Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

РОЗРОБКА ПІДСИСТЕМИ НОТИФІКАЦІЙ ДЛЯ СИСТЕМИ АВТОМАТИЗАЦІЇ УНІВЕРСИТЕТУ / DEVELOPMENT OF A NOTIFICATION SUBSYSTEM FOR THE UNIVERSITY'S AUTOMATION SYSTEM

Текстова частина до курсової роботи

за спеціальністю “Комп’ютерні науки” 122

Керівник курсової роботи

д.т.н., проф. Глибовець А.М.

_____

(підпис)

“____” _____ 2024 р.

Виконав студент 3-го року навчання,

Освітньої програми “Комп’ютерні науки”, 122

Сметанюк Володимир Олексійович

“____” _____ 2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мережних технологій

проф., д.ф.-м.н.

_____ Г.І. Малашонок

(підпис)

“____”_____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Сметанюку Володимиру Олексійовичу 3-го року навчання факультету інформатики

ТЕМА: РОЗРОБКА ПІДСИСТЕМИ НОТИФІКАЦІЙ ДЛЯ СИСТЕМИ АВТОМАТИЗАЦІЇ УНІВЕРСИТЕТУ

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Календарний план
Анотація
Вступ
Розділ 1. Огляд технологій
Розділ 2. Архітектура сервісу нотифікацій
Розділ 3. Застосування та тестування
Висновки
Список використаної літератури

Дата видачі “_____”_____ 2024 р. Керівник

(підпис)

Завдання отримав

(підпис)

Календарний план виконання курсової роботи

Тема: Розробка підсистеми нотифікацій для системи автоматизації університету

№ з/п	Назва етапу	Термін виконання	Примітка
1	Постановка задачі курсової роботи	Лютий 2023	
2	Дослідження правил побудови листів	Березень 2023	
3	Огляд технологій	Квітень 2023	
4	Виконання практичної частини	Квітень 2023	
5	Оформлення текстової частини	Квітень 2023	
6	Висновок	Квітень 2023	

Студент Сметанюк В.О.

Керівник Глибовець А.М.

“___”_____2024 р.

ЗМІСТ

АНОТАЦІЯ

У даній курсовій роботі розглянуто архітектурні підходи до побудови застосунків, основні технологій для роботи з базами даних, безпекою, засобів поширення сповіщень мовою програмування *Java*, роботу зі спам-фільтрами та поради щодо побудови листів та шаблонів. Використано оглянуті інструменти для розробки системи нотифікацій, розроблено шаблони листів. Наведено результати використання після розширення сервісу курсових робіт.

Ключові слова: мікросервісна архітектура, фреймворк, бібліотека, бази даних, електронна пошта, програмний інтерфейс, міграції бази даних, *Firebase*.

ВСТУП

Розробка систем автоматизації для університетів є ключовим елементом у підвищенні ефективності та продуктивності їх діяльності. Одним із важливих компонентів таких систем є підсистема нотифікацій, що забезпечує своєчасне та точне інформування користувачів про різноманітні події та зміни. Ця підсистема має покращити взаємодію між студентами та викладачами, та сприяти створенню більш зручного та ефективного освітнього середовища. Автоматизована інформаційна система *Smart*, розроблена в університеті, полегшує управління навчальним процесом і забезпечує ефективну роботу з великим обсягом даних. Проте, на жаль, зараз системі не вистачає створення та надсилання сповіщень. Розробці підсистеми нотифікацій і буде присвячено дану роботу.

Актуальність цієї роботи полягає в тому, що, на нашу думку, вдосконалення функціоналу платформи *Smart* сприятиме розв'язанню низки проблем, які виникають у процесі навчання в університеті, зокрема:

1) Студенти не можуть бути своєчасно проінформовані про завершення перевірки або відхилення курсової роботи.
2) Викладачі можуть пропустити завантаження нової курсової роботи.
3) Студенти не можуть бути своєчасно проінформовані про підтвердження або відхилення заявки на тему курсової роботи, а викладачі - про створення таких заявок.

Саме тому, мета роботи - розробка підсистеми нотифікацій для покращення досвіду роботи з автоматизованою інформаційною системою *Smart*.

Завдання: створити мікросервіс, що відповідатиме за надсилання сповіщень користувачам.

Робота складається з трьох розділів.

У першому розділі наведено огляд архітектурних підходів до побудови застосунків, основних технологій для роботи з базами даних, безпекою, засобів поширення сповіщень, роботи зі спам-фільтрами та порад щодо побудови листів та шаблонів.

У другому розділ описано структуру сервісу та деталі реалізації компонентів.

У третьому розділі наведено приклади побудови шаблонів листів, їхнє тестування та використання сервісу нотифікацій в іншому сервісі.

У даній роботі було використано наступні методи наукового дослідження: аналіз, порівняння, класифікація, експеримент, опис.

РОЗДІЛ 1. ОГЛЯД ТЕХНОЛОГІЙ

1.1 Архітектурні підходи до побудови серверних застосувань

1.1.1 Монолітна архітектура

Монолітна архітектура - це традиційний підхід до розробки програмного забезпечення, при якому весь застосунок створюється як єдина та цілісна система. Усі його компоненти взаємодіють між собою і розгортаються на одному сервері або групі серверів [1].

Монолітна архітектура має декілька переваг:

1) зазвичай вони простіші в розробці та тестуванні, оскільки всі частини програми взаємодіють напряму, що сприяє швидкому та легкому вирішенню проблем та відслідковуванню помилок;
2) такі застосунки можуть бути простішими в управлінні, оскільки все програмне забезпечення працює на одній технологічній платформі [1].

Проте, монолітні застосунки мають більше недоліків:

1) дуже складні під час масштабування, оскільки розширення системи може бути обмеженим через високу зв’язаність усіх компонентів;
2) складна структура монолітних застосунків може ускладнити процеси розробки та підтримки;
3) якщо в будь-якому модулі є помилка, це може вплинути на доступність усієї програми;
4) моноліт обмежений технологіями, які вже використовуються в ньому;
5) невелика зміна монолітної програми вимагає повторного розгортання всього застосунку.

Отже, монолітна архітектура підходить для невеликих і середніх проєктів зі стабільним функціоналом, для розробки швидких реалізацій-прикладів та демонстрацій.

1.1.2 Мікросервісна архітектура

Застосунок з мікросервісною архітектурою зазвичай складається з набору невеликих автономних сервісів, кожен з яких є незалежним та має реалізовувати єдину бізнес-задачу в обмеженому контексті. Обмежений контекст забезпечує явний розподіл зі сторони бізнес-моделі, у межах якої існує модель домену [2].

До переваг мікросервісної архітектури можна віднести наступне:

1) оскільки кожен мікросервіс - це окрема незалежна функціональна одиниця, то після локалізації помилок необхідно виправляти код лише в ізольованій частині застосунку;
2) відмова одного мікросервіса зазвичай не веде до відмови усього застосунку;
3) більш легкий підхід до масштабування та менша локальна кодова база порівняно з монолітною архітектурою [2];
4) мала зв’язаність та ізольованість даних, що полегшує масштабування системи, оскільки зміни впливають лише на один сервіс;
5) незалежний набір технологій та мов програмування для кожного окремого мікросервіса [2];
6) можливість незалежної розробки кожного мікросервіса в межах визначеного інтерфейсу комунікацій.

До складнощів можна віднести наступне:

1) порівняно з монолітом, кожен мікросервіс простіший і легший, проте система загалом є важчою для розуміння та подальшої розробки;
2) ускладнюється пошук та виявлення помилок, адже для знаходження, ймовірно, необхідно буде пройти один або кілька рівнів комунікації;
3) необхідність у додатковому строго визначеному прошарку комунікацій, зміни якого вплинуть на інші мікросервіси, які його використовують.

Мікросервісна архітектура більш складна у реалізації та проектуванні, проте може бути надійнішою на відміну від моноліту. Тому даний тип більше підходить для великих та чітко визначених систем з продуманим розподілом задач. [2]

1.1.3 Комунікація між мікросервісами

Зв’язок між сервісами може бути реалізований за допомогою різних типів комунікації, проте усіх їх можна поділити на два загальних класи за послідовністю запитів. Синхронний - коли клієнт не може продовжувати роботи, поки не отримає відповідь на запит, та асинхронний - коли клієнт після надсилання запиту продовжує виконувати роботу, а відповідь, що може повернутись пізніше, буде використано після отримання. Зазвичай за використання другого підходу значення запитів не є критичним, тому не відбувається блокування програми [3].

