у код, який може виконувати лише віртуальна машина, швидкість таких програм не дуже відрізняється від аналогічних, написаних на мовах, які транслюються у машинний код. Це можливо завдяки JIT компілятору, який транслює байткод Java машини у машинний код під час роботи програми.

Зараз Java широко використовується для створення backend-у та у Enterprise розробці. Також неможливо не згадати про розробку під Android, де Java була основною мовою для написання додатків, поки на її місто не прий-

шла мова Kotlin. Однак, попри це, програми для Android все одно є програмами для віртуальної машини Java, тільки не для стандартної JVM, а для Android Runtime (раніше Dalvik).

У мові Java неможливо явне видалення об'єкта з пам'яті - натомість реалізовано складання сміття. Традиційним прийомом, що дає збирачеві сміття «натяк» на необхідність звільнення пам'яті, є привласнення змінної порожнього значення null, що може виявитися ефективним при необхідності звільнити об'єкт, що не потрібний, посилання на який зберігається в довгоживучому об'єкті. Це, однак, не означає, що об'єкт, замінений значенням null, буде неодмінно і негайно вилучений, але є гарантія, що цей об'єкт буде видалено саме в майбутньому. Даний прийом лише усуває посилання на об'єкт, тобто відв'язує покажчик від об'єкта в пам'яті. Об'єкт не буде видалений збирачем сміття, поки на нього вказує хоча б одне посилання зі змінних або об'єктів, що використовуються.

Java є об‘єктно орієнтованою мовою. Будь-яка функція може існувати лише всередині класу. На методи перетворилися й стандартні функції. Наприклад, Java немає функції sin(), а є метод Math.sin() класу Math (що містить, крім sin(), методи cos(), exp(), sqrt(), abs() і багато інших). Конструктори Java не вважаються методами. Деструкторів у Java немає.

### 2.7.2 Мова розмітки YAML

YAML – це мова для структуризації даних, розроблений у 2001 році Кларком Евансом. Широко використовується як мова для опису конфігураційних файлів для різних програм. Ця мова відрізняється від інших високою читаністю та простотою опису різних структур даних. Основні елементи YAML [22]:

- Потоки YAML використовують друковані Unicode-символи, як UTF-8, так і UTF-16.
- Відступи з прогалин (символи табуляції не допускаються) використовуються для позначення структури.

- Коментарі починаються з символу решітки (#), можуть починатися в будь-якому місці рядка і продовжуються до кінця рядка.
- Списки позначаються початковим дефісом (-) з одним членом списку на рядок, або члени списку укладаються у квадратні дужки ([ ]) і поділяються комою та пробілом (, ).
- Асоціативні масиви представлені двокрапкою з пробілом (: ) у вигляді «ключ: значення», по одній парі ключ-значення на рядок, або у вигляді пар, укладених у фігурні дужки та розділених комою та пробілом (, ).
- Ключ в асоціативному масиві може мати як префікс знак (?), що дозволяє вказати складний ключ, наприклад представлений у вигляді списку.
- Рядки записуються без лапок, однак можуть бути укладені в одиночні або подвійні лапки.
- Всередині подвійних лапок можуть бути використані екрановані символи в С-стилі, що починаються зі зворотною косою (\).
- YAML дозволяє задавати підстановки за допомогою якорів & та псевдонімів (*).

У лістингу 2.2 приведений приклад із базових конструкції цієї мови розмітки. Інші елементи та синтаксичні конструкції мови можна знайти у повній специфікації на офіційному сайті [23].

Лістинг 2.1 – Базові елементи мови

```
string: "Строка"
number: 32
floating_point: 0.8
map:
  key: "Значення"
  key2: 16
string_list:
  - "Строковий елемент 1"
  - "Строковий елемент 2"
map_list:
  - name: "Елемент 1"
    position: 1
  - name: "Елемент 2"
    position: 2
```

### 2.7.3 Збиральник проектів Gradle

Програми на Java компілюються за допомогою компілятора javac. Для середніх та великих програм, які складаються з десятків, або сотен класів, та мають залежність від інших компонентів, ручне збирання за допомогою тільки компілятора дуже складне та не доцільне. Для рішення цієї проблеми були створені різні збиральники проектів. Збиральник проектів – це програма, яка автоматизує більшість рутинних завдань зі збірки проекту, такі як управляння залежностями, цілями для збірки, оптимізація збірки, тощо.

Для Java існує чимало збиральників. Їх можна поділити на імперативні та декларативні. До імперативних можна віднести вже майже невикористовуваний збиральник Apache Ant. Він приймає на вхід XML файл з детальним описом задач для збірки. Імперативним він є тому, що розробнику необхідно чітко вказати як потрібно зібрати проект.

Після Apache Ant мейнстримом став декларативний збірник Maven. Він також використовує XML для опису проекту, але на відміну від Ant, для базових проектів Maven дозволяє описати лише базові речі, наприклад ім‘я проекту, список залежностей, та інші властивості проекту. Для більш складних проектів є безліч плагінів, як можна підключити до проекту у тому ж файлі. Але використання XML для опису збірки більш великих проектів приводить до громіздких та складно читаних файлів. На даний час ще одною з найбільш популярних систем збірки проектів є Gradle, який вирішує більшість проблем, присутніх у попередніх збірниках.

Gradle — система автоматичної збірки, побудована на принципах Ant та Maven, але надає DSL на мовах Groovy або Kotlin замість традиційної XML-подібної форми представлення конфігурації проекту. Gradle був розроблений для багатопроектних збірок, що розширюються, і підтримує каскадну модель розробки, визначаючи, які компоненти дерева збірки не змінилися і які завдання, залежні від цих частин, не вимагають перезапуску. У світі збиральників проектів це називається інкрементальною збіркою. Він також підтримує

використання репозиторіїв Maven для підключення залежностей. Писати скрипти збірки з використанням Gradle DSL дуже просто та ефективно, навіть якщо для збірки необхідно виконувати нестандартні задачі.

### 2.7.4 Середовище розробки Intellij IDEA

У професійній розробці, для підвищення зручності та швидкості роботи з програмним кодом, проектуванням та рефакторингом, застосовуються програмні засоби, які називаються IDE (Integrated Development Environment).

Для мови програмування Java існує чимало IDE різної якості. За даними опиту [24] за 2020 рік, на ринку домінують три Java IDE – Intellij IDEA (62%), Eclipse (20%), та NetBeans (10%). Популярність першої обґрунтований. Під час роботи, IDEA сильно виділяється такими зручними можливостями, як:

- Розуміння контексту. IDE розуміє, на яка частина програмного коду редагується у даний час, без необхідності виділяти її.
- Розумне авто доповнення. IDE робить авто доповнення з розумінням контексту, пропонуючи ім‘я змінних, функцій та інших частин коду, зважаючи на типи даних, їх ім‘я, попередній досвід, та інші фактори.
- Language injection. Коли в одному файлі редагується код на різних мовах, IDEA розуміє це, та продовжує надавати розумні доповнення та перевірки коду.
- Потужний рефакторинг. Детальний аналіз коду дає можливість проводити рефакторинг, який буде каскадно змінювати весь вихідний код при необхідності. У тому числі, якщо він на різних мовах.
- Різні мілкі, але дуже зручні особливості поведінки, такі як авто доповнення синтаксичних конструкцій кавичок, дужок, відступів, автоматичне збереження файлу без натискання Ctrl+S, тощо.

Такі особливості, багато інтеграцій з іншими сервісами, зручний інструментарій для проектування, та стабільність роботи роблять IDEA майже стандартом для професійної розробки Java в більшості компаній на даний момент.

Звичайно, такі можливості не безкоштовні для апаратного забезпечення. IDEA потребує значно більше оперативної пам‘яті, ніж її аналоги. Це зумовлено детальним аналізом всього проекту для підтримки інструментарію, описаного раніше. Зазвичай IDEA потребує від 1.5 до 2 ГБ оперативної пам‘яті. Але на сучасних комп‘ютерах ця вимога більш ніж виконувана.

## 2.8 Огляд використовуваних бібліотек

### 2.8.1 Guice

Guice - це простий DI (Dependency Injection) фреймфорк для мови програмування Java, розроблений компанією Google [25]. Dependency Injection – це патерн, який широко використовується при написані середніх та великих додатків на різних мовах програмування, в тому числі і на Java. Такий патерн значно полегшує управління залежностями, коли програма складається з великої кількості класів, які залежать один від одного.

Guice, як і більшість DI фреймворків для Java, працює за допомогою рефлексії, інструменту Java, який надає можливість зчитувати та модифікувати мета інформацію класів, методів, атрибутів та інших частин програми під час її виконання. При використанні патерну DI, об'єкти приймають свої залежності через конструктор. Тому, щоб створити об'єкт, спочатку потрібно створити його залежність. Однак, щоб створити кожну залежність окремо, потрібно, у свою чергу, створити її залежність. Таким чином, насправді нам потрібно створити граф залежностей.

Створення графів об'єктів вручну дуже трудомістко, та може призвести до помилок і ускладнює тестування. Guice може створити граф об'єктів за розробника. Але спочатку Guice потрібно налаштувати, щоб він створив граф саме так, як потрібно [26].

### 2.8.2 Log4J

Log4J – це бібліотека для логування (запису подій програми у деякий файл), створена організацією Apache. Досвід показує, що логування є важливою складовою циклу розробки, бо воно надає точний контекст про запуск та роботу програми.

Логінг має свої недоліки. Він може уповільнити роботу програми. Якщо записувати події занадто часто, це може призвести до небажаного переповнення журналу подій. Щоб усунути ці проблеми, log4j розроблено як надійний, швидкий і розширюваний. Оскільки ведення журналів рідко є основним напрямком програми, API log4j прагне бути простим для розуміння та використання.

API для Log4j відокремлений від реалізації [27], що дає зрозуміти розробникам додатків, які класи та методи вони можуть використовувати, забезпечуючи сумісність наперед. Це дозволяє розробникам Log4j покращити впровадження безпечно та сумісним чином.

Основним об‘єктом для логування є Logger. Він має свої методи для запису подій у журнал. Кожна подія має свій рівень. Рівень події визначає її вивимість у журналу або у терміналі при певних налаштуваннях. Бібліотека має наступні стандартні рівні: DEBUG, INFO, WARN, ERROR, FATAL.

Програми, що використовують API Log4j2, запитуватимуть об‘єкт Logger із певним іменем у LogManager. LogManager знайде відповідний LoggerContext, а потім отримає з нього об‘єкт Logger. Якщо Logger необхідно створити, він буде пов’язаний з LoggerConfig, який містить те саме ім’я, що й Logger, або ім’я батьківського пакета, або кореневий LoggerConfig. Об’єкти

LoggerConfig створюються з декларацій Logger у конфігурації. LoggerConfig пов'язаний з додатками, які фактично доставляють LogEvents. Приклад отримання об‘єкту логеру приведений у лістингу 2.2.

Лістинг 2.2 – Приклад створення об‘єкту Logger.

```
Logger rootLog = LogManager.getRootLogger();
Logger namedLog = LogManager.getLogger("wombat");
Logger classLog = LogManager.getLogger(SomeClass.class);
```

Після історії про знайдені вразливості цієї бібліотеки, рекомендується використовувати лише останню версію, де ці вразливості вважаються виправленими.

### 2.8.3 Hibernate

Hibernate — бібліотека для мови програмування Java, призначена для вирішення задач об’єктно-реляційного відображення (ORM). Є самою популярною реалізацією специфікації JPA (Java Persistence API).

Бібліотека не тільки вирішує задачу зв'язку класів Java з таблицями бази даних (типами даних Java і типами даних SQL), а також надає засоби для автоматичного генерування та оновлення набору таблиць, побудови запитів та обробки отриманих даних. Бібліотека дозволяє значно зменшити час розробки, яке зазвичай витрачається на ручне описування коду SQL та JDBC.

Hibernate забезпечує використання мови, схожої на SQL, названої Hibernate Query Language (HQL), яка дозволяє виконувати SQL-подібні запити, записані поруч з об’єктами даних Hibernate. Запити критеріїв надаються як об'єктно-орієнтована альтернатива HQL.

Відображення Java класів із таблицями баз даних здійснюється за допомогою Java-анотацій, або конфігураційних XML-файлів (застарілий метод). Використовуючи файл XML Hibernate, можна створити скелет вихідного коду

для класу тривалого зберігання. В цьому немає необхідності, якщо використовується анотація. Hibernate може використовувати файл XML або анотації для підтримки схем баз даних [28].

### 2.8.4 Apache POI

Apache POI – це проект [29], який надає бібліотеки Java для читання та запису файлів у форматах Microsoft Office, таких як Word, PowerPoint та Excel. Бібліотека для роботи з форматами Excel надає єдиний інтерфейс для різних форматів, таких як XLS та XLSX. Apache POI це не одна бібліотека. Вона складається з наступних компонентів, які ожна використовувати разом або окремо:

- HSSF (Horrible Spreadsheet Format). Компонент читання та запису файлів MS-Excel, формат XLS.
- XSSF (XML Spreadsheet Format). Компонент читання та запису файлів MS-Excel, формат XLSX.
- HPSF (Horrible Property Set Format). Компонент отримання наборів властивостей файлів MS-Office.
- HWPF (Horrible Word Processor Format). Компонент для читання та запису файлів MS-Word, формат DOC.
- XWPF (XML Word Processor Format). Компонент читання та запису файлів MS-Word, формат DOCX.
- HSLF (Horrible Slide Layout Format). Компонент для читання та запису файлів PowerPoint, формат PPT.
- XSLF (XML Slide Layout Format). Компонент для читання та запису файлів PowerPoint, формат PPTX.
- HDGF Horrible DiaGram Format Компонент роботи з файлами MS-Visio, формат VSD.
- XDGF XML DiaGram Format Компонент роботи з файлами MS-Visio, формат VSDX.

Apache POI надає вам такі інтерфейси, як Workbook, Sheet, Row, Cell, тощо. Ці інтерфейси реалізовані конкретними класами, такими як HSSFWorkbook, HSSFSheet, HSSFRow, HSSFCell. Класи, які реалізують роботу з новим форматом XLSX мають схожу назву, але з префікс XSSF.

### 2.8.5 TelegramBots

Telegram API побудоване на протоколі HTTP. Взаємодія з цим API проходить через посилання HTTP запитів на певний адрес з тілом у форматі JSON. Для того, щоб спростити цей процес, та зменшити кількість помилок, було створено багато бібліотек для різних мов програмування, які надають абстракції для роботи з API не напряму, а через певний інтерфейс. Для мови програмування Java така бібліотека також існує. Вона називається TelegramBots.

Бібліотека TelegramBots створена для розробки чат ботів для месенджеру Telegram. Вона підтримує 2 методи отримання оновлень (запитів) від користувачів:

1. Webhooks. Отримання миттєвих оновлень на власний сервер за допомогою веб хуків. Для цього обов‘язково треба мати HTTP сервер та сертифікат для нього, для роботи через протокол HTTPS.
2. Long Polling. Отримання оновлень методом періодичних запитів до серверу. Не потребує розгортання HTTP серверу, але отримання оновлень не миттєве та більш затратно через постійне відкриття та закриття з‘єднання.

Бібліотека підтримує останні версії Telegram API та дозволяє працювати з повідомленнями, прикріпленими файлами, аудіо та відео, отримувати інформацію про користувача, який розмовляє з ботом, тощо.

## 2.9 Висновки розділу

Було здійснено огляд різних методів та засобів розробки, таких як парсинг, рендеринг, регулярні вирази. Здійснено огляд двох форматів таблиць Excel та їх внутрішню структуру. Обґрунтовано вибір мови програмування та бібліотек для розробки програми.

Для розробки використовується мова програмування Java, з використанням таких бібліотек як TelegramBots, Apache POI, Hibernate, Log4j, Guice. В якості збиральника проекту було вибрано Gradle. Здійснено огляд цих бібліотек та виявлено особливості роботи з ними.

Здійснено огляд середовища розробки Intellij IDEA Community Edition, яка була вибрана для розробки програмного продукту. Для створення конфігурації програми було обрано мову розмітки YAML.

# РОЗДІЛ 3
# РОЗРОБКА ТА ІНТЕГРАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Проектування

### 3.1.1 Загальна архітектура

До архітектури чат боту були поставлені наступні вимоги:

1. Масштабованість.
2. Розділення обов‘язків.
3. Незалежність окремих частин (модулей).

Програма буде складатись з окремих модулей, які взаємодіють між собою, та мають внутрішні компоненти. Всього можна виділити 5 модуля:

1. Модуль даних.
2. Завантажувач розкладу.
3. Рендер розкладу.
4. Ядро боту.
5. Веб сервер.

Модуль даних відповідає за доступ до даних користувачів. Він дозволяє іншим модулям абстрагуватись від способу збереження даних та має зв‘язок з базою даних певного типу. Для збереження даних буде використовуватись реляційна база даних. Важливо враховувати, що тип бази даних може бути змінений з часом, тому тісного зв‘язку з певною СУБД не повинно бути. Для цього модуль даних можна поділити на декілька рівнів. На першому рівні буде робота з чистими SQL запитами. Для цього в Java для існує стандарт JDBC. На другому рівні знаходиться ORM фреймфорк Hibernate. Він дозволить абстрагуватись від певного типу БД, бо має власну мову, схожу з SQL. На верхньому

рівні розташовані DAO класи. Саме з цими класами буде працювати інші модулі та сервіси. Такі рівні дозволять абстрагувати роботу з даними та зробить модуль даних більш гнучким.

Завантажувач розкладу відповідає за завантаження розкладу з репозиторію, та його парсинг. Сбір даних з таблиць розкладу проводиться за допомогою бібліотеки Apache POI, яка описувалась в 2 розділі.

Рендер розкладу відповідає за представлення розкладу у виді таблиці, яку бачить кінцевий користувач. Він залежить від певного типу рендеру. За умовченням для рендерингу використовується безкоштовна версія бібліотеки Aspose Cells.

Ядро боту відповідає за взаємодію з кінцевим користувачем через Telegram API. Воно включає такі компоненти як командний інтерфейс, граф команд, який представляє собою машину станів, та таймер, який взаємодіє з менеджером розкладу та відправляє оновлений розклад кінцевому користувачу. Через обмеження платформи Telegram, яка дозволяє відправляти лише 30 повідомлень в секунду, модуль має мати інтерфейс відправки повідомлень який буде складувати їх у чергу, для відправки, враховуючи це обмеження.

Веб сервер відповідає за обробку запитів з панелі керування. Він має бути побудований за шаблоном REST, та працювати на протоколі HTTP/1.1. Для обробки запитів використовуються контролери, які посилають команди на зміни менеджеру розкладу, якщо запит стосується зміни у налаштуваннях.

Ці модулі не взаємодіють між собою безпосередньо, а використовують для цього окремі сервіси. Кожен сервіс відповідає за окрему групу функцій. Так, можна виділити 2 сервіса:

1. Менеджер підписок. Відповідає за доступ до підписок користувачів на розклад. «Розмовляє» з модулем даних, так як треба отримувати та змінювати дані про підписки на розклад.
2. Менеджер розкладу. Відповідає за доступ до розкладу, який був розпарсений модулем Schedule Loader.