Також типи комунікації можна поділити на класи за кількістю отримувачів. Такими є: *point-to-point* - коли запит може опрацьовуватись єдиним сервісом, або ж *publisher/subscriber* - коли запит може опрацьовуватись одразу багатьма сервісами. Як правило використовується найкраще поєднання цих методів, для реалізації певного застосунку [3].

Найчастіше у мікросервісній архітектурі використовується синхронний *point-to-point* тип комунікації (з використанням протоколу *HTTP* та підходу *RESTFUL*). Перевагами такого поєднання є висока швидкість роботи та простота у використанні. Проте, незважаючи на популярність даного підходу, потрібно боротись з наступними вадами:

1) відповідальність за обробку помилок та створення повторних запитів лежить на клієнті;
2) утворення точок відмов: несправність одного сервісу може призвести до неможливості роботи усього застосунку.

Іншою реалізацією такого підходу може бути використання фреймворку *gRPC (Google Remote Procedure Call)*. Перевагами є вища швидкість через використання механізму *Protocol Buffers*, що серіалізує структуровані дані у бінарному форматі, генератори клієнтів з описаних сервісів, вбудована валідація даних та наявність чітких типів [4].

Ще одним підходом може бути використання протоколу *AMQP (Advanced Message Queuing Protocol)* та брокерів повідомлень. Основний принцип роботи *AMQP* полягає в тому, що опрацьований сервісом запит відправляється через обмінник, який маршрутизує його в одну або кілька черг на основі ключа маршрутизації. Далі клієнт отримує повідомлення з черги обробки. Через наявність тем (*topics*), прив'язаних до черг, можлива як *point-to-point* реалізація, так і *publisher/subscriber*. *RabbitMQ* та *Apache ActiveMQ* - найпопулярніші брокери повідомлень [5].

Отже, перевагами використання асинхронного типу комунікації є наступне:

1) підвищення продуктивності та ефективності за рахунок виконання кількох запитів одночасно;
2) можливість широкомовного розповсюдження результатів виконання;
3) перенесення відповідальності за керування процесами комунікації на окремий компонент - брокер;
4) відсутність блокування роботи після запиту.

1.1.4 Використання *Java Message Service* для обміну повідомленнями

Java Message Service (JMS) надає інтерфейс для асинхронного обміну повідомленнями в рамках платформи *Java*. JMS дозволяє застосункам обмінюватись повідомленнями між різними компонентами системи та забезпечує надійну і гнучку комунікацію. Можливе використання як *point-to-point,* так і *publisher/subscriber* моделей. За використання фреймворка *Spring,* існує можливість інтеграції з *JMS* через клас *JmsTemplate* [6].

Підсумовуючи, комунікація в мікросервісній архітектурі може бути реалізована багатьма способами, проте єдиного правильного немає. Кожен спосіб має певні переваги та недоліки, тому вибір залежить від певної задачі.

1.2 Інструменти для розробки сервісних застосунків на *Java*

1.2.1 *Java HTTP Server*

Пакет *com.sun.net.httpserver* надає простий високорівневий програмний інтерфейс для *HTTP* сервера. Можлива підтримка як *HTTP*, так і *HTTPS* протоколів. Основна задача для розробника - реалізувати інтерфейс *HttpHandler*, який використовується як зворотній виклик для обробки вхідних запитів від клієнтів. Запит і відповідь на нього називається обміном (*exchange*), він представлений класом *HttpExchange*. Для прослуховування вхідних *TCP*-запитів використовується клас *HttpServer*, який переадресовує запити на обробники, зареєстровані у сервері [7].

Хоча для розробки можна використовувати базові та гнучкі засоби, що надає *Java HTTP Server,* проте за використання даного пакету розробнику необхідно самостійно створювати обробники та відповідати за валідацію даних, що інколи ускладнює та уповільнює роботу.

1.2.2 *Java Spring Boot*

Іншим інструментом для розробки серверних застосунків на *Java* з більш високим рівнем абстракції є *Java Spring Boot.*

Java Spring Boot - це інструмент з відкритим кодом, що спрощує використання фреймворків на основі *Java,* надає можливість гнучкого налаштування, ефективний робочий процес та надійну пакетну обробку [8].

Застосунок, написаний з використанням даного фреймворку, має зручну роботу з налаштуваннями завдяки автоконфігураціям. Таке конфігурування полягає у тому, що застосунок може використовувати параметри за замовчуванням, які за потреби можна змінити. Даний інструмент надає кілька стандартних функцій з великою гнучкістю, що тим самим може скоротити час розробки та уникнути помилок через дублювання коду [8].

Ще однією перевагою *Spring Boot* є перенесення відповідальності за керування виконанням програми на спеціальний *IoC*-контейнер (I*nversion of Control*). Для початку розробки необхідно обрати одну з кількох стандартних залежностей (*Spring Starters*), без додаткового написання коду. А будь-який додатковий функціонал можна утворити перевизначенням базового або розширенням [8].

Просте створення автономного застосунку, швидкий запуск, декларативний підхід налаштування параметрів, можливість використовувати різноманітні додаткові модулі та анотації також є значною корисністю даного інструменту [9].

Отже, *Java Spring Boot* є потужним інструментом для розробки серверних застосунків з високим рівнем абстракції, до того ж популярним, а отже, має велику кількість базових та сторонніх модулів.

1.3 Інструменти *Java Spring Boot* для розробки серверних застосунків

1.3.1 *Spring Data JPA. Hibernate*

Spring Data JPA - це модуль фреймворку *Spring,* який надає для розробників зручний інтерфейс для роботи з базами даних на основі *Java Persistence API (JPA)*. Полегшення взаємодії з базами даних через високорівневий інтерфейс для операцій *CRUD (create, read, update, delete)*, а також зручні методи для створення запитів - основні переваги даного інструменту [10].

До зручності та простоти використання можна також віднести автоматичне створення запитів через виклики методів інтерфейсу, підтримку різних баз даних через використання *JPA,* підтримку пагінації та сортування, створення низькорівневих *SQL* запитів на основі назв методів у визначених інтерфейсах [10].

Spring Data JPA не реалізує *JPA*, але надає додатковий рівень абстракції. Інші *ORM*-фреймворки (*Object-Relational Mapping*), які імплементують його, забезпечують взаємодію з базою даних, керування зв'язками між сутностями, а також перетворення записів з реляцій на об'єкти *Java* та навпаки [10].

Одним із таких фреймворків є *Hibernate,* що використовується за замовчуванням у *Spring Data JPA.* Його особливостями є те, що він підтримує відкладену ініціалізацію, численні стратегії отримання та оптимістичне блокування з автоматичним керуванням версіями та міткою часу [11].

Хоча використовувати “чисті” *SQL* запити також можливо, використання абстракції *Spring Data JPA* сприятиме зменшенню кількості помилок та послабленню залежності від конкретної бази даних.

1.3.2 Міграції баз даних. Flyway

Під час розробки комплексних застосунків часто існує потреба в інструменті, який би відповідав за контроль стану бази даних, а також поточну версію та сумісність схеми. Таким інструментом є системи міграцій баз даних. Вони забезпечують впорядкованість та відтворюваність структури і стану бази даних; надають версійність, що дає змогу повернути систему до попереднього стану; консистентність та легкість керування, оскільки можна створити сценарій або послідовність міграцій для зведення бази даних до актуального сумісного стану.

Одним із таких інструментів є фреймворк Flyway. Він дозволяє автоматизувати процес оновлення та підтримки, а також надає зручний спосіб управління структурою баз даних для різних середовищ розробки. Підтримує багато СКБД, серед яких PortgreSQL, MariaDB, Oracle, MySQL та багато інших. Перевагою також є наявність двох типів міграцій: версійні та повторювані [12].

Перші мають формат назви файлів: V<version>__<name>.<sql | java>, де <version> - це унікальна версія міграції, <name> - унікальна назва файлу міграцій записана у форматі snake case, що коротко описує зміни, та <sql | java> - можливі розширення файлів. Такі міграції виконуються один раз під час загального оновлення [12].

Останні мають схожий формат назв, проте не мають версії та починаються з літери “R” замість “V”. Повторювані міграції мають чек-суму та застосовуються кожного разу після її зміни. Також вони виконуються після версійних міграцій, якщо такі є [12].

Як і у багатьох інших системах міграцій, даний інструмент використовує допоміжну таблицю для відслідковування застосованих міграцій, що має назву *flyway_schema_history* [12].

Усі додаткові налаштування мають визначатись у файлі *flyway.conf*, а налаштування щодо розміщення файлів міграцій - у файлі *flyway.locations*. Також варто зазначити, що *Flyway* підтримує концепцію автоналаштувань [12].

1.3.3 Захист серверних застосунків. *Spring Security*

Коли застосунок доходить до фінальної частини розробки або переходить до стану публічного використання, критично важливим є застосування усіх необхідних рівнів захисту та безпеки. Доцільним буде впровадження рівнів доступу та закриття відкритих ендпоінтів (якщо це не публічний *API*), авторизації, автентифікації, базової фільтрації запитів на атаки (ін’єкції та експлойти). З усіма перерахованими вище потребами може впоратись фреймворк *Spring Security*.

Даний інструмент надає наступні механізми автентифікації, проведення ідентифікація користувача: базова автентифікація, *OAuth, JWT, LDAP, OpenID* та інші. Після отримання автентифікаторів, дані про користувача доповнюються інформацією про нього та дозволами. У випадку помилки ідентифікації має бути створено *AuthenticationException* [13].

Авторизація - перевірка рівнів дозволу користувача - відбувається після проходження автентифікації. Правила доступу та ролі можуть бути визначені в конфігураціях інструменту. У випадку помилки авторизації має бути створено *AccessDeniedException* [14].

Наведені вище заходи безпеки працюють за принципом ланцюжка відповідальностей (*chain of responsibilities*), отже кількість та рівні перевірок можуть регулюватись розробником. Також значною перевагою є наявність анотацій *@Secured* та *@RolesAllowed*, що дають можливість визначити перелік ролей, у яких буде доступ до анотованих методів; *@PreAuthorize* та *@PostAuthorize*, що виконують авторизацію до та після виконання метода відповідно; та інші [15].

1.4 Засоби поширення сповіщень

1.4.1 *Email*

Засоби поширення сповіщень є ключовим елементом комунікації між сервісом та користувачем. Основна задача будь-якої системи нотифікацій - оперативне інформування про події, пов’язані з оновленнями або важливими подіями, особливо для тих, які мають часове обмеження або потребують якнайшвидшого реагування. Одним із найпоширеніших засобів залишається електронна пошта.

Розглянемо переваги.

1) Електронну пошту наразі мають майже усі, хто користується інтернетом. Дуже багато сервісів так чи інакше пов’язані з поштою, тому такий канал поширення сповіщень є універсальним.
2) У листі можна надсилати не тільки текст, а й *HTML*-файли із стилями, зображення, отже такий підхід достатньо гнучкий.
3) Процес надсилання листів можна автоматизувати, а також надсилати однакові листи одразу групі людей, тому такий засіб може бути налаштовуваним та масштабованим.

До недоліків можна віднести наступне.

1) Деякі листи можуть потрапляти до спам-групи, або бути заблокованими спам фільтрами. Як наслідок, деяка важлива інформація може бути пропущеною.
2) Через надмірне використання багатьма сервісами, електронні листи часто можуть бути проігноровані користувачем, або “загубитись” серед великої кількості інших листів.
3) Такий підхід до сповіщень залежить від наявності зв’язку, а також, від інших сервісів, які надають відповідні послуги.

Варто зазначити, що для інтеграції сповіщень через пошту необхідно буде налаштовувати *SMTP* (*Simple Mail Transfer Protocol*), адже він є стандартним протоколом обміну повідомленнями між поштовими серверами. Тому правильне налаштування та конфігурація мають велике значення для забезпечення надійності та безпеки.

Хоча *Email* і має деякі ґрунтовні недоліки, він все одно залишається найпоширенішим, універсальним та масштабованим підходом до сповіщень.

1.4.2 *Firebase*

Firebase - це платформа, розроблена *Google*, для розробки додатків. Вона надає велику кількість інструментів для розробників, до яких входить *Firebase Cloud Messaging (FCM)*. Даний інструмент є ефективним засобом для надсилання сповіщень на браузери та мобільні пристрої [16].

З переваг можна виокремити наступне.

1) *FCM* легко інтегрувати з різними платформами, наприклад *IOS, Android, Web, Flutter*, та інші, через велику кількість *SDK* (*Software Development Kit*), а також *REST API* [17].
2) Наявність інструментів для збору аналітичних даних, отже можна аналізувати на скільки добре працюють сповіщення [17].

До недоліків відноситься наступне.

1) Незважаючи на велику кількість інструментів та можливостей, які надає *FCM,* деякі з них можуть вимагати додаткової плати або використання інших сервісів.
2) Неправильне налаштування сервісу може призвести до надмірної кількості сповіщень та спаму [17].
3) Як і у випадку з електронною поштою, через популярність сервісу та велику кількість нотифікацій, інколи сповіщення можуть бути проігноровані.

Отже, використання наведених засобів надає можливість оперативно інформувати користувачів застосунку, але вимагає обережного підходу до налаштувань та використання.

1.5 Методики уникнення спам-фільтрів

1.5.1 Принципи роботи спам-фільтрів

Фільтри спаму призначені для виявлення небезпечних вхідних електронних листів від зловмисників або маркетологів. Перші можуть надсилати листи, що ніби пропонують корисну послугу або захищають від неминучої небезпеки, проте насправді вони призначені для того, щоб перейти за посиланням, яке завантажує шкідливе програмне забезпечення на комп’ютер або спрямовує на небезпечний сайт. Останні можуть надсилати листи з відносно нешкідливим вмістом, але такі повідомлення будуть збільшувати загальний об’єм пошти, і у них може “загубитись” якийсь дійсно важливий лист, наприклад, сповіщення, пов'язане з курсовою роботою [18].

Усі спам-фільтри мають однакову мету - не дати потрапити небажаним листам до вхідної пошти. Проте різні типи таких фільтрів мають різні принципи роботи [18].

1) Фільтри вмісту аналізують текст всередині листа. Часто такі листи можна розпізнати за певними шаблонами. Вони міститимуть пропозиції угод, рекламу відвертих матеріалів або інший вплив на людські емоції, почуття та бажання. Фільтри такого типу можуть шукати певні ключові слова, наприклад «знижка», «залишилось мало часу» або «пропозиція». Для спрацювання фільтру, зазвичай має бути багаторазове використання специфічного слова. [18]

2) Фільтри чорного списку перевіряють чи відправник наявний і відповідному переліку, і якщо так - то не даються такому листу потрапити до вхідних. Такий підхід є достатньо точним, проте списки постійно потрібно оновлювати, адже зловмисник може легко змінити адресу або навіть домен пошти з якої надсилається спам [18].
3) Фільтри заголовків перевіряють заголовок електронної пошти, для визначення легальності джерела. Наприклад адреси Інтернет-протоколу (*IP*), якими зазвичай користуються зловмисники [18].
4) Мовні фільтри перевіряють чи вхідний лист написаний мовою, якою зазвичай спілкується користувач. Інколи зловмисники можуть атакувати користувачів з інших країн, тому такі дії мають призводити до потрапляння листа у папку зі спамом. Хоча у діловому спілкуванні або у спілкування з іноземними клієнтами такий підхід може хибно позитивно реагувати на правильний лист, тому необхідно перевіряти спам-папку у таких випадках [18].
5) Фільтри на основі правил використовують деякі або усі з наведених вище типів фільтрі, які можна налаштовувати під власні потреби або потреби компанії. Наприклад, можна налаштувати спам-фільтр на певний шаблон адреси відправника, деякі специфічні слова у вмісті та заголовку, та інші [18].
6) Баєсівські фільтри створюють патерни на основі того, які листи користувач класифікує як спам. Наприклад, якщо надсилати усі листи від певного користувача до спаму, то даний тип фільтру розпізнає та запам’ятає таке правило і надалі усі такі листи будуть потрапляти до спам-пошти автоматично [18].