Важливо враховувати, що бот повинен отримувати дані або запити від інших, віддалених сервісів. В архітектурі позначимо їх як зовнішні модулі. До таких модулей відносяться:

1. Віддалений репозиторій з розкладом занять.
2. Сервер Telegram. Віддає дані про запити користувачів під час «розмови» з ботом.
3. Панель керування. Є клієнтом, в не сервером, але важливо враховувати як ще одну точку входу.

На рисунку 3.1 візуально представлена архітектура зі всіма модулями, які були виділені раніше.

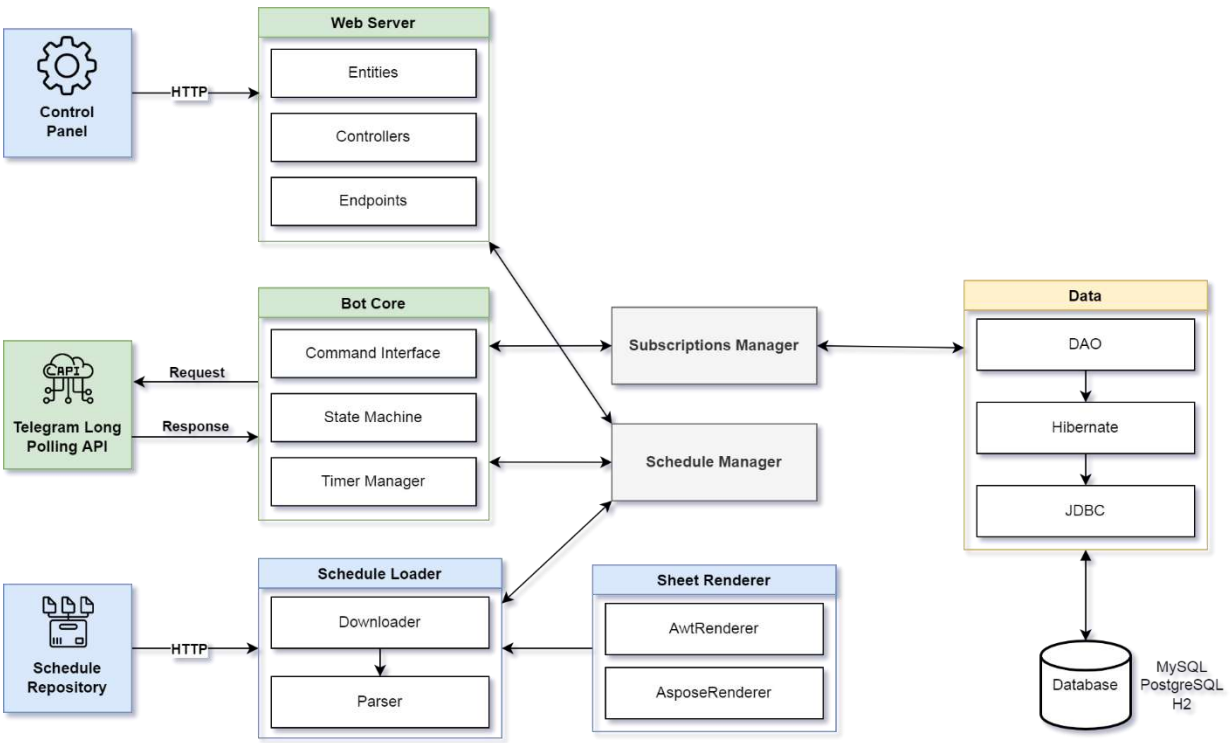


Рисунок 3.1 – Архітектура чат боту

### 3.1.2 База даних

Для зберігання даних використовується реляційна модель. На рисунку 3.2 позначена структура таблиць бази даних бота та їх відносини.

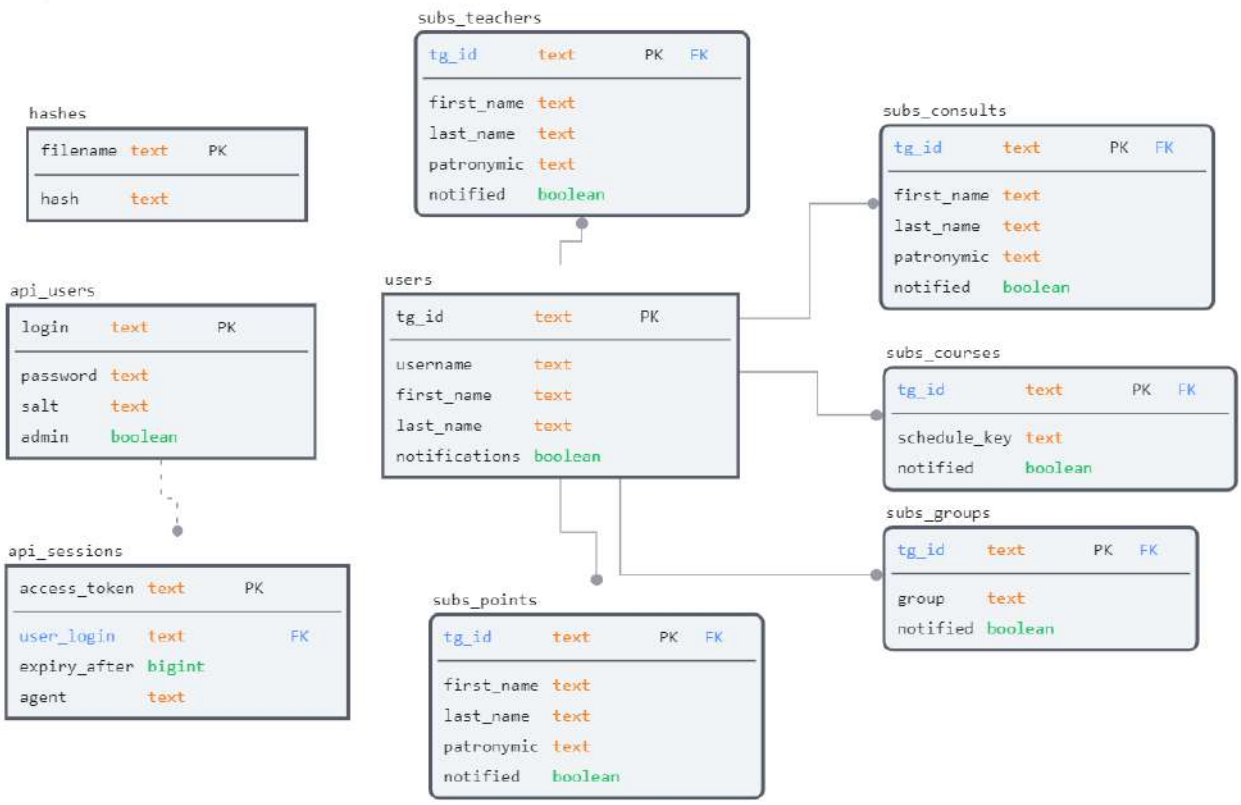


Рисунок 3.2 – Структура таблиць БД

Таблиця «users» представляє користувача чат боту. В ній зберігається ідентифікатор користувача месенджеру, деякі його персональні дані, та налаштування. Так як один користувач може бути підписаний лише один раз на певний тип розкладу, зв‘язок з іншими таблицями представлений як «один до одного».

Таблиця «subs_teachers» представляє підписку на розклад викладача. Вона має зовнішній ключ для зв‘язку з таблицею користувачів. Колонки «first_name» та «last_name» це не дублювання даних з таблиці користувачів, а дані викладача, на розклад якого оформлена підписка.

Таблиця «subs_courses» представляє підписку на розклад курсу.

Таблиця «subs_groups» представляє підписку на розклад певної групи.

Таблиця «subs_points» представляє підписку на власні оцінки. На відміну від інших підписок, надсилання інформації про оновлення оцінок не треба, тому поле «notified» відсутнє.

Таблиця «hashes» зберігає хеш суми файлів розкладу для порівняння їх з актуальними даними, для пошуку оновлених файлів. Хеш сума представлена у виді строки, тому і поле для її зберігання відповідного типу.

Таблиця «api_users» необхідна для зберігання даних про користувачів панелі керування. Для безпеки даних, існує 2 типи користувачів – звичайні та адміністратори. Кожен користувач може мати багато сесій в один час, тому має зв‘язок з таблицею «api_sessions» у виді «один до багатьох».

Таблиця «api_sessions» зберігає поточні сесії користувачів панелі керування. Кожна сесія має унікальний токен доступу, який є первинним ключом у таблиці. Тому знаючи токен, який надає користувач при кожному запиті, можна отримати інформацію про користувача та його права доступу. Поле «expiry_after» зберігає інформацію про час закінчення дії сесії. Це число, яке представляє час у форматі «Unix».

### 3.1.3 REST API

Панель керування складається з двох частин – бекенду та фронтенду. Бекенд для панелі проектується за принципами REST. REST API буде працювати поверх протоколу HTTP з використанням повідомлень у форматі JSON.

Кожен запит в HTTP здійснюється на певний адрес. Для розділення різних типів запитів використовуються так звані точки прийому запиту (endpoints). По суті це URL, на який здійснюється запит. Зазвичай для описування кінцевої точки, базовий URL ігнорується, бо він спільний для всіх точок. Для панелі керування можна виділити наступні точки прийому запиту:

- /login – здійснити аутентифікацію та отримати токен нової сесії.
- /logout – видалити власну сесію.
- /stats – отримати дані про статистику користування.
- /properties – отримати або змінити налаштування базові налаштування.
- /teachers отримати або змінити налаштування розкладу викладачів.

- /consult - отримати або змінити налаштування розкладу консультацій.
- /courses - отримати або змінити налаштування розкладу курсів.
- /rendering - отримати або змінити налаштування рендерингу.
- /user/list – отримати список акаунтів корситувачів панелі.
- /user/sessions – отримати список активних сесій.
- /user/endSession – завершити сесію.
- /user/create – створити нового користувача.
- /user/edit – змінити заді певного користувача.
- /user/delete – видалити користувача панелі.

Всі запити, крім аутентифікації, повинні мати токен доступу в заголовку. Цей токен необхідний для авторизації користувача. Для авторизації в заголовку повинен бути атрибут `Authorization: <acess_token>`, де «access_token» - це токен доступу, який можна отримати після запиту до точки «/login». Якщо сесія за даним токеном не знайдена, клієнт отримає відповідь зі статусом 401 Unauthorized.

На будь-який запит може прийти помилка замість відповіді. Повідомлення помилки має такий формат:

```
{
  "error": "<error type>",
  "message": "<error description>"
}
```

Де «error» - код помилки у строковому форматі, а «message» - більш докладний опис помилки. Для вказаних раніше точок можна виділити наступні види помилок:

- invalid_login – невірний логін при аутентифікації.
- invalid_password – невірний пароль при аутентифікації.
- unefined_message – неможливо прочитати повідомлення через нестачу даних.
- invalid_token – невірний токен при завершенні сесії.

- login_taken – користувач з вказаним логіном вже існує.
- password_short – пароль надто короткий.
- user_not_found – користувач з вказаним логіном не знайдений.

Повну специфікація REST API з описом та структурою всіх повідомлень можна знайти за посиланням [30] в списку посилань.

### 3.1.4 Модель взаємодії з ботом

В якості моделі для взаємодії з ботом була вибрана модель кінцевого автомату. Кожен кінцевий автомат може бути представлений як граф. Кожен стан, який може приймати бот під час «розмови» - це вершина графу, а команди, які посилає користувач, тобто переходи між станами – це ребра графу. На рисунку 3.3 зображений графік цих станів та переходів між ними.

Переходи, які починаються з символу «/» (слеш) - це команди. Ланцюг станів завжди починається з команди. До деяких станів можуть вести одразу декілька переходів, наприклад для виведення списку викладачів користувач може ввести одну з 4 команд, але від типу команди вирішується який тип розкладу вивести. Кожен стан супроводжується певною дією з боку бота. Часто це вивід інформації про поточний стан, або відповідь на певний запит.

Кожен кінцевий стан має перехід на початок (стан «Старт»). На графіку це не позначено щоб уникнути графічної плутанини з переходами.

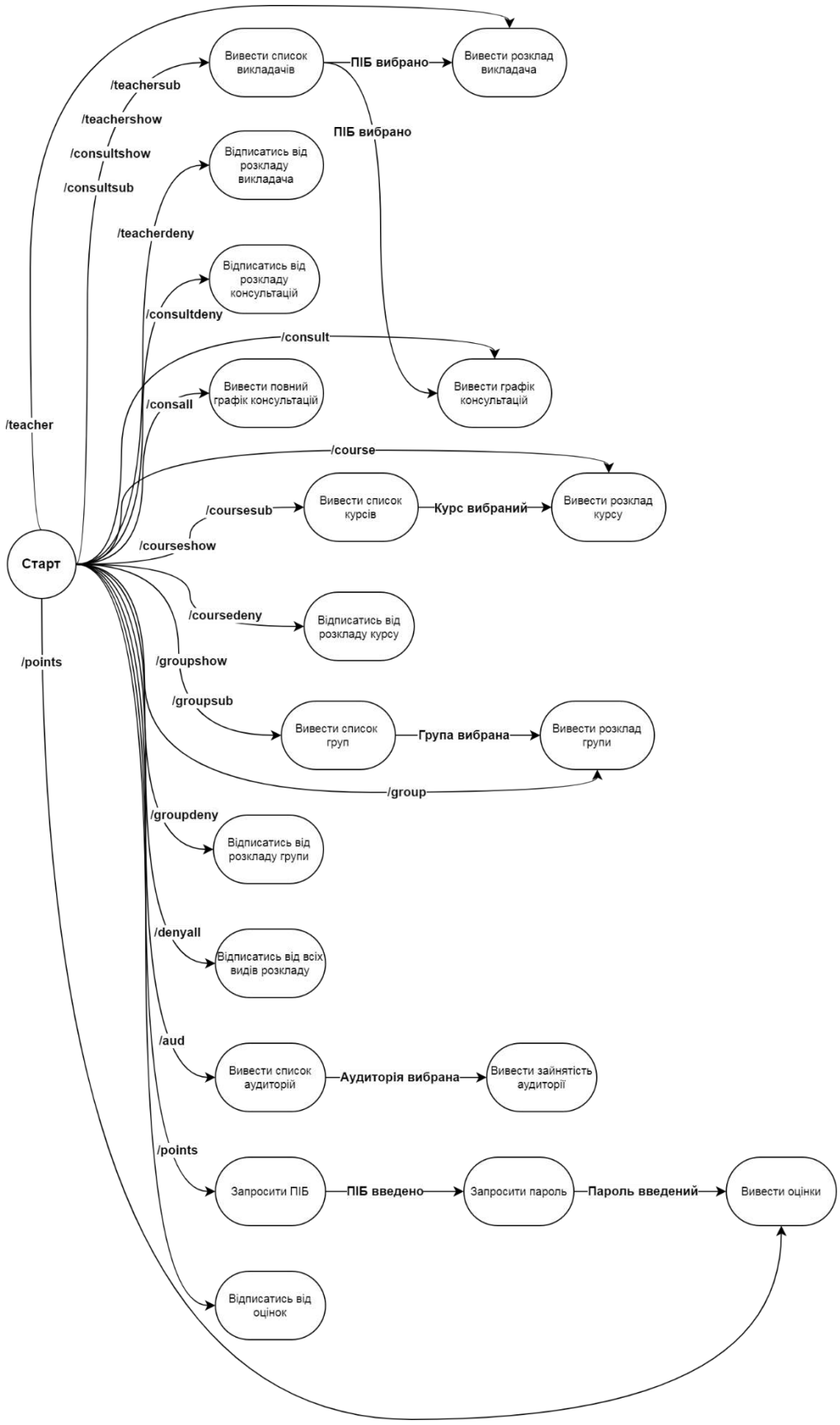


Рисунок 3.3 – Граф станів та переходів між ними

3.1.5 Структура проекту

Структуру проекту певною мірою диктують система збірки. Система збірки «Gradle», яка описана у 2 розділі, дозволяє розбивати проект на модулі. Всього можна виділити 3 модуля:

1. Модуль «api» зі спільними класами та утилітами.
2. Модуль «parser», представлення сутностей розкладу та його парсингу.
3. Головний модуль «bot», який має точку входу в програму, реалізацію інтерфейсів модуля «api», та логіку взаємодії з Telegram API.

Кожен модуль є одиницею компіляції, тому це також дозволить швидше збирати проект, завдяки інкрементальній збірці. Разом з необхідними для системи збірки файлами та директоріями, проект буде мати структуру, яка позначена на рисунку 3.4.

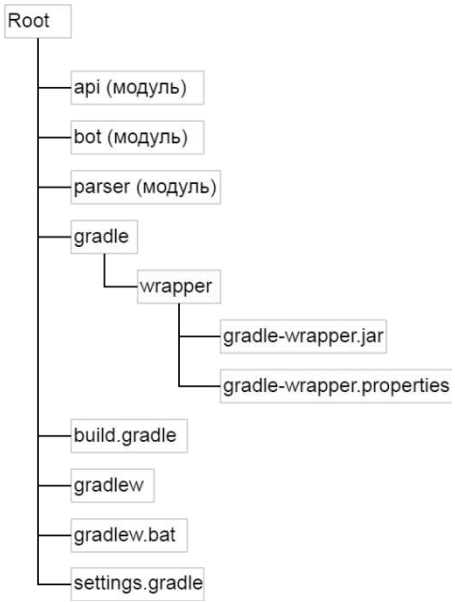


Рисунок 3.4 – Структура проекту

3.1.6 Проектування конфігурації програми

Конфігурація програми складається з 3 частин (файлів):

1. Головна конфігурація.

2. Конфігурація розкладу.
3. Конфігурація мови.

Головна конфігурація відповідає за базове налаштування програми, та містить наступні елементи:

1. Налаштування підключення до бази даних.
2. Налаштування підключення до Telegram API.
3. Налаштування регулярних виразів.
4. Налаштування підключення до серверу оцінок успіху.
5. Налаштування API панелі керування.

У лістингу 3.1 приведений приклад головної конфігурації бота.

Лістинг 3.1 – Головна конфігурація бота

```
database:
  'hibernate.dialect': "org.hibernate.dialect.H2Dialect"
  'hibernate.connection.driver_class': "org.h2.Driver"
  'hibernate.connection.url': "jdbc:h2:./database"
  'hibernate.connection.username': "user"
  'hibernate.connection.password': "user"
  'hibernate.show_sql': "false"
  'hibernate.hbm2ddl.auto': "update"
  'hibernate.jdbc.batch_size': "50"

telegram:
  bot_name: "VasyaSchedulerBot"
  token: "TOKEN"

regex:
  teacher_default: "([А-ЯЁЇІЄҐ][а-яёїієґ']{0,32})\\s*([А-ЯЁЇІЄҐ])\\.\\s*([А-
ЯЁЇІЄҐ])\\.*"
  teacher_inline: "([А-ЯЁЇІЄҐ][а-яёїієґ']{1,32}\\s*[А-ЯЁЇІЄҐ]\\.\\s*[А-
ЯЁЇІЄҐ]\\.*)\\s*(ауд\\..{1,3})"
  classroom: "[а-яёїієґ']*\\.*\\s*([0-9]{3})"

points:
  url: "http://77.93.42.90:81/uspeh/index.php"
  login_query:
"action=login&lastname=${lastname}&n1=${n1}&n2=${n2}&password=${password}&subm
it=%C2%EE%E9%F2%E8"
  login_success: 302
  timeout: 3

thread_pool_size: 6
```

Блок «database» налаштовує підключення до БД та поведінку фреймворку Hibernate, та є словарем де ключ це ім‘я властивості конфігурації Hibernate, а ключ – значення цієї властивості. Така конфігурація дозволяє дуже гнучко змінювати поведінку взаємодії з БД, або навіть тип цієї БД.