1.5.2 Рекомендації до створення листів та шаблонів

Для зменшення ймовірності потрапляння листа до спаму, варто виконувати наступні рекомендації:

1) використовувати потрібний та актуальний вміст у листах [19];
2) використовувати у розділі “тема” точні та однозначні формулювання [19];
3) прикріплювати до листа текстову версію [19];

4) перевірити коректність написання усіх слів [19];
5) уникати написання слів прописом та окличних речень [19];
6) уникати слів, що часто використовують у спам-листах (негайно, встигніть, обмежений час) [19];
7) уникати надмірних копій листа [19];
8) перевіряти чи усі посилання у листі працюють, якщо такі є [19];
9) дотримуватись однієї теми на один лист [19];
10) додавати опис зображень у атрибут “alt” [19];
11) не вкладати важливої інформації у зображення, адже деякі поштові клієнти можуть блокувати зображення до отримання дозволу від отримувача [19];
12) уникати об’ємних зображень, за можливості використовувати компресію [19];
13) віддавати перевагу зображенням формату *GIF,* замість відео [19];
14) не використовувати вкладених форм [19];
15) якщо до листа додаються файли, то варто надати посилання на них, замість прикріплення [19];
16) рекомендована ширина листа - від 600 до 800 пікселей, для того, щоб усі поштові клієнти могли коректо відобразити вміст;
17) використовувати стандартні шрифти (*Arial, Georgia, Verdana*) [19];
18) уникати малого розміру шрифту або прихованого тексту [19].

1.5.3 Додаткові методи

Ще одним важливим кроком може бути налаштування групи сервісів *SPF* запис , *DKIM*, *DMARC*.

SPF (*Sender Policy Framework*) - це записи у сервісі *DNS*, що визначають яким серверам дозволено надсилати листи з певної доменної адреси. Завдяки ньому можна перевірити, що повідомлення, які надходять із певного домену, надіслані поштовими серверами та *IP*-адресами, авторизованими відправником. Варто зазначити, що *SPF* запис може бути лише один для одного домена, проте в одному записі *SPF* може бути визначено декілька серверів і *IP*-адрес [20].

DKIM (*Domain Keys Identified Mail*) - технічний стандарт , який допомагає визначити підроблені адреси електронної пошти і запобігати спуфінгу (коли злочинець видає себе за довірену особу або пристрій) та крадіжці особистих даних. Цей інструмент додає цифровий підпис у заголовок листа, який надалі перевіряється *email*-серверами на цілісність змісту. Як і у випадку з *SPF* даний запис знаходиться у записах *DNS* [20].

DMARC (*Domain-based Message Authentication, Reporting and Conformance*) - визначає, яким чином поштовий сервер отримувача має обробляти вхідні електронні листи, якщо вони не пройшли перевірку (*SPF*, *DKIM* або обидва). Зазвичай якщо лист проходить усі фільтри - він потрапляє у список вхідних листів. Якщо ні, то є три можливих сценарії обробки [20].

1) Нічого не робити - таким чином усі результати фільтрування ігноруються [20].
2) Перемістити до карантину - найпоширеніший підхід - “погані” листи переміщають до спам-папки [20].
3) Відмова - усі листи, які не пройшли перевірку, ігноруються [20].

Таким чином зменшити ймовірність потрапляння листа до спаму можна якщо використовувати наведені підходи.

1.6 Висновки до розділу 1

У даному розділі розглянуто архітектурні підходи до побудови застосунків та їхнє порівняння, інструменти розробки серверних застосунків мовою *Java*, інструменти *Java Spring Boot* для роботи з базами даних та міграціями, засоби поширення сповіщень на базі електронних листів та *Firebase*, принципи роботи спам-фільтрів та рекомендації щодо їх уникнення.

РОЗДІЛ 2. АРХІТЕКТУРА СЕРВІСУ НОТИФІКАЦІЙ

2.1 Загальний опис

Сервіс сповіщень побудований за підходом мікросервісної архітектури, а тому є незалежним у системі сервісів *Smart*. Його основна задача - надсилати листи та сповіщенням користувачам. Використовувати даний сервіс можуть інші сервіси групи *Smart* для цієї задачі. Сервіс побудований із використанням фреймворку *Spring Boot,* бази даних *PostgreSQL* та *Spring Data JPA*, міграцій баз даних *Flyway*, черги *ActiveMQ*, *Spring SMTP* клієнта, *Spring Security* та системи *Firebase*. Узагальнена схема сервісу продемонстрована на рисунку 2.1.

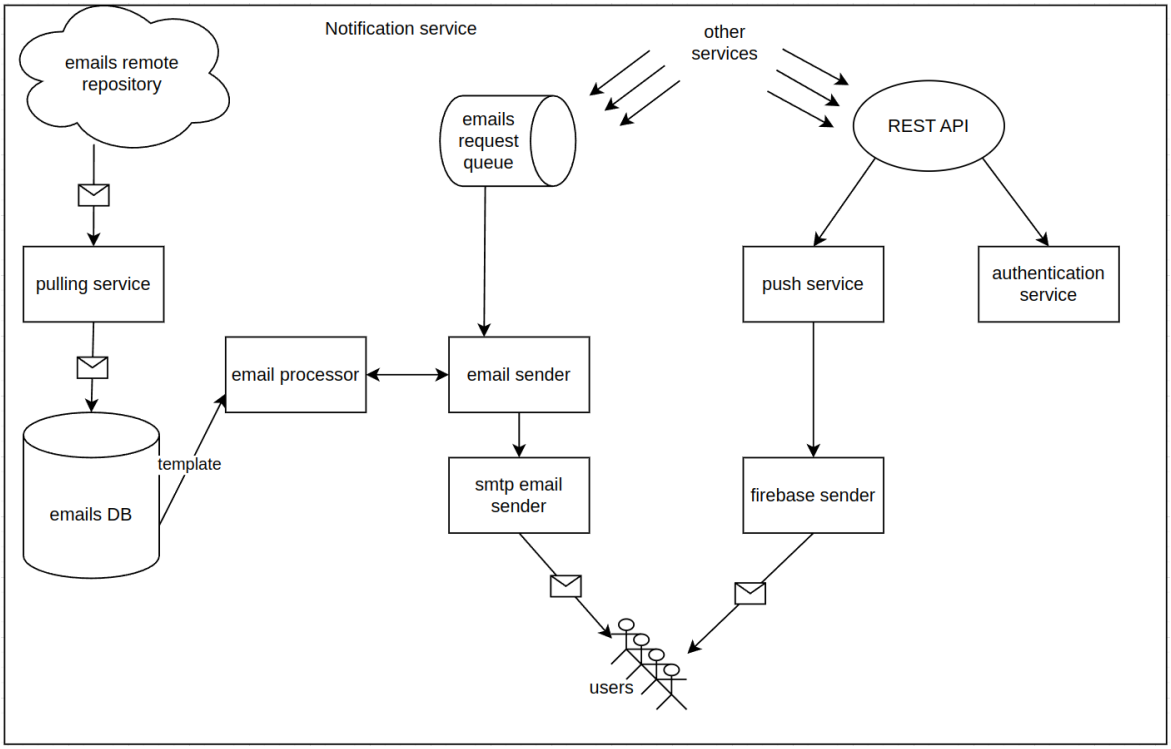

Рисунок 2.1 – Узагальнена схема сервісу нотифікацій

2.2 Побудова бази даних

2.2.1 Схема бази даних

У даному сервісі база даних виконує задачу накопичувача шаблонів листів, тому складається з однієї таблиці *email_templates* (рисунок 2.2).

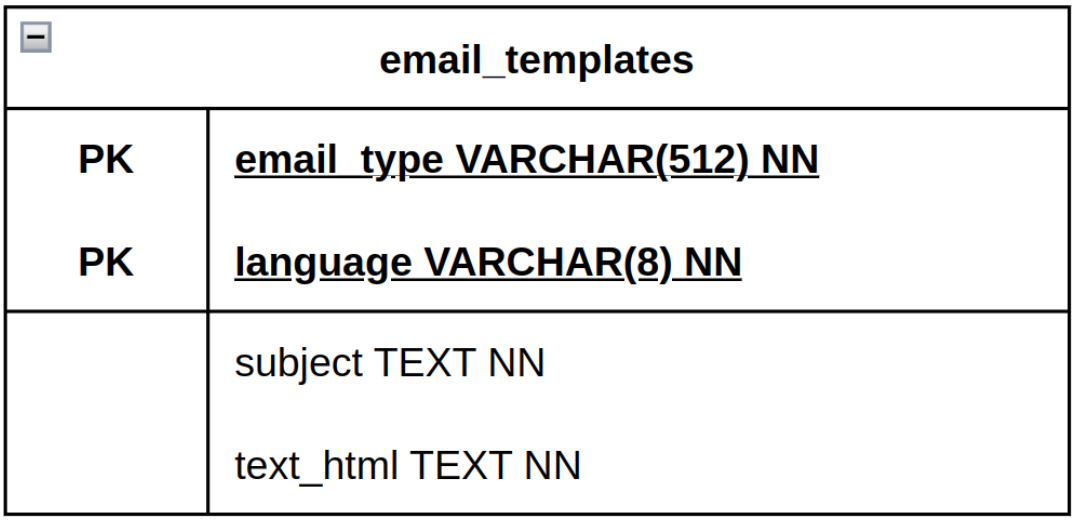

Рисунок 2.2 - Реляційна схема бази даних

email_templates - реляція для збереження шаблонів листів.

Атрибути:

- email_type - ідентифікатор типу листа, обов'язковий атрибут, частина складеного первинного ключа.
- language - визначення мови шаблону, обов'язковий атрибут, частина складеного первинного ключа.
- subject - значення поля "тема" для листа, обов'язковий атрибут.
- text_html - вміст листа, обов'язковий атрибут.

2.2.2 Міграції бази даних

Для керування міграція використано інструмент *Flyway,* через що у загальну схему було додано службову таблицю *flyway_schema_history.* Єдиною міграцією є створення основної таблиці *email_templates* (лістинг 2.1).

```
CREATE TABLE public.email_templates
(
  email_type VARCHAR(512) NOT NULL,
  language   VARCHAR(8)   NOT NULL,
  subject    TEXT         NOT NULL,
  text_html  TEXT         NOT NULL,
  CONSTRAINT email_templates_pkey PRIMARY KEY (email_type, language)
);
```

Лістинг 2.1 - міграція V1__email_templates_table.sql

2.2.3 Реалізація клієнту до бази даних

Для реалізації було використано розширення інтерфейсу *JpaRepository* з фреймворку *Spring Data JPA* (*Storage* (лістинг 2.2) - реалізація *EmailTemplateStorage* (лістинг 2.3)) та класу *EmailTemplateEntityConverter* (лістинг 2.4), що переводить клас *EmailTemplateEntity* у *EmailTemplateStorage.EmailTemplatePersistentInfo* і навпаки.

```
@RequiredArgsConstructor
@Service
public class Storage implements EmailTemplateStorage {

   private final EmailTemplateRepository emailTemplateRepository;
   private final EmailTemplateEntityConverter emailTemplateEntityConverter;

   @Override
   public Optional<EmailTemplatePersistentInfo> getEmailTemplate(EmailType emailType, List<String>
languagesInPriorityOrder) {
      return languagesInPriorityOrder.stream()
            .map(language -> getEmailTemplate(emailType, language))
            .filter(Optional::isPresent)
            .map(Optional::get)
            .findFirst();
   }

   @Override
   public Optional<EmailTemplatePersistentInfo> getEmailTemplate(EmailType emailType, String language) {
      return emailTemplateRepository.findById(new EmailTemplateKey(emailType, language))
            .map(emailTemplateEntityConverter::toEmailTemplate);
   }

   @Override
   public void saveEmailTemplate(EmailTemplatePersistentInfo emailTemplate) {

emailTemplateRepository.saveAndFlush(emailTemplateEntityConverter.toEmailTemplateEntity(emailTemplate));
   }

   @Override
   public void saveAllEmailTemplates(List<EmailTemplatePersistentInfo> emailTemplates) {
      List<EmailTemplateEntity> emailTemplateEntities = emailTemplates.stream()
            .map(emailTemplateEntityConverter::toEmailTemplateEntity)
            .collect(Collectors.toList());
      emailTemplateRepository.saveAll(emailTemplateEntities);
   }

   @Override
   public void updateEmailTemplate(EmailType emailType, String language, String subject, String textHtml) {
      EmailTemplateStorage.EmailTemplatePersistentInfo emailTemplate = getEmailTemplate(emailType,
language).orElseThrow(NotFoundException::new);
      EmailTemplateStorage.EmailTemplatePersistentInfo updatedEmailTemplate = emailTemplate.toBuilder()
            .subject(subject != null ? subject : emailTemplate.subject)
            .textHtml(textHtml != null ? textHtml : emailTemplate.textHtml)
            .build();

emailTemplateRepository.saveAndFlush(emailTemplateEntityConverter.toEmailTemplateEntity(updatedEmailTem
plate));
   }

   @Override
   public void deleteEmailTemplate(EmailType emailType, String language) {
      emailTemplateRepository.deleteById(new EmailTemplateKey(emailType, language));
   }

   @Override
   public void deleteAllEmailTemplates() {
      emailTemplateRepository.deleteAll();
   }

}
```

Лістинг 2.2 - Реалізація класу Storage

```
public interface EmailTemplateStorage {

   @AllArgsConstructor
   @Builder(toBuilder = true, builderMethodName = "")
   class EmailTemplatePersistentInfo {

      public final EmailType emailType;
      public final String language;
      public final String subject;
      public final String textHtml;

   }

   Optional<EmailTemplatePersistentInfo> getEmailTemplate(EmailType emailType, List<String>
languagesInPriorityOrder);

   Optional<EmailTemplatePersistentInfo> getEmailTemplate(EmailType emailType, String language);

   void saveEmailTemplate(EmailTemplatePersistentInfo emailTemplate);

   void saveAllEmailTemplates(List<EmailTemplatePersistentInfo> emailTemplates);

   void updateEmailTemplate(EmailType emailType, String language, String subject, String textHtml);

   void deleteEmailTemplate(EmailType emailType, String language);

   void deleteAllEmailTemplates();

}
```

Лістинг 2.3 - Інтерфейс EmailTemplateStorage

```
@Service
public class EmailTemplateEntityConverter {

   public EmailTemplateStorage.EmailTemplatePersistentInfo toEmailTemplate(EmailTemplateEntity entity) {
      return new EmailTemplateStorage.EmailTemplatePersistentInfo(entity.getEmailTemplateKey().getEmailType(),
entity.getEmailTemplateKey().getLanguage(), entity.getSubject(), entity.getTextHtml());
   }

   public EmailTemplateEntity toEmailTemplateEntity(EmailTemplateStorage.EmailTemplatePersistentInfo
emailTemplate) {
      return new EmailTemplateEntity(new EmailTemplateKey(emailTemplate.emailType, emailTemplate.language),
emailTemplate.subject, emailTemplate.textHtml);
   }

}
```

Лістинг 2.4 - клас EmailTemplateEntityConverter

2.3 Оновлення шаблонів

Для підтримки актуальності та наявності усіх шаблонів листів під час запуску сервісу відбувається викачування усіх листів з віддаленого репозиторію та оновлення відповідних записів у базі даних. Продемонструємо загальну ієрархія класів даного функціоналу (рисунок 2.2).