Блок «telegram» має поля для налаштування підключення до Telegram API.

Блок «regex» дозволяє налаштувати регулярні вирази для парсингу деяких частин розкладу. Ці вирази виділені в конфігурацію через можливу зміну або модифікацію цих частин у майбутньому.

Блок «points» має поля для налаштування підключення до серверу з результатами успіху студентів.

Поле «thread_pool_size» визначає кількість потоків для паралельної обробки запитів до бота від користувачів.

Конфігурація розкладу відповідає за налаштування мета інформації про різні типи розкладу та його рендеринг. Її структура приведена у лістингу 3.2.

Лістинг 3.2 – Конфігурація розкладу

```
check_rate: 20

day_indexes:
  "Понеділок": 0
  "Вівторок": 1
  "Середа": 2
  "Четвер": 3
  "П'ятниця": 4
  "Субота": 5
  "Неділя": 6

teachers:
  url: "https://.../Teachers.xls"
  associations:
    '1': "1"
    'К9': "1k"
    'Кол4': "4k"
    '1(з)': "zo:1 курс"

consult:
  url: "https://.../consultation.xls"
  day_point:
    col: 1
    row: 2
```

```
  teacher_point:
    col: 0
    row: 3

courses:
  - url: "https://.../1.xls"
    name: "Інститут 1 курс"
    day_point:
      col: 0
      row: 7
    group_point:
      col: 4
      row: 5

render:
  format: JPEG
  dpi: 175
```

Поле «check_rate» виставляє період перевірки розкладу у секундах.

Блок «day_indexes» налаштовує зіставлення імені дня до його індексу. Індекс першого дня є нулем.

Блок «teachers», «consult» та «courses» налаштовують доступ до розкладу відповідного типу. Блок «teachers» також має налаштування асоціацій скорочень, які можуть використовуватись у розкладі викладачів, з певним розкладом курсу.

Блок «render» налаштовує рендеринг розкладу, у саме формат вихідного зображення та його якість (кількість точок на дюйм).

Конфігурація мови має найбільш просту структуру. Це файл з великою кількістю рядків у форматі ключ-значення, де ключом є деяке унікальне ім‘я строки, а значенням сама строка. Доступ до цих строк у коді здійснюється саме через унікальне ім‘я. Це дозволить змінювати мову або окремі строки у програмі без її перекомпіляції.

## 3.2 Програмування

### 3.2.1 Зчитування конфігурації додатку

Для зручної роботи з конфігурацією програми, необхідно провести парсинг конфігураційних файлів у рідні для мови програмування структури. Для цього треба створити клас для кожного типу конфігурації. На рисунку 3.5 приведена структура класів конфігурації.

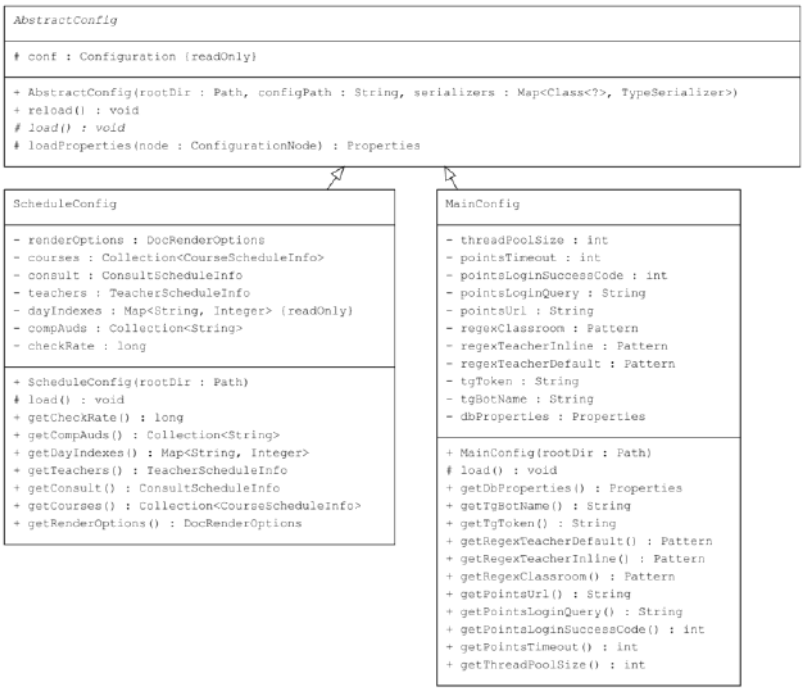


Рисунок 3.5 – Структура класів конфігурації

Клас AbstractConfig є базою для інших класів конфігурації. Він має посилання на дерево конфігурації, з якого інші класи можуть зчитувати або записувати дані. Також він має абстрактний метод load, який визивається під час кожного перезавантаження конфігурації. Для зручності роботи з деякими даними, він має метод «loadProperties», який перетворює вузел конфігурації на структуру и вигляді ключ-значення. У Java така структура називається Properties. Об‘єкт цього класу приймає наприклад фреймфорк Hibernate для внутрішнього налаштування, тому цей метод буде корисним у майбутньому. У лістингу 3.3 приведене тіло цього методу.

Лістинг 3.3 – Код методу loadProperties

```
protected Properties loadProperties(ConfigurationNode node) {
    Properties properties = new Properties();
    for (var entry : node.getChildrenMap().entrySet()) {
        String key = entry.getKey().toString();
        String value = entry.getValue().getString();
        properties.put(key, value);
    }
    return properties;
}
```

Клас MainConfig, як слідує з назви, представляє головну частину конфігурації програми. Він наслідується від AbstractConfig та реалізує його метод load. У тілі цього метода відбувається зчитування даних з конфігурації за їх шляхом (шляхом є послідовність ключів, яка веде до певного поля), та їх запис у відповідне поле об‘єкту. Надалі досуп до даних конфігурації здійснюється через ці поля, а точніше через getter-методи для цих полей.

Клас ScheduleConfig працює так само, як головний клас конфігурації, але зчитує інший файл. Слід зауважити, що цей клас працює з більш складними конструкціями, власними типами даних. Тому для кожного типу, який зчитується з конфігурації, заздалегідь створено так званий клас-десеріалізатор. Це клас, який відповідає за побудову більш складного об‘єкту з примітивів, які були зчитані з файлу конфігурації.

### 3.2.1 Програмування типів розкладу

Перш за все, необхідно створити систему ідентифікації розкладу. За цим ідентифіктором можна буде однозначно ідентифікувати таблицю розкладу. Кожна таблиця розкладу знаходиться у певному файлі та аркуші з унікальним ім‘ям. Тому для ідентифікації достатньо знати ім‘я файлу розкладу та ім‘я аркуша. Слід зауважити, що порядок та наявність деяких аркушів з розкладом не є гарантованою, тому порядковий номер аркуша не підійде для ідентифікації

розкладу. Тому ідентифікатор розкладу буде мати строковий тип, та записуватись у форматі <ім‘я файлу>:<ім‘я аркуша>, де ім‘я файлу є так званим простором імен, бо може мати більше ніж один ключ, а ім‘я аркушу є ключем доступу до розкладу. Також, важливо щоб і простір імен і ключ були завжди в одному регістрі, щоб запобігти помилок при їх порівнянні. Для зберігання такого ідентифікатора був створений клас NamespacedKey. Його структура приведена на рисунку 3.6.

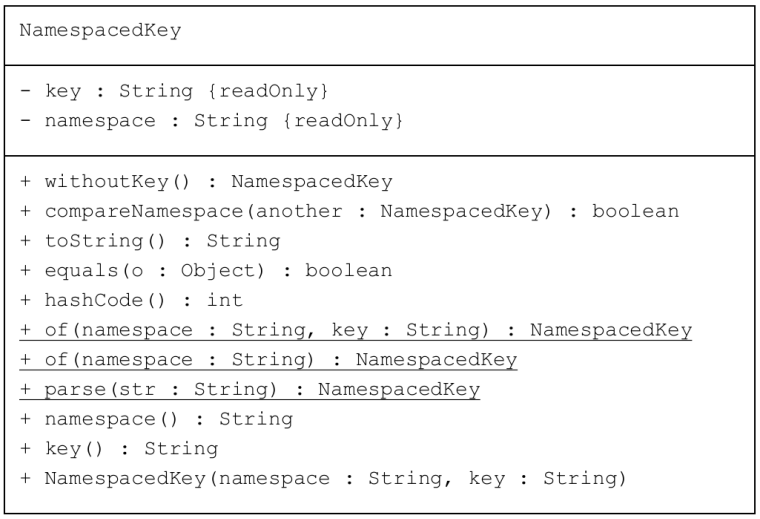


Рисунок 3.6 – Клас ідентифікатора розкладу

Перш за все клас перевизначає методи equals() та hashCode() для правильного порівняння об‘єктів цього класу при пошуку. Для перетворення об‘єкту класу на строку, клас перевизначає метод toString(). Його тіло приведено у лістингу 3.4.

Лістинг 3.4 – Метод toString()

```
public String toString() {
    return key.isEmpty() ? namespace : String.format("%s:%s", namespace, key);
}
```

Для створення об‘єкту класу зі строки, був створений статичний метод parse(), який приймає певну строку та повертає екземпляр ідентифікатора. Тіло метода приведено у лістингу 3.5.

Лістинг 3.5 – Метод parse()

```
public static NamespacedKey parse(String str) {
    Preconditions.checkNotNull(str, "Raw NamespaceKey cannot be null");
    String[] arr = str.split(":");
    if (arr.length == 1) return of(arr[0]);
    if (arr.length != 2) throw new IllegalArgumentException("Invalid
namespaced key: " + str);
    return of(arr[0], arr[1]);
}
```

Слідуючи об‘єктно орієнтованому підходу, відповідальність за збереження та роботу з інформацією про розклад буде розділена по класам. Інформацію про розклад можна розділити на два типи:

- Інформація, яка відома заздалегідь. Це метаінформація розкладу. До неї відносяться посилання на розклад, та позиції ключових точок.
- Інформація, яка отримується після парсингу. До цієї інформації відносяться такі дані, як список пар, викладачів, або інших даних, відповідних певному типу розкладу.

Відповідно до типів розкладу, можна виділити наступні класи метаінформації розкладу:

- TeacherScheduleInfo – метаінформація розкладу викладачів.
- CourseScheduleInfo - метаінформація розкладу курсу.
- ConsultScheuleInfo - метаінформація розкладу консультацій.

Для слідування принципам DRY, для цих класів був написаний базовий (абстрактний) клас. Структура цього класу позначена на рисунку 3.7.

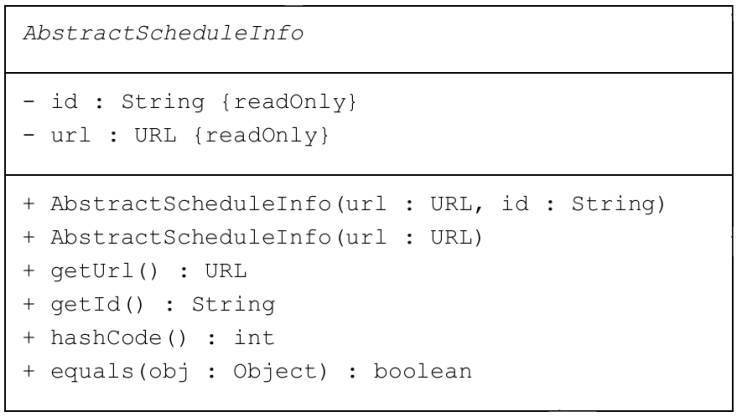


Рисунок 3.7 – Структура класу AbstarctScheduleInfo

Клас має два поля – «id» та «url». Ідентифікатор має бути унікальним для кожного розкладу. За замовченням ідентифікатором є ім‘я файлу розкладу. Структура всіх класів метаінформації про розклад мають структуру, яка позначена на рисунку 3.8.

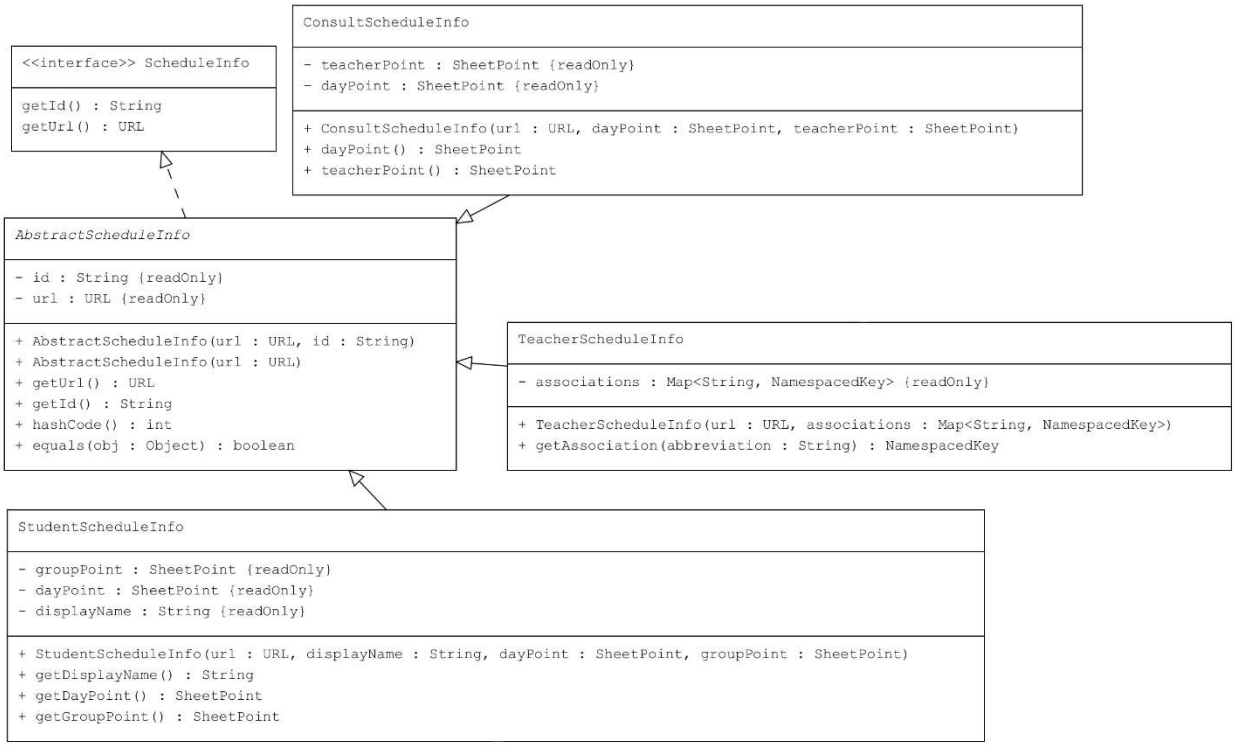


Рисунок 3.8 – Структура класів метаінформації

Клас TeacherScheduleInfo має поле «associations» типу Map, в якому зберігається інформація про відношення скорочення у таблиці зайнятості до певного ідентифікатору розкладу. За цим ідентифікатором можна буде знайти потрібний розпарсений розклад.

Клас CourseScheduleInfo має поля для ключових точок парсингу та ім‘я розкладу, яке буде відображатись користувачу.

Клас ConsultSchedule має поля для ключових точок парсингу файлу, без додаткових полей та методів.

Наступним кроком є створення класів самого розкладу. Відповідно до типів розкладу, можна виділити наступні класи:

- TeacherSchedule – розклад викладачів.
- CourseSchedule – розклад курсу.
- ConsultSchedule – розклад консультацій.
- ClassroomSchedule – розклад зайнятості аудиторії.

Ці класи зберігають відповідають за інформацію про розпарсений розклад. Як і класи метаінформації, вони наслідують від базового класу AbstractSchedule. Структура базового класу розкладу позначена на рисунку 3.9.

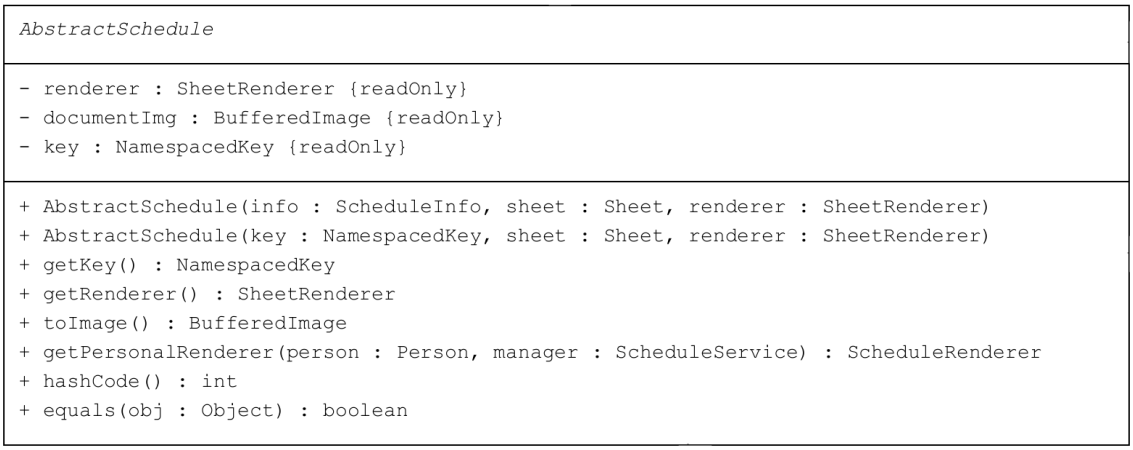


Рисунок 3.9 – Базовий клас розкладу

Клас має посилання на певний клас-рендер, зображення всього документу, яке формується після парсингу, та унікальний ідентифікатор розкладу. Для рендерингу певної частини розкладу (частіше всього персональний розклад користувача) створений абстрактний метод getPersonalRenderer(), який повертає екземпляр класу, який реалізує інтерфейс ScheduleRenderer. Стурктура класів для рендерингу описана у наступній частині підрозділу.

Всі раніше описані класи розкладу наслідують від базового класу. Тож їх структура виглядає так, як показано на рисунку 3.10. Детально кожен клас буде розглядатись далі.


Рисунок 3.10 – Структура класів розкладу

Клас TeacherSchedule представляє розклад викладачей (всіх разом). Він має структуру, позначену на рисунку 3.11. Клас зберігає інформацію про розклад кожного викладача у мапі (клас реалізуючий інтерфейс Map), де ключом є деякий клас Person, який буде описаний далі, а значенням є список днів викладача.