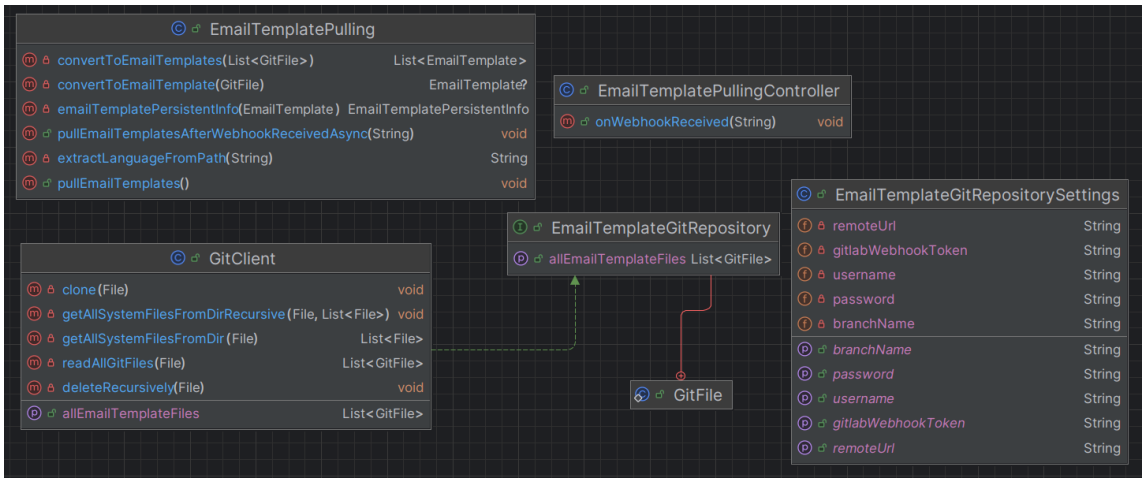

Рисунок 2.2 - Загальна ієрархія класів Pulling пакету

Основна логіка оновлення сконцентрована у класі *EmailTempaltePulling* (лістинг 2.5). У ньому послідовно завантажуються усі доступні шаблони з віддаленого репозиторію через допоміжний клас *GitClient* (лістинг 2.6), що в свою чергу реалізує інтерфейс *EmailTemplateGetRepository*.

```
@Slf4j
@Service
public class EmailTemplatePulling {

   private final EmailTemplateGitRepository gitRepository;
   private final EmailTemplateProvider provider;
   private final EmailTemplateStorage storage;
   private final EmailTemplateGitRepositorySettings settings;
   private final TransactionTemplate transactionTemplate;

   public EmailTemplatePulling(EmailTemplateGitRepository gitRepository, EmailTemplateProvider provider,
EmailTemplateStorage storage, EmailTemplateGitRepositorySettings settings, PlatformTransactionManager
transactionManager) {
      this.gitRepository = gitRepository;
      this.provider = provider;
      this.storage = storage;
      this.settings = settings;
      this.transactionTemplate = new TransactionTemplate(transactionManager);

      pullEmailTemplates();
   }

   @Async
   public void pullEmailTemplatesAfterWebhookReceivedAsync(String gitlabToken) {
      if (!gitlabToken.equals(settings.getGitlabWebhookToken()))
         throw new ForbiddenException();
      pullEmailTemplates();
   }

   public void pullEmailTemplates() {
      log.info("Refreshing email templates...");
      long timePullingStartedMillis = System.currentTimeMillis();
      try {
         List<EmailTemplateGitRepository.GitFile> files = gitRepository.getAllEmailTemplateFiles();
         List<EmailTemplate> emailTemplates = convertToEmailTemplates(files);
         transactionTemplate.executeWithoutResult(transactionStatus -> {
            storage.deleteAllEmailTemplates();

storage.saveAllEmailTemplates(emailTemplates.stream().map(this::emailTemplatePersistentInfo).collect(Collectors.
toList()));
         });
         long timePullingSeconds = (System.currentTimeMillis() - timePullingStartedMillis) / 1000;
         log.info("Refreshing email templates successfully finished in {} seconds", timePullingSeconds);
      } catch (Exception e) {
         long timePullingSeconds = (System.currentTimeMillis() - timePullingStartedMillis) / 1000;
         log.error("Refreshing email failed after {} seconds, | Exception: {}", timePullingSeconds, e);
      }
   }

   private List<EmailTemplate> convertToEmailTemplates(List<EmailTemplateGitRepository.GitFile> files) {
      return files.stream()
            .map((file) -> {
               try {
                  return convertToEmailTemplate(file);
               } catch (Exception e) {
                  log.error("Unable to parse file {}", file.name);
                  return null;
               }
            })
            .filter(Objects::nonNull)
            .collect(Collectors.toList());
```

```
    }

    private EmailTemplate convertToEmailTemplate(EmailTemplateGitRepository.GitFile file) {
        String fileNameWithoutExtension = file.name.substring(0, file.name.lastIndexOf("."));
        String language = extractLanguageFromPath(file.path);

        EmailType emailType = EmailType.valueOf(fileNameWithoutExtension);

        String[] contentLines = file.content.split("\n");
        if (contentLines.length < 2) return null;

        String subject = contentLines[0]
                .replaceAll("<!--", "")
                .replaceAll("-->", "")
                .trim();

        String textHtml = Arrays.stream(contentLines)
                .skip(1)
                .collect(Collectors.joining("\n"));

        return provider.getEmailTemplate(new EmailTemplateStorage.EmailTemplatePersistentInfo(emailType,
language, subject, textHtml));
    }

    private String extractLanguageFromPath(String path) {
        int lastSlashIndex = path.lastIndexOf(File.separator);
        return path.substring(lastSlashIndex - 2, lastSlashIndex);
    }

    private EmailTemplateStorage.EmailTemplatePersistentInfo emailTemplatePersistentInfo(EmailTemplate
emailTemplate) {
        return new EmailTemplateStorage.EmailTemplatePersistentInfo(
                emailTemplate.emailType,
                emailTemplate.language,
                emailTemplate.subject,
                emailTemplate.textHtml
        );
    }

}
```

Лістинг 2.5 - Реалізація класу EmailTemplatePulling

2.4 Формування листа

Будь-яке повідомлення формується з певного шаблону (*template*) з відповідними підстановками деяких параметрів, якщо такі існують. Дана логіка реалізована за допомогою трьох класів: *EmailTemplateProcessor* (лістинг 2.6), *EmailTemplateProvider* (лістинг 2.7), *EmailFromTemplateSender* (лістинг 2.8).

EmailTemplateProcessor відповідає за обробку шаблонів: підстановку та знаходження списку параметрів та формування кінцевого вигляду листа.

EmailTemplateProvider надає шаблон листа за параметрами типу і пріоритетної мови.