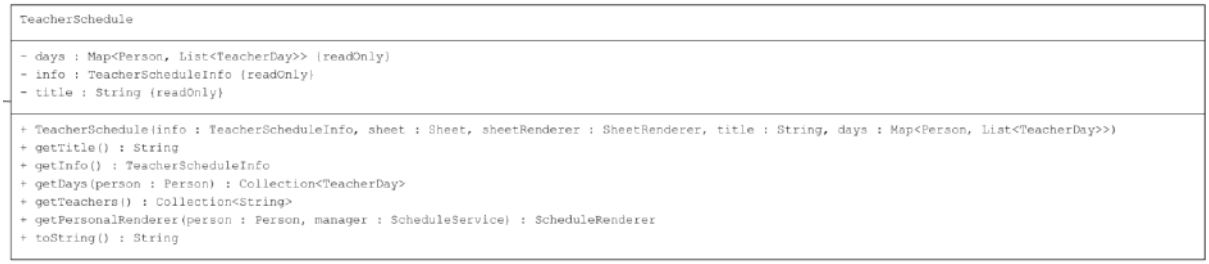


Рисунок 3.11 – Клас TeacherSchedule

Клас Person, який використовується у багатьох класах, в тому числі у розкладі викладачів, представляє певного учасника системи, людину. Його об‘єкт зберігає ім‘я, прізвище та по-батькові людини, та дозволяє безпечно порівнювати їх, в тому числі проводити приблизне порівняння. Структура цього класу позначена на рисунку 3.12.

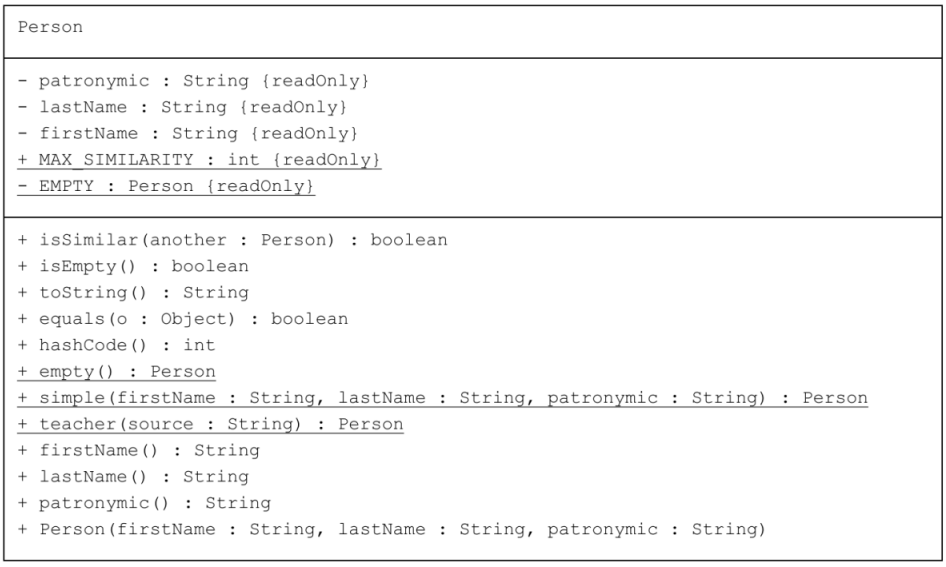


Рисунок 3.12 – Клас Person

Більше всього цікавий метод isSimilar(). Він приймає екземпляр цього ж класу, та повертає булево значення, яке говорить чи схожі порівняні персони. Степінь схожості розраховується за допомогою алгоритму «Відстань Левенштейну», використовуючи прізвище персони. Тіло цього методу можна побачити у лістингу 3.7.

Лістинг 3.7 – Метод isSimilar() класу Person

```
public boolean isSimilar(Person another) {
    if (isEmpty())
        return false;

    int ln = Levenshtein.calcDistance(this.lastName(), another.lastName());
    return ln <= MAX_SIMILARITY
            && this.firstName().equals(another.firstName())
            && this.patronymic().equals(another.patronymic());
}
```

Бачимо, що результатом порівняння строк за Левенштейном є деяке число, яке означає кількість модифікацій однієї строки, для повної відповідності другої. Максимальна кількість модифікацій позначена у константі MAX_SIMILARITY, та має значення 1 (один).

Порівняння персон за схожістю необхідно для пошуку викладачів, прізвища яких мають різне написання у різних таблицях розкладу.

### 3.2.1 Програмування парсеру таблиць

Для кожного типу розкладу необхідний власний клас з реалізацією парсингу цього розкладу. Відповідно до типів розкладу, можна виділити наступні класи парсерів:

- TeacherScheduleLoader – парсер розкладу викладачів.
- CourseScheduleLoader – парсер розкладу курсів.
- ConsultScheduleLoader – парсер розкладу консультацій.

Будь який парсер має спільну логіку, яку можна винести у абстрактний клас, від якого будуть наслідувати всі інші. Структура цього класу показана на рисунку 3.13.

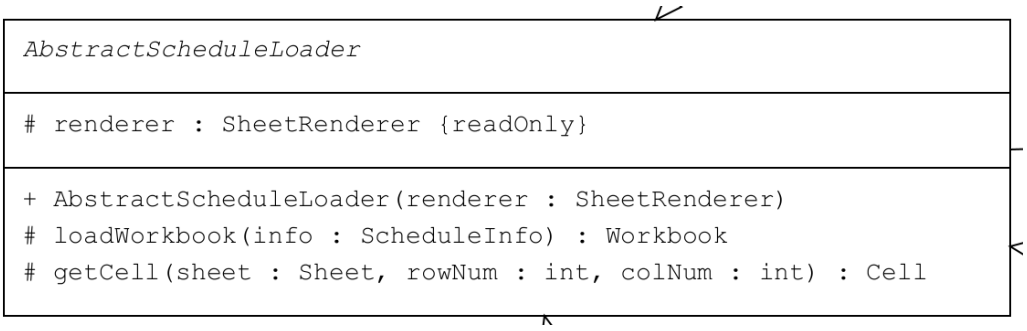


Рисунок 3.13 – Клас AbstractScheduleLoader

Метод loadWorkbook() завантажує документ Excel за посиланням, яке вказано у метаінформації розкладу. Він має тіло, яке приведено у лістингу 3.8.

Лістинг 3.8 – Метод loadWorkbook()

```
protected Workbook loadWorkbook(ScheduleInfo info) throws
ScheduleParseException {
    try (InputStream in = info.getUrl().openStream()) {
        String filename = info.getUrl().toString();
        Workbook workbook = null;

        if (filename.endsWith(".xlsx")) {
            workbook = new XSSFWorkbook(in);
        } else if (filename.endsWith(".xls")) {
            workbook = new HSSFWorkbook(in);
        }

        return workbook;
    } catch (IOException e) {
        throw new ScheduleParseException("Cannot load " +
info.getUrl().toString(), e);
    }
}
```

Так як бібліотека Apache POI не має механізму для виявлення формату документу, це робиться вручну, за допомогою перевірки розширення файлу.

Класи-парсери, які наслідують від базового, мають структуру, яка позначена на рисунку 3.14.

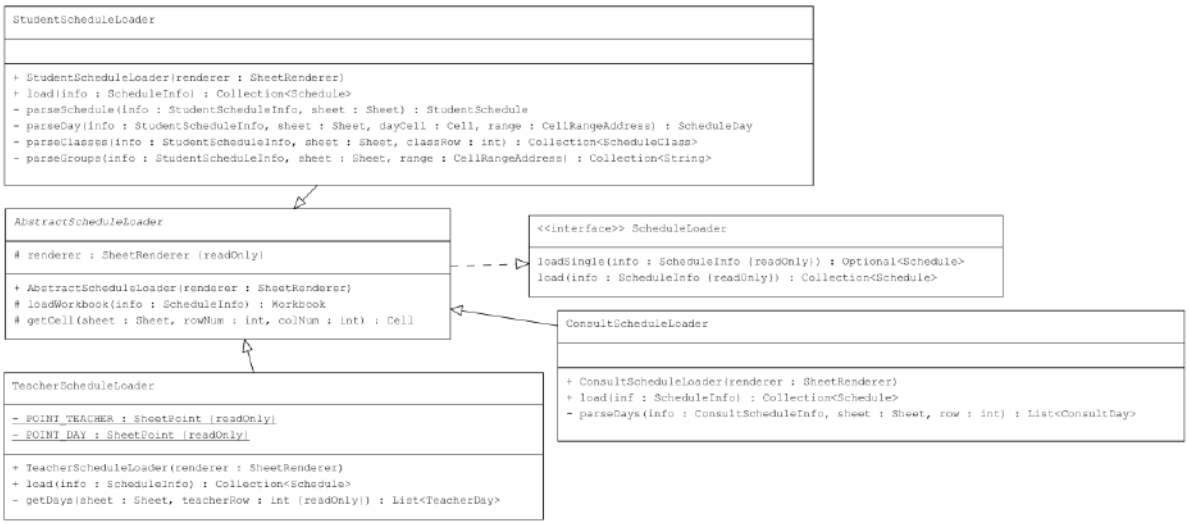


Рисунок 3.14 – Структура класів-парсерів

Парсинг розкладу виконується за допомогою певних вказівників на осередки. Парсинг розкладу викладачів починається з пошуку першого осередка з ім‘ям викладача та створенням вказівника нього. Пошук можливий завдяки координатам, які заздалегідь вписані у файл конфігурації програми. Кожен раз, коли вказівник на осередок викладача переміщується вниз, робиться ітерація по всім осередкам, які мають інформацію про пари. За це відповідає реалізація метода load(), який визначений у базовому класі парсера. Частина цього метода приведена у лістингу 3.9.

Лістинг 3.9 – Обхід таблиці викладачів

```
Map<Person, List<TeacherDay>> teachers = new HashMap<>();
int row = POINT_TEACHER.row();
Cell teacherCell = getCell(sheet, row, POINT_TEACHER.col());

while (!ExcelUtil.isEmptyCell(teacherCell)) {
    Person person = Person.teacher(ExcelUtil.getCellValue(teacherCell));
    List<TeacherDay> days = getDays(sheet, row);
    teachers.put(person, days);

    row++;
    teacherCell = getCell(sheet, row, POINT_TEACHER.col());
}
```

За обхід днів та пар розкладу викладача відповідає метод getDays(). Він має цикл, який інкрементує позицію вказівника на пару після запису даних попередньої пари.

### 3.2.3 Програмування рендеру розкладу

Рендеринг розкладу може проводитись різними методами. Для уніфікації цього процесу був створений абстрактний клас SheetRenderer. Він має методи, які повинен реалізувати кожен backend клас для рендерингу листа. В якості бекенду для рендерингу на даний час використовується бібліотека Aspose Cells, яка має відповідний функціонал. Для тестування та експериментів бус також створений власний клас для рендерингу за допомогою вбудованого в стандартну бібліотеку Java API для малювання AWT. Структура всіх класів рендерингу позначена на рисунку 3.15.

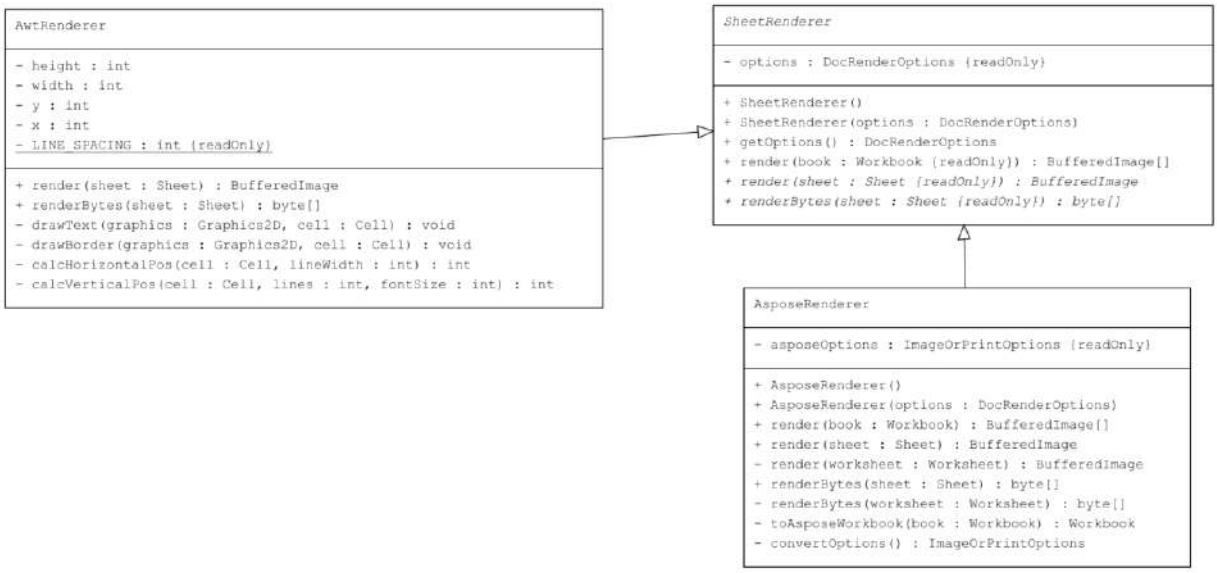


Рисунок 3.15 – Структура класів для рендерингу

За допомогою єдиного API рендерингу, всі класи для малювання таблицы розкладу можуть працювати з одним інтерфейсом не зважаючи на те, яким методом він буде малюватись.

Рендеринг розкладу починається з будування його моделі. Модель будується на основі отриманих за допомогою парсингу даних, які зберігаються у таких об‘єктах, класи яких були описані раніше, це TeacherSchedule, CourseSchedule та інші. Відповідно до класів цих об‘єктів можна виділити наступні класи для рендерингу:

- TeacherScheduleRenderer – відповідає за будівництво та рендеринг персонального розкладу викладача.
- CourseScheduleRenderer – відповідає за будівництво та рендеринг розкладу курсу.
- GroupScheduleRenderer – відповідає за будівництво та рендеринг розкладу групи.
- ConsultScheduleRenderer – відповідає за будівництво та рендеринг персонального розкладу консультацій.
- ClassroomScheduleRenderer – відповідає за будівництво та рендеринг розкладу зайнятості певної аудиторії.

Так як є загальна логіка та дані для всіх цих класів, можна виділити абстрактний клас. Його структура позначена на рисунку 3.16.

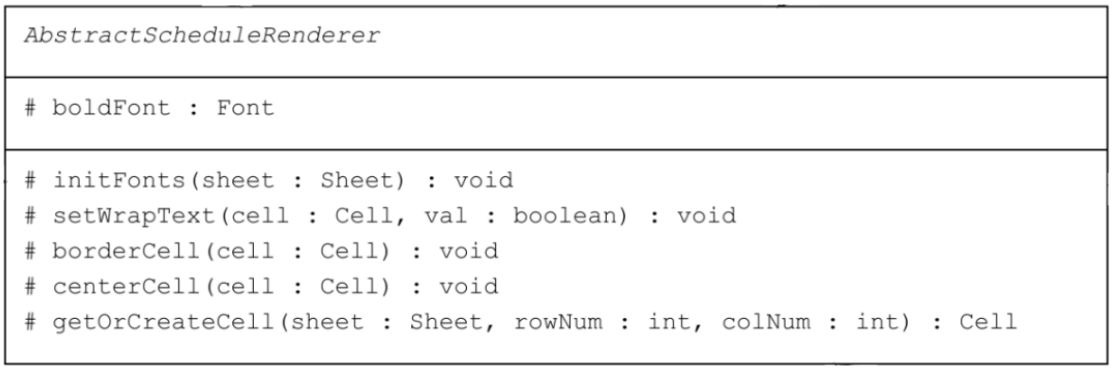


Рисунок 3.16 – Клас AbstractScheduleRenderer

Клас має методи для ініціалізації шрифтів, та допоміжні методи, які будуть часто використовуватись при рендерингу розкладу.

Рендеринг розкладу викладача починається зі створення шапки. Шапка має статичні та динамічні дані. За створення шапки відповідає метод drawHeader(). Якщо убрати код для оформлення, код заповнення осередків шапки даними приведений у лістингу 3.10.

Лістинг 3.10 – Створення шапки розкладу викладача

```
Cell titleCell = getOrCreateCell(sheet, 0, 0);
Cell nameCell = getOrCreateCell(sheet, 1, 0);
Cell dayTitleCell = getOrCreateCell(sheet, 2, 0);
Cell classNumTitleCell = getOrCreateCell(sheet, 2, 1);
Cell classTimeTitleCell = getOrCreateCell(sheet, 2, 2);
Cell classesTitleCell = getOrCreateCell(sheet, 2, 3);

titleCell.setCellValue(schedule.getTitle());
nameCell.setCellValue(person.toString());
dayTitleCell.setCellValue(lang.of("schedule.render.head.days"));
classNumTitleCell.setCellValue(lang.of("schedule.render.head.classnum"));
classTimeTitleCell.setCellValue(lang.of("schedule.render.head.classtime"));
classesTitleCell.setCellValue(lang.of("schedule.render.head.classes"));
```

Рендеринг тіла таблиці робиться двома методами – drawDay() і drawClass(). Метод drawDay() має цикл для ітерації по дням тижня певного викладача. Метод drawClass() відповідає за малювання осередку певної пари. Для цього на основі скорочення та асоціацій, які налаштовані для розкладу викладачів, робиться пошук по даним з розкладу курсу, на який є посилання у таблиці зайнятості. Для більш розумного пошуку є три допоміжні методи: getScheduleByCourses(), findClasses() та getAlternateClasses().

Метод getScheduleByCourses() приймає дані про пару з таблиці зайнятості, та повертає колекцію з розкладу курсів, на які посилається таблиця зайнятості. Його тіло приведено у лістингу 3.11.

Лістинг 3.11 – Метод getScheduleByCourses()

```
private Collection<CourseSchedule> getScheduleByCourses(TeacherClass
teacherClass) {
    List<CourseSchedule> schedules = new LinkedList<>();

    for (String course : teacherClass.courses()) {
        NamespacedKey scheduleKey =
this.schedule.getInfo().getAssociation(course);

        if (scheduleKey == null) // Try to find schedule by raw "pointer"
```

```
            scheduleKey =
this.schedule.getInfo().getAssociation(teacherClass.raw());

        if (scheduleKey != null) {
            CourseSchedule sch = (CourseSchedule)
manager.getCourseSchedule(scheduleKey);
            if (sch != null) schedules.add(sch);
        }
    }

    return schedules;
}
```

Пошук за «сирим» вказівником був добавлений через неоднозначність формату запису посилань на розклад курсів. Так, зазвичай, курси відділені комою, але іноді посилання може мати кому, тому стандартна операція split() над строкою може віддати невірний результат. Пошук за «сирим» посиланням може повернути правильний результат, при умові, що для цього посилання існує асоціація.

Метод findClasses() приймає екземпляр розкладу курсу, дані викладача та номер пари, та повертає всі пари, які призначені на запрошуваного викладача. Ці пари і виводяться у таблиці персонального розкладу, замість скорочень. Тіло методу приведене у лістингу 3.13.

Лістинг 3.12 – Метод findClasses()

```
private Collection<CourseClass> findClasses(
        CourseSchedule schedule,
        TeacherDay day,
        int classNum,
        Person teacher,
        boolean retAlternate
) {
    int dayIndex = TimeTable.getDayIndex(day.getName());
    Optional<CourseDay> dayOpt = schedule.getDay(dayIndex);

    if (dayOpt.isPresent()) {
        Collection<CourseClass> classes = dayOpt.get().getClasses(classNum,
person);

        if (!classes.isEmpty())
            return classes;
    }

    return retAlternate
            ? getAlternateClasses(schedule, day, classNum, teacher)
```

```
            : Collections.emptyList();
}
```

Якщо ні одна пара за посиланням не була знайдена, визивається метод getAlternateClasses(). Він приймає ті ж аргументи, що і метод findClasses(), але робить пошук за всіма таблицями розкладу у файлі. Деякі таблиці розкладу, наприклад розклад заочного відділення, знаходиться в одному файлі в різних листах. У разі відсутності хоч одної пари викладача, робиться припущення, що інформація про пару може бути у інших листах. Цей пошук є найбільш затратним, але і визивається він не часто. Тіло методу getAlternateClasses() приведене у лістингу 3.13.

Лістинг 3.13 – Метод getAlternateClasses()

```
private Collection<CourseClass> getAlternateClasses(
        CourseSchedule schedule,
        TeacherDay day,
        int classNum,
        Person teacher
) {
    for (CourseSchedule member : schedule.getFileGroup()) {
        Collection<CourseClass> classes = findClasses(member, day, classNum,
teacher, false);

        if (!classes.isEmpty())
            return classes;
    }

    return Collections.emptyList();
}
```

Наприкінці створення таблиці розкладу, якщо ні одна пара не була знайдена, на її місці виводиться посилання з таблиці зайнятості, для ручного пошуку викладачем. Після того, як модель розкладу була побудована, вона відправляється на фінальний рендеринг у растрове зображення.

Побудова та рендеринг розкладу інших категорій майже нічим не відрізняється від побудови розкладу викладача, за винятком відсутності системи пошуку пар.

### 3.2.4 Налаштування Guice DI

Для спрощення доступу до залежностей (об‘єктів) використовується техніка Dependency Injection за допомогою бібліотеки Guice, яка була описана у другому розділі.

Для організації джерел залежностей в Guice треба створити модулі. Кожен модуль має перевизначити метод configure(), в якому . Для поточного проекту було створено модулі BaseModule, PersistenceModule, ServicesModule, WebModule.

Модуль BaseModule визначає базові залежності, такі як конфігурація програми. Його код приведений у лістингу 3.14.

Лістинг 3.14 – Метод configure() модуля BaseModule

```
protected void configure() {
    bind(Path.class)
            .annotatedWith(Names.named("appDir"))
            .toInstance(rootDir);

    bind(MainConfig.class).toInstance(conf);
    bind(ScheduleConfig.class).toInstance(scheduleConf);
    bind(Language.class).toInstance(Language.builder()
            .source(ConfigSources.resource("/lang.yml", this)
                    .copyTo(rootDir))
            .build());
}
```

Модуль PersistenceModule визначає залежності, які відносяться до бази даних. До таких залежностей выдноситься фабрика сесій, Dao класи, тощо. Його код приведений у лістингу 3.15.

Лістинг 3.15 – Метод configure() модуля PersistenceModule

```
protected void configure() {
    SessionFactory sessionFactory = initHibernate();

    bind(SessionFactory.class).toInstance(sessionFactory);

    bind(TeacherSubsDao.class);
    bind(ConsultSubsDao.class);
    bind(CourseSubsDao.class);
    bind(GroupSubsDao.class);
    bind(PointsSubsDao.class);
```

```
    bind(NoticesDao.class);
    bind(HashesDao.class);
    bind(ApiUserDao.class);
    bind(ApiSessionDao.class);
}
```

Метод initHibernate() ініціалізує фабрику сесій Hibernate. Його тіло приведено у лістингу 3.16.

Лістинг 3.16 – Тіло методу initHibernate()

```
Configuration configuration = new Configuration();

configuration.addProperties(conf.getDbProperties());
configuration.addAnnotatedClass(SubscriptionPoints.class);
configuration.addAnnotatedClass(SubscriptionTeacher.class);
configuration.addAnnotatedClass(SubscriptionConsult.class);
configuration.addAnnotatedClass(SubscriptionCourse.class);
configuration.addAnnotatedClass(SubscriptionGroup.class);
configuration.addAnnotatedClass(ScheduleHash.class);
configuration.addAnnotatedClass(TeacherNotice.class);
configuration.addAnnotatedClass(ApiUser.class);
configuration.addAnnotatedClass(ApiSession.class);

StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder()
        .applySettings(configuration.getProperties());

SessionFactory sessionFactory =
configuration.buildSessionFactory(builder.build());
```

Визовом метода addAnnotatedClass() ми додаємо клас певної сутності БД у конфігурацію Hibernate. Більш детально про налаштування Hibernate мова йдеться у наступній частині цього підрозділу.

Модуль ServicesModule визначає залежності які відносяться до певних сервісів (менеджерів), таких як менеджер розкладу, менеджер часу та ін. Код налаштування залежностей у цьому модулі приведено у лістингу 3.17.

Лістинг 3.17 – Метод configure() модуля ServicesModule

```
protected void configure() {
    bind(SubsService.class).in(Scopes.SINGLETON);
    bind(ScheduleService.class)
            .to(ScheduleServiceImpl.class)
            .in(Scopes.SINGLETON);
    bind(PointsService.class).in(Scopes.SINGLETON);
    bind(SchedulerBot.class).in(Scopes.SINGLETON);
```

```
        bind(TimerService.class).in(Scopes.SINGLETON);
        bind(ApiUserService.class).in(Scopes.SINGLETON);
    }
```

Надалі, для доступу до цих залежностей достатньо лише вказати анотацію @Inject у конструкторі класу, в який треба запровадити залежність. Для впровадження залежності об‘єкт цього класу, його треба створити не через ключове слово «new», а використовуючи об‘єкт Injector, який надає бібліотека Guice. Запит залежностей через конструктор приведений на прикладі менеджеру підписок у лістингу 3.18.

Лістинг 3.18 – Запит залежностей через конструктор

```
public final class SubsService {

    private final TeacherSubsDao teacherDao;
    private final ConsultSubsDao consultDao;
    private final CourseSubsDao coursesDao;
    private final PointsSubsDao pointsDao;
    private final NoticesDao noticesDao;
    private final GroupSubsDao groupsDao;

    @Inject
    public SubsService(TeacherSubsDao teacherDao, ConsultSubsDao consultDao,
CourseSubsDao coursesDao, PointsSubsDao pointsDao, NoticesDao noticesDao,
GroupSubsDao groupsDao) {
        this.teacherDao = teacherDao;
        this.consultDao = consultDao;
        this.coursesDao = coursesDao;
        this.pointsDao = pointsDao;
        this.noticesDao = noticesDao;
        this.groupsDao = groupsDao;
    }
}
```

### 3.2.4 Програмування взаємодії з БД

Взаємодія з базою даних відбувається за допомогою фреймворку Hibernate, який описувався у другому розділі. Першим кроком впроваждення цього фреймворку буде створення сутностей, які представляють записи у таблицях бази даних. Відповідно до спроектованих раніше таблиць, можна виділити наступні класи сутностей:

- BotUser – представляє користувача системи.
- SubsTeacher – представляє підписку на розклад викладача.
- SubsConsult – представляє підписку на розклад консультацій.
- SubsCourse – представляє підписку на розклад курсу.
- SubsGroup – представляє підписку на розклад групи.
- SubsPoints – представляє запис даних для авторизації на сервері успішності.
- ScheduleHash – предсатвляє останню хеш суму файлу розкладу.
- ApiUser – представляє користувача панелі керування.
- ApiSession – представляє сесію користувача панелі керування.

Кожна сутність має бути відмічена спеціальними анотаціями, які визначають структуру майбутньої таблиці БД, зв‘язки з іншими сутностями, або навіть власні типи та дескриптори до них. Приклад класу, анотованого таким чином, видно на рисунку 3.17.

```java
@Entity
@Table(name = "users")
public class BotUser {

    @Id
    @Column(name = "tg_id")
    private String tgId;
    private String username;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;
    private boolean notifications;

    @OneToOne(mappedBy = "user", cascade = CascadeType.ALL)
    @PrimaryKeyJoinColumn
    private SubsTeacher subsTeacher;
```

Рисунок 3.17 – Анотований клас BotUser

Створені класи сутностей необхідно зареєструвати для обробки доданих аннотацій. Враховуючи, що в проекті використовується метод Dependency Injection, буде доцільно ініціалізувати Hibernate та його сутності у модулі PersistenceModule, який відповідає за реєстрацію залежностей, які відносяться до роботи за базою даних. Ініціалізація сутностей приведена у лістингу 3.19.

Лістинг 3.19 – Ініціалізація сутностей Hibernate

```
Configuration configuration = new Configuration();

configuration.addProperties(conf.getDbProperties());

configuration.addAnnotatedClass(SubsPoint.class);
configuration.addAnnotatedClass(SubsTeacher.class);
configuration.addAnnotatedClass(SubsConsult.class);
configuration.addAnnotatedClass(SubsCourse.class);
configuration.addAnnotatedClass(SubsGroup.class);
configuration.addAnnotatedClass(ScheduleHash.class);
configuration.addAnnotatedClass(BotUser.class);

StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder()
        .applySettings(configuration.getProperties());

SessionFactory sessionFactory =
configuration.buildSessionFactory(builder.build());
```

Для абстрагування управління сутностями було вирішено використовувати паттерн DAO. Він передбачає створення додаткових класів які інкапсулюють логіку роботи з БД (наприклад сирі SQL запити), та надають інтерфейс для роботи з БД простими методами, властивими для мови програмування.

Для кожного типу сутності треба створити DAO клас. Тоді можна виділити наступні класи:

- UserDao – робота з сутностями користувачів.
- SubsTeacherDao – робота з сутностями підписки на розклад викладачів.
- SubsConsultDao - робота з сутностями підписки на розклад консультацій.
- SubsCourseDao - робота з сутностями підписки на розклад курсів.

- SubsGroupDao - робота з сутностями підписки на розклад груп.
- HashesDao – робота з хеш сумами файлів розкладу.

Базовим класом для всіх DAO класів є абстрактний клас Dao, який має методи для створення та автоматичного закриття сесій (підключень). Його структура позначена на рисунку 3.17.

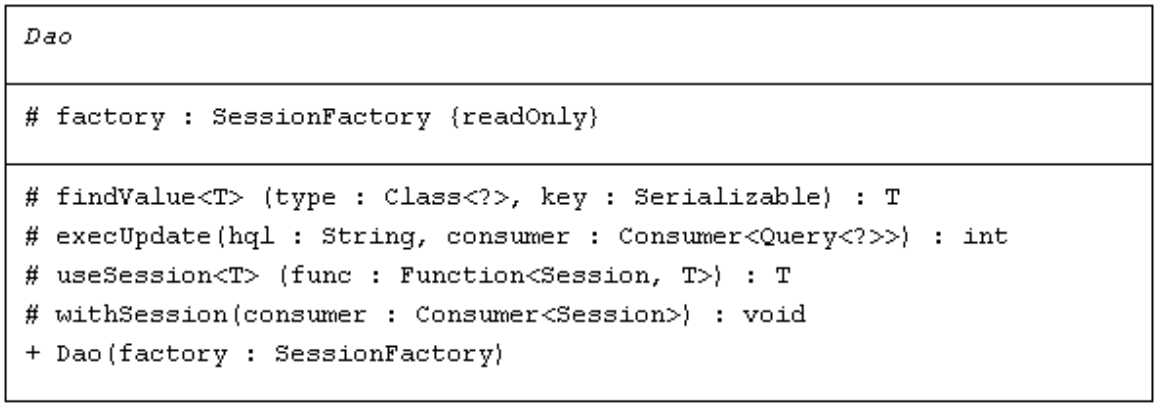


Рисунок 3.18 – Клас Dao

Так, наприклад, він має метод withSession() для створення сесії без повертання результату. Код цього метода приведений у лістингу 3.20. Важливим є те, що використовуючи try-with-resources, сесія автоматично закривається після виходу з try блоку.

Лістинг 3.20 – метод withSession()

```
protected void withSession(Consumer<Session> consumer) {
    try (Session session = factory.openSession()) {
        Transaction t = session.beginTransaction();
        consumer.accept(session);
        t.commit();
    }
}
```

Робота з базою даних через Hibernate та створений абстрактний клас, видна на прикладі класу SubsTeacherDao. Наприклад, для автоматичного сповіщення є задача знайти всіх користувачів, які не отримали сповіщення. Для

цього є метод findNotNotified(), який також приймає ліміт на кількість знайдених сутностей. Його код приведений у лістингу 3.21.

Лістинг 3.21 – Метод findNotNotified()

```
public Collection<SubsTeacher> findNotNotified(int limit) {
    return useSession(session -> {
        Query<?> q = session.createQuery("from SubsTeacher where notified =
false");
        q.setMaxResults(limit);
        return (Collection<SubsTeacher>) q.list();
    });
}
```

Як і інші подібні методи, він використовує HQL (Hibernate Query Language) замість SQL для можливості змінити тип бази даних на інший у будь який час. Одною з особливостей цієї мови запитів є використання назви сутності замість назви таблиці, тому вибірка йде з сутностей типу SubsTeacher. Інші DAO класи та їх методи майже не відрізняються один від одного та мають подібні до представлених CRUD методи.

Для використання створених класів в інших місцях, їх необхідно зареєструвати для впровадження через Guice DI. Для цього ми вже створювали PersistenceModule. Після реєстрації всіх конпонентів, метод configure() виглядає так, як позначено на рисунку 3.19.

```
@Override
protected void configure() {
    SessionFactory sessionFactory = initHibernate();

    bind(SessionFactory.class).toInstance(sessionFactory);

    bind(UserDao.class);
    bind(SubsTeacherDao.class);
    bind(SubsConsultDao.class);
    bind(SubsCourseDao.class);
    bind(SubsGroupDao.class);
    bind(SubsPointsDao.class);
    bind(HashesDao.class);
}
```

Рисунок 3.19 – Реєстрація компонентів в Guice

### 3.2.5 Програмування менеджеру підписок

Менеджер підписок надає доступ та можливість управління підписками користувачів на розклад. Він реалізований у класі SubsService. Конструктор та залежності цього класу вже описувались у підрозділі про впровадження залежностей. Клас залежить від DAO класів, які надають доступ до таблиць БД з інформацією про підписки. Структура цього класу позначена на рисунку 3.21.

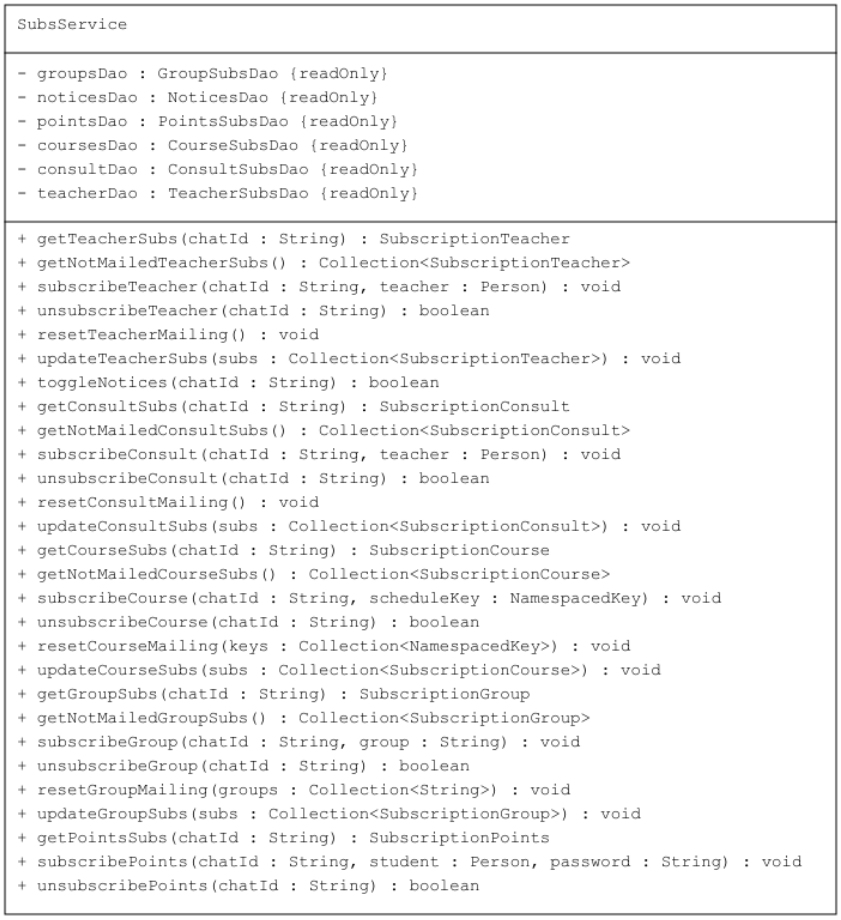


Рисунок 3.21 – Структура класу SubsService

Суть цього менеджеру в наданні інтерфейсу взаємодії з підписками без безпосередньої роботи з CRUD методами від DAO класів.

### 3.2.6 Програмування командної системи

Система «розмови» з ботом складається з двох частин – команд та запитів. Команда починається з символу «/» (слеш) та не має аргументів для простоти використання на мобільних пристроях. Запит може бути у виді вводу простого тексту або натискання на кнопку віртуальної клавіатури в чаті.

Відповідно до спроектованої моделі, а саме графу команд, для створення станів та переходів між ними, були написані класи State та ChatInput відповідно. Для зберігання сесії «розмови» створений клас ChatSession. Структура цих класів позначена на рисунку 3.22.