EmailFromTemplateSender агрегує два попередні класи та надає можливість надіслати лист із параметрами адреси отримувача, типу листа, мови листа і параметрів шаблона.

```
@Service
public class EmailTemplateProcessor {

   public static final String NESTED_TEMPLATE_PREFIX = "template";
   public static final String FIELD_PREFIX = "parameter";

   public Email processEmailTemplate(EmailTemplate template, Map<String, String> data) {
      String titleUnescaped = unescaped(template.subject);
      Set<String> allFields = Stream.concat(
            template.fields.stream(),
            template.nestedEmailTemplates.stream()
                  .map(emailTemplate -> emailTemplate.fields)
                  .reduce((fields1, fields2) -> Stream.concat(fields1.stream(), fields2.stream()).collect(Collectors.toSet()))
                  .orElse(Collections.emptySet())
                  .stream()
      ).collect(Collectors.toSet());
      Map<String, String> dataFiltered = new HashMap<>(data);
      data.entrySet().stream()
            .filter(dataEntry -> !allFields.contains(dataEntry.getKey()))
            .forEach(dataEntry -> dataFiltered.remove(dataEntry.getKey()));
      Map<String, String> fields =
associateFieldsWithFormatterToFields(findFieldsWithFormatter(template.textHtml), template.fields);
      String bodyUnescaped = unescaped(template.textHtml);
      String bodyProcessedNestedTemplates = substituteNestedEmailTemplates(bodyUnescaped,
template.nestedEmailTemplates);
      String bodyProcessedFields = substituteFields(bodyProcessedNestedTemplates, fields, dataFiltered);
      String bodyPlainText = Jsoup.parse(bodyProcessedFields).text();
      return new Email(titleUnescaped, bodyPlainText, bodyProcessedFields);
   }

   private String unescaped(String value) {
      return new String(value.getBytes(StandardCharsets.UTF_8), StandardCharsets.UTF_8);
   }

   private Map<String, String> associateFieldsWithFormatterToFields(Set<String> fieldsWithFormatters, Set<String>
fields) {
      Map<String, String> result = new HashMap<>();
      fieldsWithFormatters.forEach(fieldWithFormatter -> {
         Optional<String> field = fields.stream().filter(fieldWithFormatter::startsWith).findFirst();
         result.put(fieldWithFormatter, field.orElse(fieldWithFormatter));
      });
      return result;
   }

   private String substituteNestedEmailTemplates(String text, Set<EmailTemplate> nestedEmailTemplates) {
      for (EmailTemplate nestedEmailTemplate : nestedEmailTemplates) {
         text = substituteNestedEmailTemplate(text, nestedEmailTemplate);
      }
      return text;
   }

   private String substituteNestedEmailTemplate(String text, EmailTemplate nestedEmailEmailTemplate) {
      return text.replace("{{" + NESTED_TEMPLATE_PREFIX + "." + nestedEmailEmailTemplate.emailType.name() +
"}}", nestedEmailEmailTemplate.textHtml);
   }

   private String substituteFields(String text, Map<String, String> fields, Map<String, String> data) {
      for (Map.Entry<String, String> fieldsEntry : fields.entrySet()) {
         String fieldWithFormatter = fieldsEntry.getKey();
         String field = fieldsEntry.getValue();
         text = substituteField(text, fieldWithFormatter, field, data);
```

```
        }
        return text;
    }

    private String substituteField(String text, String fieldWithFormatter, String field, Map<String, String> data) {
        return text.replace("{{" + FIELD_PREFIX + "." + fieldWithFormatter + "}}", data.getOrDefault(field, field));
    }

    public Set<String> findFields(String textHtml) {
        return findFieldsWithFormatter(textHtml);
    }

    private Set<String> findFieldsWithFormatter(String textHtml) {
        Matcher matcher = getMather("\\{\\{" + FIELD_PREFIX + "\\.([A-Za-z .(/\")]+)}}", textHtml);
        Set<String> result = new HashSet<>();
        while (matcher.find()) {
            String fieldWithFormatter = matcher.group(1);
            result.add(fieldWithFormatter);
        }
        return result;
    }

    public Set<String> findNestedEmailTemplateNames(String textHtml) {
        Matcher matcher = getMather("\\{\\{" + NESTED_TEMPLATE_PREFIX + "\\.([A-Z ]+)}}", textHtml);
        Set<String> result = new HashSet<>();
        while (matcher.find()) {
            String nestedEmailTemplateName = matcher.group(1);
            result.add(nestedEmailTemplateName);
        }
        return result;
    }

    private Matcher getMather(String regex, String text) {
        Pattern pattern = Pattern.compile(regex);
        return pattern.matcher(text);
    }

}
```

Лістинг 2.6 - Клас EmailTemplateProcessor

```
@Slf4j
@RequiredArgsConstructor
@Service
public class EmailTemplateProvider {

   private final EmailTemplateProcessor emailTemplateProcessor;
   private final EmailTemplateStorage emailTemplateStorage;

   public EmailTemplate getEmailTemplate(EmailType emailType, List<String> languagesInPriorityOrder) {
      EmailTemplate emailTemplate = emailTemplateStorage.getEmailTemplate(emailType,
languagesInPriorityOrder)
            .map(this::fillWithDynamicData)
            .orElse(null);
      if (emailTemplate != null && !emailTemplate.language.equals(languagesInPriorityOrder.get(0)))
         log.warn("Template not found | Name: $templateName, Language:
${languagesInPriorityOrder.firstOrNull()}");
      return emailTemplate;
   }

   public EmailTemplate getEmailTemplate(EmailTemplateStorage.EmailTemplatePersistentInfo
emailTemplatePersistentInfo) {
      return fillWithDynamicData(emailTemplatePersistentInfo);
   }

   private EmailTemplate fillWithDynamicData(EmailTemplateStorage.EmailTemplatePersistentInfo persistentInfo)
{
      Set<String> fields = findFields(persistentInfo.textHtml);
      Set<EmailTemplate> nestedEmailTemplates = findNestedTemplates(persistentInfo.textHtml,
persistentInfo.language);
      return new EmailTemplate(
            persistentInfo.emailType,
            persistentInfo.language,
            persistentInfo.subject,
            persistentInfo.textHtml,
            fields,
            nestedEmailTemplates
      );
   }

   private Set<String> findFields(String textHtml) {
      return emailTemplateProcessor.findFields(textHtml);
   }

   private Set<EmailTemplate> findNestedTemplates(String textHtml, String languageTopPriority) {
      return emailTemplateProcessor.findNestedEmailTemplateNames(textHtml).stream()
            .map(emailTypeName -> Arrays.stream(EmailType.values())
                  .filter(emailType -> emailType.name().equals(emailTypeName))
                  .findFirst()
                  .orElse(null))
            .filter(Objects::nonNull)
            .map(emailType -> emailTemplateStorage.getEmailTemplate(emailType, Arrays.asList(languageTopPriority,
EmailTemplate.DEFAULT_LANGUAGE))
                  .map(nestedEmailTemplate -> new EmailTemplate(nestedEmailTemplate.emailType,
nestedEmailTemplate.language, nestedEmailTemplate.subject, nestedEmailTemplate.textHtml,
findFields(nestedEmailTemplate.textHtml), Collections.emptySet()))
                  .orElse(null))
            .filter(Objects::nonNull)
            .collect(Collectors.toSet());
   }

}
```

Лістинг 2.7 - Клас EmailTemplateProvider

```
@RequiredArgsConstructor
@Service
public class EmailFromTemplateSender {

   private final EmailTemplateProvider emailTemplateProvider;
   private final EmailTemplateProcessor emailTemplateProcessor;
   private final EmailSender emailSender;

   public void sendEmail(String receiverEmail, EmailType emailType, String language, Map<String, String> data)
throws MessagingException, UnsupportedEncodingException {
      sendEmail(List.of(receiverEmail), emailType, language, data);
   }

   public void sendEmail(List<String> receiverEmails, EmailType emailType, String language, Map<String, String>
data) throws MessagingException, UnsupportedEncodingException {
      List<String> languagesInPriorityOrder;
      if (language == null)
         languagesInPriorityOrder = List.of(EmailTemplate.DEFAULT_LANGUAGE);
      else
         languagesInPriorityOrder = List.of(language, EmailTemplate.DEFAULT_LANGUAGE);
      EmailTemplate emailTemplate = emailTemplateProvider.getEmailTemplate(emailType,
languagesInPriorityOrder);
      sendEmail(receiverEmails, emailTemplate, data);
   }

   public void sendEmail(List<String> receiverEmails, EmailTemplate emailTemplate, Map<String, String> data)
throws MessagingException, UnsupportedEncodingException {
      Email email = emailTemplateProcessor.processEmailTemplate(emailTemplate, data);
      emailSender.sendEmail(receiverEmails, email);
   }

}
```

Лістинг 2.8 - Клас EmailFromTemplateSender

2.5 Надсилання сповіщень

2.5.1 *EmailSender*

Для надсилання сповіщень використовується електронна пошта. Основна логіка для надсилання листів описана у інтерфейсі *EmailSender* (лістинг 2.9). Його реалізовує клас *SmtpEmailSender* (лістинг 2.10), що в свою чергу використовує *JavaMailSender* - базовий інструмент з фреймворку Spring, який надає можливість відправляти електронні листи за допомогою *JavaMail API.* Даний функціонал використовує описаний вище клас *EmailFromTemplateSender*.

```
public interface EmailSender {


   void sendEmail(List<String> receiverEmails, Email email) throws MessagingException,
UnsupportedEncodingException;


}
```