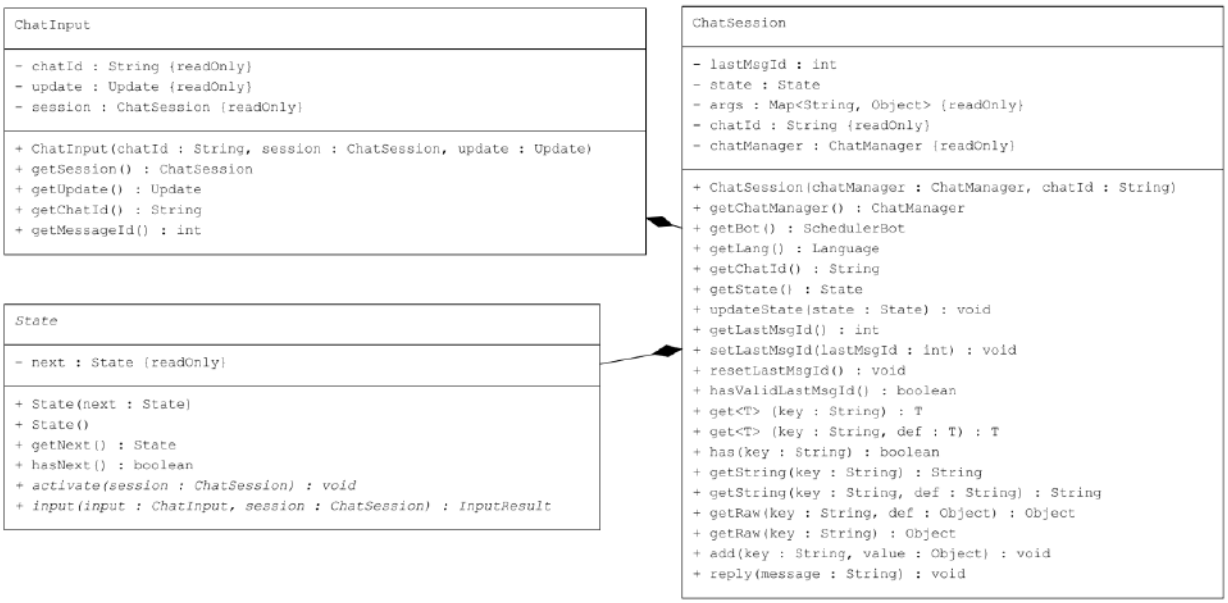


Рисунок 3.22 – Структура класів командної системи

Клас State є абстрактним, та представляє стан діалогу з ботом. Він має посилання на наступний стан та два головних метода – activate() та input().

Метод activate() визивається автоматично, коли стан діалогу оновлюється на до певного стану. Поведінка при активації визначається реалізацією цього метода у класі певного стану, але частіше всього цей метод виводить в чат інформацію про зміну стану, наприклад запит за ввід певної інформації.

Метод input() визивається, коли користувач робить ввід даних, поки знаходиться на поточному стані. Сигнатуру метода видно на рисунку 3.22, він

приймає об‘єкт класу ChatInput, та поточну сесію діалогу, яка має інформацію про користувача та діалог. Метод повертає результат вводу у виді одного з трьох варіантів перечислення: NEXT, WRONG та STAY. Від результату обчислення залежить подальша поведінка боту. При результаті NEXT, бот має перейти до наступного стану, якщо він існує, тоді у наступного стану буде визваний метод activate(). При результаті WRONG бот виведе помилку користувачу, вірогідно через неправильній ввід даних. При результаті STAY бот не буде робити ніяких дій.

Стан, який частіше всього використовується це список варіантів вибору. Його реалізація знаходиться у класі ChoiceState. Його структура позначена на рисунку 3.23.

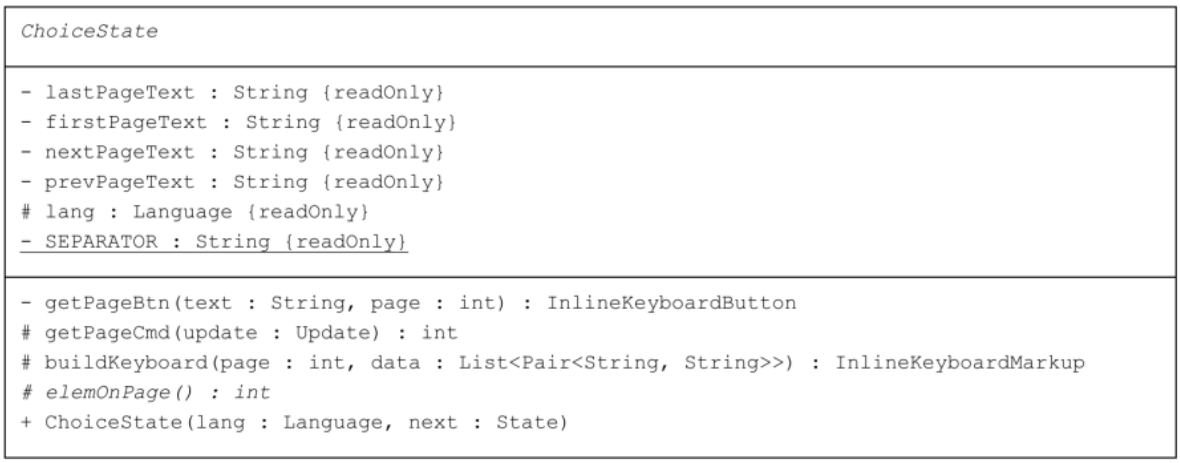


Рисунок 3.23 – Клас ChoiceState

Клас наслідує від State та додає декілька методів, які мають реалізувати класи, які будуть наслідувати від нього. Найважливішим методом є метод buildKeyboard() який приймає список пар типу строчка-строчка як аргумент та віддає екземпляр клавіатури з варіантами вибору, яку виведе бот. Першою строчкою у парі є назва кнопки, яка буде виведена, а другою – текстова команда, яка буде виконана при натисканні. Код генерування клавіатури вибору приведений у лістингу 3.22.

Лістинг 3.22 – Код генерування клавіатури

```
protected InlineKeyboardMarkup buildKeyboard(
      int page,
      List<Pair<String, String>> data
) {
    var builder = InlineKeyboardMarkup.builder();
    int pages = (int) Math.ceil((float) data.size() / elemOnPage());
    int from = Math.max(0, elemOnPage() * page);
    int to = Math.min(from + elemOnPage(), data.size());
    List<Pair<String, String>> pageData = data.subList(from, to);
    List<InlineKeyboardButton> row = new LinkedList<>();

    int i = 0;
    for (Pair<String, String> pair : pageData) {
        if (i % 2 == 0) {
            builder.keyboardRow(row);
            row = new LinkedList<>();
        }

        row.add(InlineKeyboardButton.builder()
                .text(pair.key())
                .callbackData(pair.value())
                .build());
        i++;
    }

    if (!row.isEmpty())
        builder.keyboardRow(row);

    if (data.size() > elemOnPage()) {
        if (page >= pages - 1) {
            // Last page
            builder.keyboardRow(Arrays.asList(
                    getPageBtn(firstPageText, 0),
                    getPageBtn(prevPageText, page - 1)
            ));
        } else if (page == 0) {
            // First page
            builder.keyboardRow(Arrays.asList(
                    getPageBtn(nextPageText, page + 1),
                    getPageBtn(lastPageText, pages - 1)
            ));
        } else {
            builder.keyboardRow(Arrays.asList(
                    getPageBtn(firstPageText, 0),
                    getPageBtn(prevPageText, page - 1),
                    getPageBtn(nextPageText, page + 1),
                    getPageBtn(lastPageText, pages - 1)
            ));
        }
    }

    return builder.build();
}
```

Важливою частиною тут є аргумент page, за допомогою якого можна розрахувати, які елементи треба вивести. Це зумовлено як обмеженням на кількість кнопок у клавіатурі, так і зручністю використання.

Всі стани діалогу зберігаються у структурі HashMap, де ключом є назва команди, а значенням стартовий стан. Ці данні зберігає об‘єкт класу ChatManager. Саме цей клас відповідає за управління діалогами. Його структура позначена на рисунку 3.24.

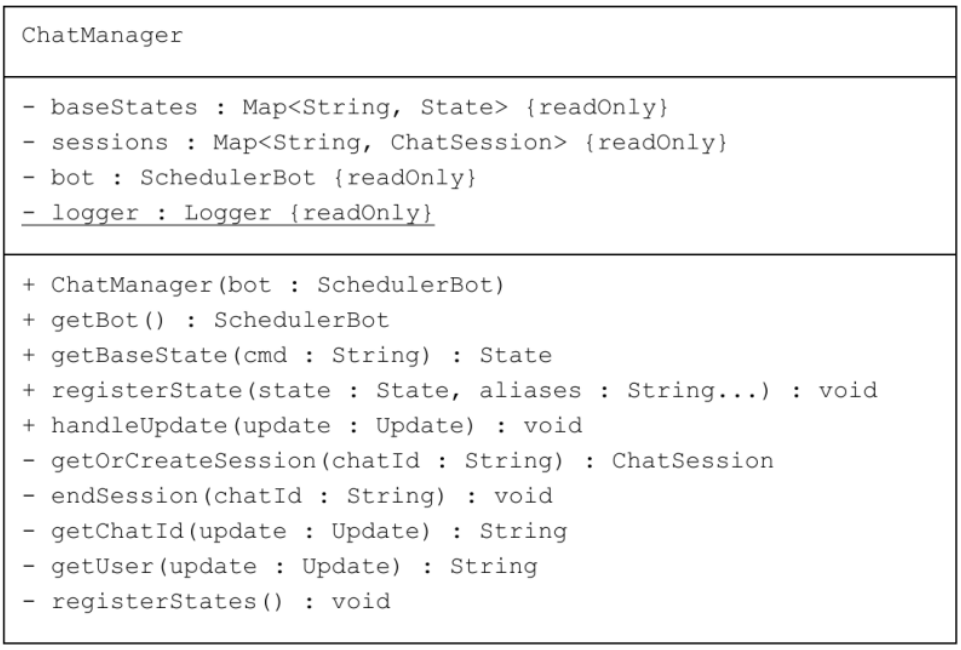


Рисунок 3.24 – Клас ChatManager

Найбільш важливим тут є метод handleUpdate(), в який передаються дані про запит до бота. У лістингу 3.23 приводиться частина коду цього метода, яка управляє ходом діалогу за результатом обробки запиту.

Лістинг 3.23 – Управління діалогом за результатом запиту

```
State state = session.getState();

if (state != null) {
    InputResult result = state.input(input, session);

    if (result.equals(InputResult.STAY))
```

```
            return;

        if (result.equals(InputResult.NEXT)) {
            session.updateState(state.getNext());
            return;
        }

        if (!state.hasNext())
            endSession(chatId);
    }

    bot.sendMessage(session, bot.getLang().of("cmd.unsupported"));
```

### 3.2.7 Програмування таймеру розкладу

Таймер відповідає за перевірку оновлення розкладу та синхронізацію масової розсилку цього розкладу. Для цього був створений клас TimerService. Його структура позначена на рисунку 3.25.

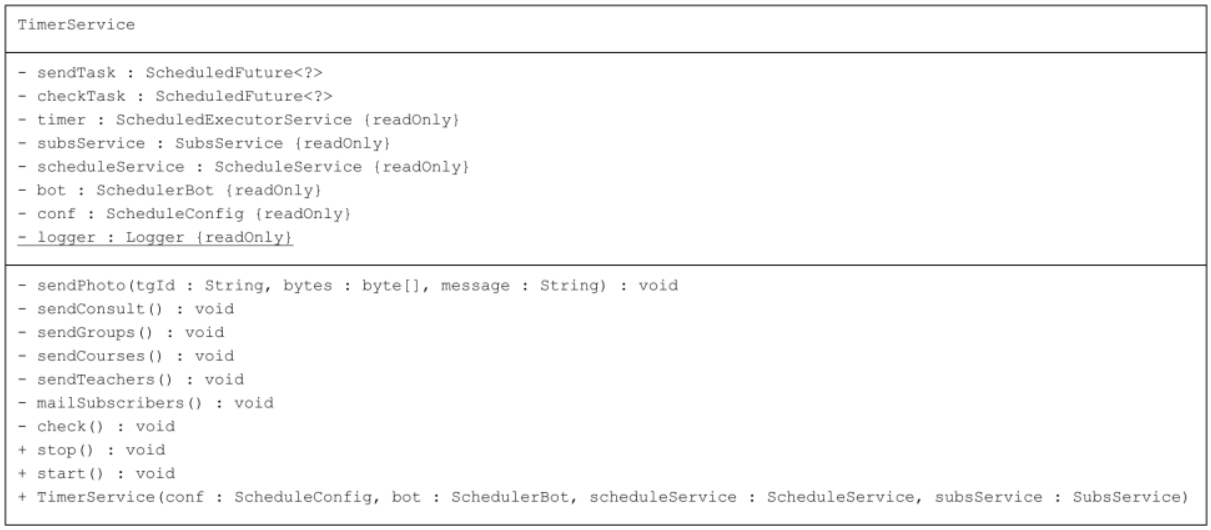


Рисунок 3.25 – Клас TimerService

Клас містить 2 окремих таймери – таймер перевірки оновлень та таймер розсилки розкладу. Їх ініціалізація відбувається у методі start(), який приведений у лістингу 3.24.

Лістинг 3.24 – Метод start() класу TimerService

```
public void start() {
    checkTask = timer.scheduleWithFixedDelay(this::check, 0L,
```

```
                conf.getCheckRate(), TimeUnit.SECONDS);

        sendTask = timer.scheduleWithFixedDelay(this::mailSubscribers, 0L,
                2L, TimeUnit.SECONDS);
    }
```

Перший таймер, з періодом зазначеним у конфігурації, визиває метод check() для перевірки оновлень розкладу. Метод за допомогою менеджеру розкладу робить запит на оновлення розкладу. Якщо розклад був оновлений, для кожного користувача, який був підписаний на цей розклад, видаляється флаг про закінчення сповіщення. Приклад коду який це робить приведений у лістингу 3.25.

Лістинг 3.25 – Перевірка оновлення розкладу викладачів

```
if (scheduleService.reloadTeacherSchedule(false)) {
    subsService.resetTeacherMailing();
    logger.info("Teacher schedule reloaded. Marked everyone for mailing");
}
```

Після того як флаг оповіщення був знятий, таймер розсилки при наступній перевірці не сповіщених користувачів, відправить їм оновлений розклад, бо він був оновлений ще при першій перевірці. Так, раз в 2 секунди визивається метод mailSubscribers(), який послідовно оповіщає всіх хто підписався на відповідний розклад. Приклад сповіщення приведений у виді коду сповіщення викладачів у лістингу 3.26.

Лістинг 3.26 – Сповіщення викладачів

```
Collection<SubscriptionTeacher> notMailed =
subsService.getNotMailedTeacherSubs();

if (!notMailed.isEmpty()) {
    for (SubscriptionTeacher sub : notMailed) {
        ScheduleRenderer renderer = scheduleService.getTeacherSchedule()
                .getPersonalRenderer(sub.getTeacher(), scheduleService);

        sendPhoto(sub.getTelegramId(), renderer.renderBytes(),
                bot.getLang().of("mailing.teacher"));

        sub.setReceivedMailing(true);
    }
```

```
    subsService.updateTeacherSubs(notMailed);

    logger.info("Sent " + notMailed.size() + " messages during teachers
mailing");
}
```

Як видно, першим кроком є знаходження всіх підписаних користувачів зі знятою міткою про завершене сповіщення. Якщо такі користувачі знайдені, для кожного рендериться розклад, та відсилається через чат-бот. тільки на прикінці розсилки всі сповіщені користувачі відмічаються як сповіщені одним запитом до БД. Така систем розсилки робить її більш гарантованою, у випадку якщо під час розсилки трапиться критичний сбій, тому що невідмічені про успішне сповіщення користувачі все одно будуть сповіщені при наступній роботі таймеру.

### 3.2.8 Програмування головного класу бота

Головним класом боту є SchedulerBot, який наслідує від бібліотечного класу TelegramLongPollingBot. Він відповідає за взаємодію з Telegram API за допомогою бібліотеки TelegramBots. Його структура позначена на рисунку 3.26.

Клас має методи для відправки повідомлень різного типу, та слухає запити від користувачів. Для прослухування запитів користувачів, треба перевизначити метод onUpdateReceived(). Його код приведений у лістингу 3.27

Лістинг 3.27 – Прослухування запитів

```
public void onUpdateReceived(Update update) {
    CompletableFuture.runAsync(() ->
            chatManager.handleUpdate(update), threadPool);
}
```

Для ефективності обробки запитів, був створений пул потоків. Кількість потоків у ньому налаштовується у конфігурації програми. Таким чином, кожен запит виконується не блокуючи головний поток.

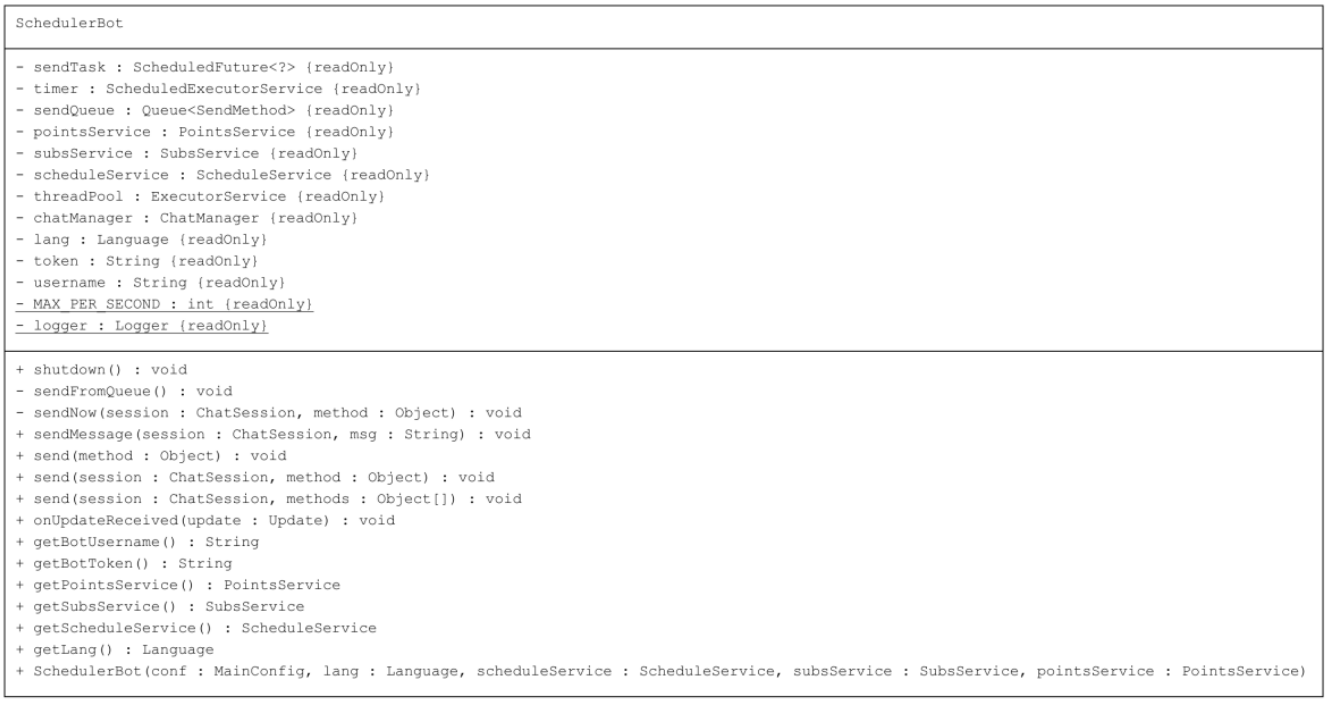


Рисунок 3.26 – Клас SchedulerBot

Також клас здійснює управління відправкою повідомлень, через обмеження Telegram у 30 повідомлень в секунду. Для цього була створена черга повідомлень (поле sendQueue, позначене у структурі класу), та таймер, який кожну секунду забирає до 30 повідомлень з черги та відправляє їх вже безпосередньо до користувачів. За це відповідає метод sendFromQueue(), який і визивається таймером. Його код приведений у лістингу 3.28.

Лістинг 3.28 – Метод sendFromQueue()

```
private void sendFromQueue() {
    if (!exe.isShutdown()) {
        for (int i = 0; i < MAX_PER_SECOND; i++) {
            SendMethod method = sendQueue.poll();

            if (method != null)
                sendNow(method.session(), method.method());
        }
    }
}
```

В свою чергу, метод send(), який визивається у багатьох містах, не відправляє повідомлення одразу, а лише додає його до черги, використовуючи стандартний метод класу Queue - offer(), який додає елемент у кінець черги. Код цього метода приведений у лістингу 3.29.

Лістинг 3.29 – Метод send()

```
public void send(ChatSession session, Object method) {
    if (method != null)
        sendQueue.offer(new SendMethod(session, method));
}
```

## 3.3 Розгортання

Процес розгортання будь якого серверу починається з перевірки апаратного та програмного комплексу. Для встановлення чат боту, інститут надав доступ до серверу ProLiant DL 380 G4, який підтримує до двох процесорів Intel Xeon з технологією EM 64 T, що розширюється до 12 Гб пам'ять PC 2-3200 R DDR 2 з частотою 400 МГц, три повнорозмірні слоти розширення, гігабітний мережевий контролер і вбудований контролер SmartArray 6 і RAID. Крім того, цей сервер оснащений оригінальним дисковим кошиком, який забезпечує роботу жорстких дисків SCSI на одному або двох каналах.

На час розгортання, на даному сервері були встановлені такі апаратні елементи, як:

- Процесор Intel Xeon 3.3 ГГц
- RAM об‘ємом 3 ГБ

В якості операційної системи, на сервері встановлено Ubuntu Server версії 16.04. Для управління сервером віддалено використовується протокол SSH. Сервер, який був наданий інститутом, вже має налаштований SSH сервер. Також, для передачі файлів необхідний доступ до файлової системи через протокол FTP, який також встановлений та налаштований на даному інститутом сервері.

Діаграма розгортання позначена на рисунку 3.27. Головним вузлом є «Application Server», який має три модуля – середовище виконання, виконуваний файл, та локальну базу даних. Він автоматично підключається до зовнішніх вузлів по протоколу HTTPS, тому головною задачею є встановлення вузлу «Application Server».

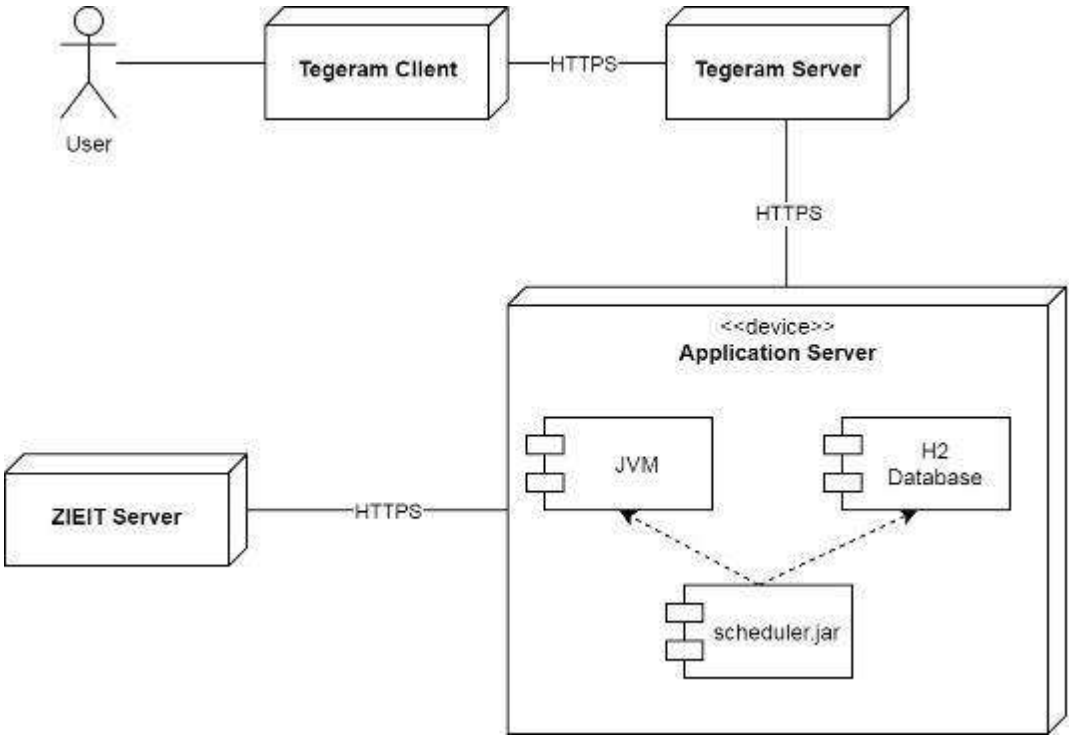


Рисунок 3.27 – Діаграма розгортання серверу

### 3.3.1 Встановлення середовища виконання програми

Персональний помічник являє собою програму, написану на мові Java, тому для її розгортання необхідне середовище виконування, яке називається JVM (Java Virtual Machine). Ubuntu Server, яка встановлена на фізичному сервері інституту, використовує пакетний менеджер APT (Advanced Packaging Tool). Тому вся перевірка залежностей будет проходити автоматично.

Першим кроком при встановленні пакету є оновлення списку репозиторіїв. Це робиться командою `apt update`, як показано на рисунку 3.28 Для багатьох операції треба використовувати права адміністратора.

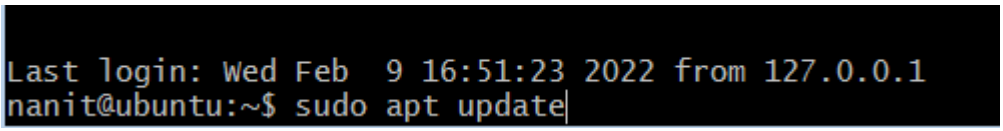


Рисунок 3.28 – Оновлення списку репозиторіїв

Після оновлення необхідно знайти ім‘я пакету останньої версії JVM. На момент розгортання остання версія – 17. Для цього можна використати команду пошуку, яка позначена на рисунку 3.29 Ця команда виводить знайдені пакети за ім‘ям, введений у команду.


Рисунок 3.29 – Пошук потрібного пакету

Так як для роботи чат боту необхідна лише JVM, доцільно вибрати пакет з мінімальним набором компонентів. Згідно стандарту, пакети з приставкою «jre» (Java Runtime Environment) містять лише JVM та необхідні для її роботи бібліотеки, без компілятору та інших інструментів. Тому для встановлення будемо використовувати пакет openjdk-17-jre. На рисунку 3.30 приведена команда, яка використовується для встановлення нового пакету.

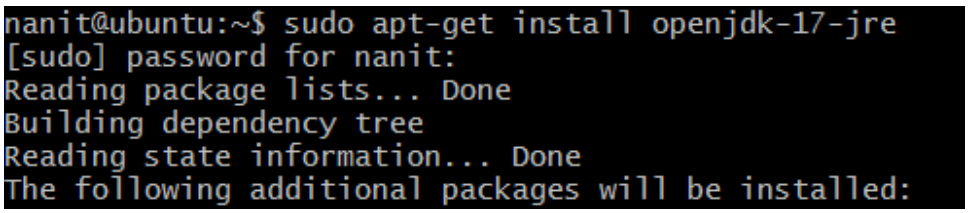


Рисунок 3.30 – Встановлення пакету

Після погодження зі встановленням, та деякого часу, пакет та всі необхідні залежності будуть встановлені на сервері. Для перевірки наявності віртуальної машини у системі, та правильно налагоджений змінних середовища, можна ввести команду для перевірки версії віртуальної машини, яка приведена на рисунку 3.31.

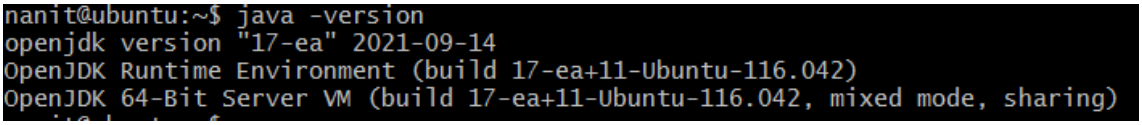


Рисунок 3.31 – Перевірка наявності віртуальної машини у системі

Після того, як версія встановленої машини виводиться, можна вважати, що середовище для розгортання чат боту успішно налагоджене. Наступний крок – встановлення чат боту.

### 3.3.2 Встановлення чат боту

Для встановлення чат боту використовується заздалегідь скомпільований .jar файл, який є головним виконуваним файлом. Для переміщення файлів між сервером та локальною машиною використовується протокол FTP. Для цього можна використати будь який FTP клієнт.

Для роботи чат боту необхідно налаштувати базу даних. Було вирішено використовувати локальну легку базу даних H2. Для того, щоб мати доступ до неї, боту необхідний JDBC драйвер, який заздалегідь був завантажений з офіційного сайту програми. Тож для розгортання необхідно скопіювати на сервер два файли:

- Головний виконуваний файл scheduler.jar
- h2-2.1.100.jar – JDBC драйвер для управління БД

Після того, як файли скопійовані на сервер, необхідно провести перший запуск програми для того, щоб вона створила файли конфігурації. Для цього

можна ввести коротку команду для запуску Java програми, яка приведена на рисунку 3.32.

```
nanit@ubuntu:~/scheduler$ java -jar scheduler.jar
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impac
t performance.
[20:27:39] INFO: HHH000412: Hibernate Core {[WORKING]}
[20:27:39] INFO: HHH000206: hibernate.properties not found
[20:27:40] INFO: HCANN000001: Hibernate Commons Annotations {5.0.4.Final}
[20:27:40] INFO: HHH000130: Instantiating explicit connection provider: org.hibe
rnate.hikaricp.internal.HikariCPConnectionProvider
[20:27:40] INFO: HikariPool-1 - Starting...
[20:27:41] INFO: HikariPool-1 - Start completed.
```

Рисунок 3.32 – Перший старт програми

Після першого запуску були створені всі необхідні файли. Перший раз бот не може підключитись до БД та API Telegram, бо вони ще не налаштовані. Для початку налаштування основних параметрів треба відкрити файл config.yml. Для редагування файлів на сервері можна використовувати утиліту «nano».

Для налаштування підключення до бази даних, необхідно змінити властивості у блоці «database», як показано на рисунку 3.33. Всі можливі властивості та їх значення можна знайти у документації до фреймворку Hibernate.

```
# Настройка подключения к БД через Hibernate
database:
  'hibernate.dialect': "org.hibernate.dialect.H2Dialect"
  'hibernate.connection.driver_class': "org.h2.Driver"
  'hibernate.connection.url': "jdbc:h2:./database;mode=MySQL"
  'hibernate.connection.username': "user"
  'hibernate.connection.password': "user"
  'hibernate.show_sql': "false"
  'hibernate.hbm2ddl.auto': "update"
  'hibernate.jdbc.batch_size': "50"
```

Рисунок 3.33 – Налаштування доступу до БД

Для отримання списку запитів до боту, необхідно вписати його токен та ім‘я у відповідні поля блоку «telegram», як показано на рисунку 3.34.

```
# Настройка подключения к Telegram боту
telegram:
  bot_name: "VasyaTheBot"
  token: "1091413821:AAEvSrjYdVPqV6_Sfe7eC1nOpCF6_wQxjgs"
```

Рисунок 3.34 – Налаштування токену чат бота

Після цього можна ще раз запустити бота та перевірити що він провантажується до кінця, як показано на рисунку 3.35.

```
nanit@ubuntu:~/scheduler$ java -jar scheduler.jar
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.
[20:54:32] INFO: HHH000412: Hibernate Core {[WORKING]}
[20:54:32] INFO: HHH000206: hibernate.properties not found
[20:54:33] INFO: HCANN000001: Hibernate Commons Annotations {5.0.4.Final}
[20:54:33] INFO: HikariPool-1 - Starting...
[20:54:33] WARN: Registered driver with driverClassName=org.h2.Driver was not found, trying direct
antiation.
[20:54:33] INFO: HikariPool-1 - Start completed.
[20:54:33] INFO: HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
[20:54:35] INFO: Loading schedule ...
[20:54:37] INFO: Loaded schedule '1k'
[20:54:37] INFO: Loaded schedule '2k'
[20:54:38] INFO: Building classroom schedule ...
[20:54:38] INFO: Classroom schedule has been built
[20:54:38] INFO: Loaded teachers schedule
[20:54:38] INFO: Loaded consultations schedule
[20:54:38] INFO: All schedule loaded
[20:54:38] INFO: Starting long polling bot ...
[20:54:39] INFO: Started timer
[20:54:39] INFO: Done! Scheduler ready to work
[20:54:39] INFO: HHH000397: Using ASTQueryTranslatorFactory
```

Рисунок 3.35 – Другий запуск бота

Для стабільної та безперебійної роботи бота, треба налагодити його автоматичний перезапуск при аварійному виході програми або перезавантажені операційної системи. Для цього використовується стандартний для Ubuntu та багатьох інших дистрибутивів Linux, підсистема ініціалізації та управління службами systemd. Для того, щоб створити нову службу, треба написати скрипт для її запуску у директорії /etc/systemd/system. Кожен файл цієї директорії з розширенням .service є окремою службою. Для створення служби, яка буде контролювати життєвий цикл чат боту, створемо файл який буде називатись scheduler.service. Далі додамо інструкції, які позначені на рисунку 3.36.

```
[Unit]
Description=ZIEIT scheduler bot
[Service]
User=nanit
WorkingDirectory=/home/nanit/scheduler

ExecStart=/home/nanit/scheduler/start.sh

SuccessExitStatus=0
TimeoutStopSec=5
Restart=on-failure
RestartSec=5
```

Рисунок 3.36 – Скрипт для запуску сервісу

Параметр ExecStart це шлях до скрипту запуску Java програми start.sh. Він містить строку, яка запускає програму:

```
java -jar scheduler.jar
```

Для старту загрузки сервісу у підсистему system, треба ввести наступну команду:

```
sudo systemctl enable scheduler.service
```

Після того, як сервіс доданий, його треба активувати наступною командою:

```
sudo systemctl start scheduler
```

Після старту сервісу, його статус можна подивитись за допомогою команди `systemctl status scheduler`. Статус чат боту приведений на рисунку 3.37. В даному випадку сервіс був запущений коректно та працює.

```
nanit@ubuntu:~/scheduler$ sudo systemctl stop scheduler
nanit@ubuntu:~/scheduler$ sudo systemctl start scheduler
nanit@ubuntu:~/scheduler$ sudo systemctl status scheduler
• scheduler.service - ZIEIT scheduler bot
     Loaded: loaded (/etc/systemd/system/scheduler.service; static; vendor preset: enabled)
     Active: active (running) since Thu 2022-02-10 11:35:40 EET; 4s ago
   Main PID: 1832 (start.sh)
      Tasks: 21
     Memory: 133.0M
        CPU: 8.191s
     CGroup: /system.slice/scheduler.service
             ├─1832 /bin/sh /home/nanit/scheduler/start.sh
             └─1833 java -jar scheduler.jar

Feb 10 11:35:41 ubuntu start.sh[1832]: [11:35:41] INFO: HHH000412: Hibernate Core {[WORKING]}
Feb 10 11:35:41 ubuntu start.sh[1832]: [11:35:41] INFO: HHH000206: hibernate.properties not fo
Feb 10 11:35:41 ubuntu start.sh[1832]: [11:35:41] INFO: HCANN000001: Hibernate Commons Annotat
Feb 10 11:35:41 ubuntu start.sh[1832]: [11:35:41] INFO: HikariPool-1 - Starting...
Feb 10 11:35:41 ubuntu start.sh[1832]: [11:35:41] WARN: Registered driver with driverClassName
Feb 10 11:35:42 ubuntu start.sh[1832]: [11:35:42] INFO: HikariPool-1 - Start completed.
Feb 10 11:35:42 ubuntu start.sh[1832]: [11:35:42] INFO: HHH000400: Using dialect: org.hibernat
Feb 10 11:35:43 ubuntu start.sh[1832]: [11:35:43] INFO: Loading schedule ...
Feb 10 11:35:44 ubuntu start.sh[1832]: [11:35:44] INFO: Loaded schedule '1k'
Feb 10 11:35:44 ubuntu start.sh[1832]: [11:35:44] INFO: Loaded schedule '2k'
```

Рисунок 3.37 – Статус сервісу чат боту

Тепер чат бот буде автоматично завантажуватись при старті системи, та при аварійному відключені програми.

### 3.3.3 Перевірка робото спроможності розгорнутої системи

Після розгортання необхідно перевірити робото спроможність системи. Для цього було створено 12 тест кейсів.

Case 1

Назва: Базова робота команд.

Очікуваний результат: Вивід списку доступних команд.

Отриманий результат: Вивід списку доступних команд.

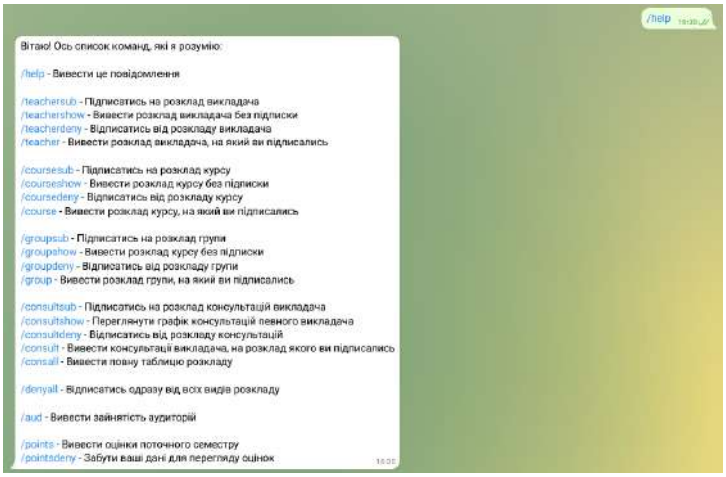


Рисунок 3.38 - Вивід програми

Case 2

Назва: Робота розкладу курсів.

Очікуваний результат: Вивід розкладу обраного курсу у виді зображення.

Отриманий результат: Вивід розкладу обраного курсу у виді зображення.

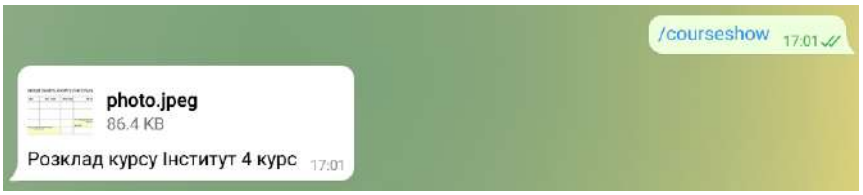


Рисунок 3.39 - Результат запиту

Case 3

Назва: Робота розкладу зайнятості викладача.

Очікуваний результат: Вивід зайнятості обраного викладача у виді зображення.

Отриманий результат: Вивід зайнятості обраного викладача у виді зображення.

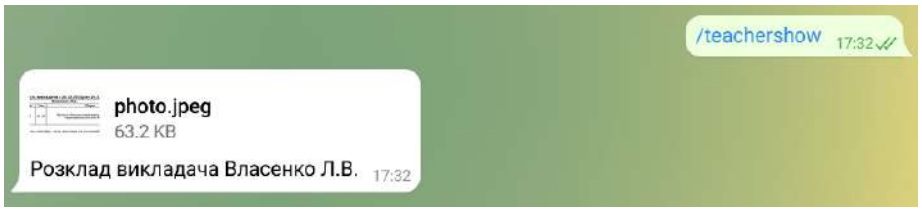


Рисунок 3.40 - Результат запиту

Case 4

Назва: Процес підписки на розклад викладача.

Очікуваний результат: Вивід зайнятості обраного викладача у виді зображення. Повідомлення про оформлену підписку на обраний розклад.

Отриманий результат: Вивід зайнятості обраного викладача у виді зображення. Повідомлення про оформлену підписку на обраний розклад.

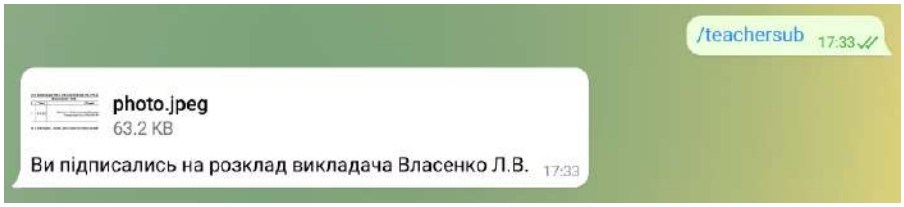


Рисунок 3.41 - Результат запиту

Case 5

Назва: Робота підписки на розклад викладача.

Очікуваний результат: Вивід розкладу викладача, на якого була підписка.

Отриманий результат: Вивід розкладу викладача, на якого була підписка.

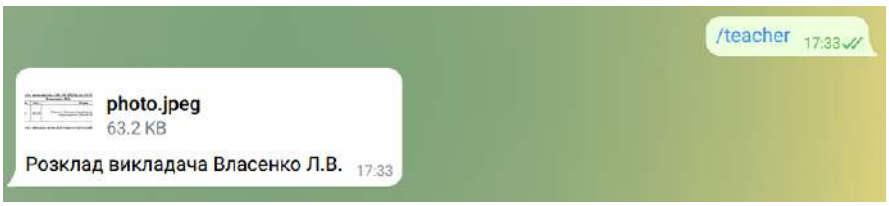


Рисунок 3.42 - Результат запиту

Case 6

Назва: Автоматичне отримання розкладу викладача при оновлені.

Очікуваний результат: Бот автоматично присилає оновлений розклад.

Отриманий результат: Бот автоматично присилає оновлений розклад.

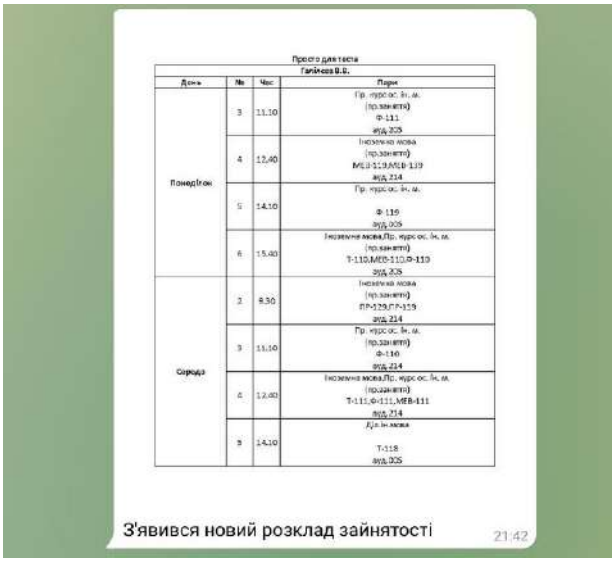


Рисунок 3.43 - Результат запиту

Case 7

Назва: Вивід розкладу групи.

Очікуваний результат: Виведений розклад обраної групи.

Отриманий результат: Виведений розклад обраної групи.


Рисунок 3.44 - Результат запиту

Case 8

Назва: Вивід розкладу консультацій.

Очікуваний результат: Виведений розклад консультацій обраного викладача.

Отриманий результат: Виведений розклад консультацій обраного викладача.

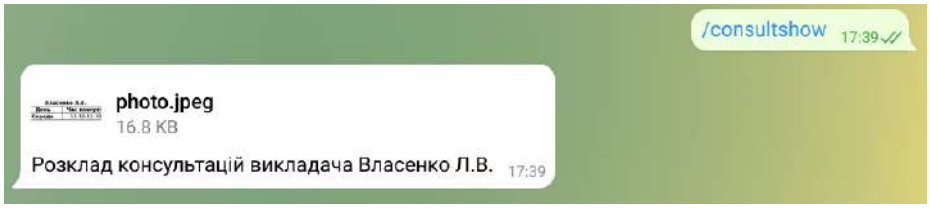


Рисунок 3.45 - Результат запиту

Case 9

Назва: Підписка на розклад консультацій.

Очікуваний результат: Виведений розклад консультацій обраного викладача. Повідомлення про успішну підписку.

Отриманий результат: Виведений розклад консультацій обраного викладача. Повідомлення про успішну підписку.

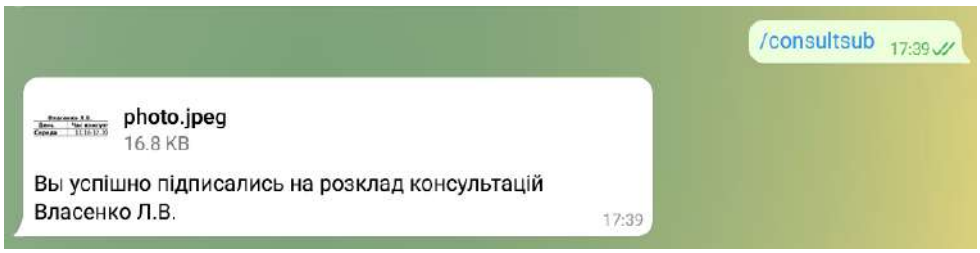


Рисунок 3.46 - Результат запиту

Case 10

Назва: Підписка на розклад групи.

Очікуваний результат: Виведений розклад обраної групи. Повідомлення про успішну підписку.

Отриманий результат: Виведений розклад обраної групи. Повідомлення про успішну підписку.

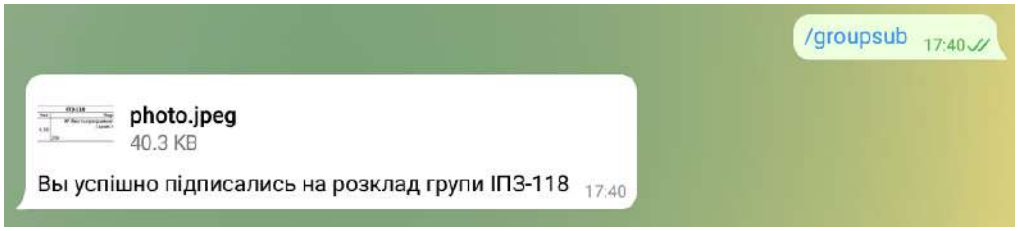


Рисунок 3.47 - Результат запиту

Case 11

Назва: Перевірка підписки на розклад консультацій.

Очікуваний результат: Виведений розклад консультацій, на який була підписка.

Отриманий результат: Програма не відповідає, навіть після декількох запитів. На інші запити у той самий час відповідь коректна.


Рисунок 3.48 - Результат виконання команди декілька разів

Case 12

Назва: Перевірка навантаження при великій кількості запитів.

Очікуваний результат: З інтервалом 50 мс, програма не повинна використовувати більше ніж 600 МБ оперативної пам‘яті, та 80% ЦП.

Отриманий результат: Максимально програма використовувала приблизно 580 МБ оперативної пам‘яті, та 75-80% ЦП. Використання ресурсів до та під час навантаження вказано на рисунках 3.49 та 3.50 відповідно.


Рисунок 3.49 - Використання ресурсів без навантаження


Рисунок 3.50 - Використання ресурсів з навантаженням

## 3.4 Висновки розділу

Відповідно до висунутих вимог, було спроектовано систему модульного типу. Спроектовано структура таблиць бази даних для зберігання даних про

користувачів, підписки на розклад, та службову інформацію. Також було спроектовано конфігурацію програми та базову структуру проекту.

Спроектована система була запрограмована з використанням обраних технологій. Було запрограмовано парсинг різних типів таблиць, їх рендеринг, сутності та взаємодію з базою даних, різні менеджери та парсинг конфігурації.

Запрограмована система була протестована та розгорнута на базі інституту ЗІЕІТ.

# ВИСНОВКИ

Було здійснено огляд поточної системи публікації розкладу ЗІЕІТ та інших ЗВО. Встановлено потребу у створенні системи швидкого сповіщення про новий розклад зі зручним переглядом та персональними налаштуваннями користувача.

Здійснено огляд систем електронного відображення розкладу різних ЗВО, та найбільш ефективних механізмів автоматичного сповіщення користувачів. Встановлено, що сповіщення через месенджер є найбільш доцільним для даної задачі. Враховуючи вимоги до швидкості сповіщення та легкості доступу до даних про розклад для найбільшої групи людей, вирішено реалізувати систему у виді чат-боту для месенджеру. Здійснено порівняльну характеристику найбільш популярних месенджерів, в першу чергу як платформ для створення чат-ботів, та вирішено використовувати месенджер Telegram.

Проведено дослідження щодо можливості отримувати актуальні дані про розклад занять, та виявлено, що парсинг публікованих таблиць є найбільш доцільним.

Для розробки чат-боту було обрано наступний стек технологій: Java 17, Apache POI, Hibernate, TelegramBots. В якості середовища розробки була обрана Intellij IDEA Community Edition.

Спроектована система була запрограмована з використанням обраного технологічного стеку. Готовий чат-бот був протестований та розгорнутий на базі інституту ЗІЕІТ.

Розроблений програмний продукт відповідає поставленим до нього цілям, а саме надає швидкий доступ до розкладу занять з можливістю автоматичного сповіщення, та має простий у використанні інтерфейс.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Офіційний сайт Запорізького Інститут Економіки та Інформаційних Технологій. [Електронний ресурс]. – Режим доступу: www. URL: https://www.zieit.edu.ua/ (дата звернення: 01.02.22)
2. Державна служба статистики України. [Електронний ресурс]. – Режим доступу: www. URL: https://ukrstat.gov.ua (дата звернення: 01.02.22)
3. Автоматизована система управління навчальним закладом. [Електронний ресурс]. – Режим доступу: www. URL: http://mkr.org.ua/ (дата звернення: 01.02.22)
4. Офіційний сайт Запорізького Національного Університету. [Електронний ресурс]. – Режим доступу: www. URL: https://www.znu.edu.ua/ (дата звернення: 01.02.22)
5. Офіційний сайт Запорізького Державного Медичного Університету. [Електронний ресурс]. – Режим доступу: www. URL: https://zsmu.edu.ua/ (дата звернення: 02.02.22)
6. Офіційний сайт Київського Європейського Університету. [Електронний ресурс]. – Режим доступу: www. URL: https://e-u.edu.ua/ (дата звернення: 02.02.22)
7. Офіційний сайт Національного медичного університету імені О.О. Богомольця. [Електронний ресурс]. – Режим доступу: www. URL: http://nmuofficial.com/ (дата звернення: 02.02.22)
8. Чат-бот від Національного Університету "Полтавська Політехніка Імені Юрія Кондратюка". [Електронний ресурс]. – Режим доступу: www. https://nupp.edu.ua/news/rozklad-zanyat-zavzhdi-v-telefoni-universitet-zapustiv-na-telegram-chat-bot-dlya-studentiv.html (дата звернення: 02.02.22)
9. Telegram Messenger. [Електронний ресурс]. – Режим доступу: www. URL: https://telegram.org/ (дата звернення: 04.02.22)

10. Viber Messenger. [Електронний ресурс]. – Режим доступу: www. URL: https://www.viber.com/ (дата звернення: 04.02.22)
11. WhatsApp Messenger. [Електронний ресурс]. – Режим доступу: www. URL: https://www.whatsapp.com/ (дата звернення: 04.02.22)
12. Facebook Messenger. [Електронний ресурс]. – Режим доступу: www. URL: https://www.messenger.com/ (дата звернення: 04.02.22)
13. Визначення парсингу. [Електронний ресурс]. – Режим доступу: www. URL: https://uk.wikipedia.org/wiki/Синтаксичний_аналіз (дата звернення: 05.02.22)
14. Впровадження машинного навчання в документообіг. [Електронний ресурс]. – Режим доступу: www. URL: https://www.tadviser.ru/a/417209/ (дата звернення: 05.02.22)
15. Rendering. [Електронний ресурс]. – Режим доступу: www. URL: https://en.wikipedia.org/wiki/Rendering_(computer_graphics)/ (дата звернення: 06.02.22)
16. Rasterization. [Електронний ресурс]. – Режим доступу: www. URL: https://en.wikipedia.org/wiki/Rasterisation (дата звернення: 06.02.22)
17. Microsoft XLS format specification. [Електронний ресурс]. – Режим доступу: www. URL: https://interoperability.blob.core.windows.net/files/MS-XLS/%5bMS-XLS%5d.pdf (дата звернення: 07.02.22)
18. Microsoft XLSX format specification. [Електронний ресурс]. – Режим доступу: www. URL: https://interoperability.blob.core.windows.net/files/MS-XLSX/%5bMS-XLSX%5d.pdf (дата звернення: 07.02.22)
19. Client-Server architecture. [Електронний ресурс]. – Режим доступу: www. URL: https://cio-wiki.org/wiki/Client_Server_Architecture/ (дата звернення: 10.02.22)
20. What is REST. [Електронний ресурс]. – Режим доступу: www. URL: https://restfulapi.net/ (дата звернення: 10.02.22)

21. Чат бот онлайн банку Монобанк. [Електронний ресурс]. – Режим доступу: www. URL: https://www.monobank.ua/contacts/ (дата звернення: 11.02.22)
22. Мова розмітки YAML. [Електронний ресурс]. – Режим доступу: www. URL: https://wikipedia.org/wiki/YAML (дата звернення: 12.02.22)
23. Специфікація мови YAML. [Електронний ресурс]. – Режим доступу: www. URL: https://yaml.org/spec/1.2.2/ (дата звернення: 12.02.22)
24. Популярність IDE для Java. [Електронний ресурс]. – Режим доступу: www. URL: https://snyk.io/blog/intellij-idea-dominates-the-ide-market-with-62-adoption-among-jvm-developers/ (дата звернення: 14.02.22)
25. Guice Framework. [Електронний ресурс]. – Режим доступу: www. URL: https://github.com/google/guice/ (дата звернення: 16.02.22)
26. Керівництво користувача Guice. [Електронний ресурс]. – Режим доступу: www. URL: https://netvl.github.io/guice/users-guide.html (дата звернення: 17.02.22)
27. Log4j2 Manual. [Електронний ресурс]. – Режим доступу: www. URL: https://logging.apache.org/log4j/2.x/manual/index.html (дата звернення: 18.02.22)
28. Бібліотека Hibernate. [Електронний ресурс]. – Режим доступу: www. URL: https://ru.wikipedia.org/wiki/Hibernate_(библиотека) (дата звернення: 12.03.22)
29. Apache POI. [Електронний ресурс]. – Режим доступу: www. URL: https://poi.apache.org/components/index.html (дата звернення: 15.03.22)
30. Специфікація REST API панелі керування. [Електронний ресурс]. – Режим доступу: www. URL: https://github.com/Nan1t/Scheduler/blob/rest/REST.md (дата звернення: 19.03.22)

# ДОДАТОК А
# ВИХІДНИЙ ПРОГРАМНИЙ КОД

Файл NamespacedKey.java

```
package edu.zieit.scheduler.api;

import com.google.common.base.Preconditions;

import java.io.Serializable;
import java.util.Objects;

/**
 * Useful Id wrapper to identify schedule by namespace (filename) and key (sheet name)
 */
public record NamespacedKey(String namespace, String key) implements Serializable {

    /**
     * Get same key but only with namespace name, without key
     * @return Namespaced key instance
     */
    public NamespacedKey withoutKey() {
        return of(namespace);
    }

    /**
     * Check is namespace equals to another
     * @param another Another NamespacedKey
     * @return true is namespaces equals or false otherwise
     */
    public boolean compareNamespace(NamespacedKey another) {
        return another.namespace().equals(this.namespace());
    }

    @Override
    public String toString() {
        return key.isEmpty() ? namespace : String.format("%s:%s", namespace, key);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o instanceof NamespacedKey key) {
            return this.namespace.equalsIgnoreCase(key.namespace)
                    && this.key.equalsIgnoreCase(key.key);
        }
        return false;
    }

    @Override
    public int hashCode() {
        return Objects.hash(namespace.toLowerCase(), key.toLowerCase());
    }

    /**
     * Create namespaced key from namespace and key
     * @param namespace Namespace name
     * @param key Key name
     * @return New NamespaceKey instance
     */
    public static NamespacedKey of(String namespace, String key) {
        return new NamespacedKey(namespace.strip(), key.strip());
    }

    /**
     * Create namespaced key only from namespace
```

```
     * @param namespace Namespace name
     * @return New NamespaceKey instance
     */
    public static NamespacedKey of(String namespace) {
        return of(namespace, "");
    }

    /**
     * Parse key from string value like 'namespace:key'
     * @param str Raw key string
     * @return Parsed key instance
     */
    public static NamespacedKey parse(String str) {
        Preconditions.checkNotNull(str, "Raw NamespaceKey cannot be null");
        String[] arr = str.split(":");
        if (arr.length == 1) return of(arr[0]);
        if (arr.length != 2) throw new IllegalArgumentException("Invalid namespaced key: " + str);
        return of(arr[0], arr[1]);
    }
}
```

Файл Pair.java

```
package edu.zieit.scheduler.api;

public record Pair<K, V>(K key, V value) {

    public static <K, V> Pair<K, V> of(K key, V val) {
        return new Pair<>(key, val);
    }

}
```

Файл Dao.java

```
package edu.zieit.scheduler.api.persistence;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.query.Query;

import java.io.Serializable;
import java.util.function.Consumer;
import java.util.function.Function;

public abstract class Dao {

    protected final SessionFactory factory;

    public Dao(SessionFactory factory) {
        this.factory = factory;
    }

    /**
     * Create new session and start transaction.
     * After consumer code complete, transaction will be automatically committed
     * @param consumer Session callback
     */
    protected void withSession(Consumer<Session> consumer) {
        try (Session session = factory.openSession()) {
            Transaction t = session.beginTransaction();
            consumer.accept(session);
            t.commit();
        }
    }

    protected <T> T useSession(Function<Session, T> func) {
        try (Session session = factory.openSession()) {
            return func.apply(session);
```