Лістинг 2.9 - Інтерфейс EmailSender

```
@Slf4j
@RequiredArgsConstructor
@Service
public class SmtpEmailSender implements EmailSender {


   private final EmailSenderSettings settings;
   private final JavaMailSender emailSender;


   @Override
   public void sendEmail(List<String> receiverEmails, Email email) throws MessagingException,
UnsupportedEncodingException {
      MimeMessage message = emailSender.createMimeMessage();
      MimeMessageHelper messageHelper = new MimeMessageHelper(message, true, Charsets.UTF_8.name());
      String senderEmail = settings.getSenderEmail();
      String senderFullName = settings.getSenderFullName();
      messageHelper.setFrom(senderEmail, senderFullName);
      if (settings.getReplyToEmail() != null && !settings.getReplyToEmail().isEmpty())
         messageHelper.setReplyTo(settings.getReplyToEmail());
      messageHelper.setTo(receiverEmails.toArray(new String[]{}));
      messageHelper.setSubject(email.subject);
      messageHelper.setText(email.textPlain, email.textHtml);
      emailSender.send(message);
   }


}
```

Лістинг 2.10 - Клас SmtpEmailSender

2.5.2 *Firebase*

Інший канал розповсюдження сповіщень реалізований за допомогою платформи *Firebase*. Основний і єдиний метод надсилання *sendPushToTopic* визначений в інтерфейсі *PushSender* (лістинг 2.11). Він імплементований класом *FirebasePushSender* (лістинг 2.12), який використовує *Firebase*, як засіб надсилання. *PushService* (лістинг 2.13) використовує *FirebasePushSender* та є сервісною обгорткою, що додатково перевіряє наявність доступу для надсилання сповіщень.

```
public interface PushSender {

  void sendPushToTopic(String topic, String title, String text) throws PushException;

}
```

Лістинг 2.11 - Інтерфейс PushSender

```
@Slf4j
@Service
public class FirebasePushSender implements PushSender {


   public FirebasePushSender(FirebaseSettings firebaseSettings) {
      if (StringUtils.isEmpty(firebaseSettings.getAdminSdkKeyFilePath())) {
         log.warn("Firebase admin SDK key not configured..");
      } else {
         try {
            FirebaseOptions options = FirebaseOptions.builder()

.setCredentials(GoogleCredentials.fromStream(Files.newInputStream(Paths.get(firebaseSettings.getAdminSdkKeyFil
ePath()))))
                  .build();
            FirebaseApp.initializeApp(options);
         } catch (IOException e) {
            log.error("Unable to initialize firebase service | {}", e.getMessage());
         }
      }
   }


   @Override
   public void sendPushToTopic(String topic, String title, String text) throws PushException {
      Message message = Message.builder()
            .setNotification(Notification.builder()
                  .setTitle(title)
                  .setBody(text)
                  .build())
            .setTopic(topic)
            .build();
      try {
         FirebaseMessaging.getInstance().send(message);
      } catch (FirebaseMessagingException e) {
         log.error("Failed to send push to topic | {}", e.getMessage());
         throw new PushException();
      }
   }


}
```

Лістинг 2.12 - Клас FirebasePushSender

```
@Slf4j
@RequiredArgsConstructor
@Service
public class PushService {


   private final PushSender pushSender;


   public void sendPushToTopic(String topic, String title, String text) throws PushException {
      if (!SecurityContextAccessor.hasPermission(PermissionsEnum.SEND_PUSH_TO_TOPIC))
         throw new ForbiddenException();
      log.info("User {} is sending push message with title {}", SecurityContextAccessor.getBehalfOnEmail(), title);
      pushSender.sendPushToTopic(topic, title, text);
   }


}
```

Лістинг 2.13 - Клас PushService

2.6 Комунікація

2.6.1 *EmailQueue*

Одним із засобів комунікації між сервісом нотифікацій та іншими є черга *ActiveMQ* та *Spring JMS*. Даний підхід забезпечує неблокуючу роботу з сервісом сповіщень, адже не потрібно очікувати відповіді після ініційованого надсилання листа. До того ж, у випадку несправності сервісу, ненадіслані листи залишаться у черзі та чекатимуть на надсилання після відновлення роботи. Обробник запитів на надсилання реалізований у класі *EmailSendRequestReceiver* (лістинг 2.14), а клієнт, що формує запити - у класі *EmailSender* (лістинг 2.15).

```
@Slf4j
@Service
@RequiredArgsConstructor
public class EmailSendRequestReceiver {

    private final EmailFromTemplateSender emailFromTemplateSender;

    @JmsListener(id = "email-request-receiver", destination = "${email.queue}")
    public void emailSendRequestReceiver(final EmailSendRequest emailSendRequest) {
        log.info("Received new email send request: {}", emailSendRequest);
        try {
            emailFromTemplateSender.sendEmail(emailSendRequest.getToEmail(), emailSendRequest.getEmailType(),
emailSendRequest.getLanguage(), emailSendRequest.getParameters());
        } catch (MessagingException | UnsupportedEncodingException e) {
            log.error("Unable to process request | message {}", e.getMessage());
            throw new RuntimeException(e);
        }
    }

}
```

Лістинг 2.14 - Клас EmailSendRequestReceiver

```
@Slf4j
@RequiredArgsConstructor
public class EmailSender {

    private final String emailQueue;
    private final JmsTemplate jmsTemplate;

    public void sendEmailRequest(final EmailSendRequest request) {
        log.info("Send email request: {}", request);
        jmsTemplate.convertAndSend(emailQueue, request);
    }

}
```

Лістинг 2.15 - Клас EmailSender

2.6.2 *REST API*

Ще одним підходом для комунікації є сервер з *REST* архітектурою. У ньому присутня логіка авторизації та частина *API* для надсилання *Firebase* сповіщень.

Клас *JwtTokenDecoder* (лістинг 2.16) відповідає за декодування та перевірку JWT-токенів. Клас *JWTAuthorizationFilter* (лістинг 2.17) відповідає знаходження та обробку JWT-токена з запиту.

```
@Component
@RequiredArgsConstructor
public class JwtTokenDecoder {

    private final SecurityConstants securityConstants;

    public String getServerNameFromToken(String token) {
        DecodedJWT jwt = JWT.require(Algorithm.HMAC512(securityConstants.getSecret().getBytes()))
                .build()
                .verify(token.replaceFirst(securityConstants.getTokenPrefix(), ""));
        return jwt.getSubject();
    }

}
```

Лістинг 2.16 - Клас JwtTokenDecoder

```
@Slf4j
public class JWTAuthorizationFilter extends BasicAuthenticationFilter {

   private final SecurityConstants securityConstants;
   private final UserPermissionService userPermissionService;
   private final JwtTokenDecoder jwtTokenDecoder;

   public JWTAuthorizationFilter(AuthenticationManager authManager, SecurityConstants securityConstants,
                         UserPermissionService userPermissionService, JwtTokenDecoder jwtTokenDecoder) {
      super(authManager);
      this.securityConstants = securityConstants;
      this.userPermissionService = userPermissionService;
      this.jwtTokenDecoder = jwtTokenDecoder;
   }

   @Override
   protected void doFilterInternal(HttpServletRequest req,
                           HttpServletResponse res,
                           FilterChain chain) throws IOException, ServletException {
      SecurityContextHolder.getContext().setAuthentication(getAuthentication(req));
      chain.doFilter(req, res);
   }

   private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {
      String token = request.getHeader(securityConstants.getHeaderString());
      String behalfOn = getBehalfOnEmail(request);
      if (StringUtils.hasText(token) && StringUtils.hasText(behalfOn) &&
token.startsWith(securityConstants.getTokenPrefix())) {
         try {
            String user = jwtTokenDecoder.getServerNameFromToken(token);
            if (securityConstants.getServersLogins().contains(user)) {
               final String role = userPermissionService.getRoleByEmail(behalfOn);
               final Set<PermissionsEnum> permissions = userPermissionService.getPermissions(role);
               final AuthenticatedUser authenticatedUser = AuthenticatedUser.builder()
                     .token(token)
                     .serverEmail(user)
                     .behalfOnEmail(behalfOn)
                     .role(role)
                     .permissions(permissions)
                     .build();


               log.debug("Authenticate user: {}", authenticatedUser);


               return new UsernamePasswordAuthenticationToken(authenticatedUser, null,
authenticatedUser.getPermissions());
            }
            return null;
         } catch (TokenExpiredException ex) {
            log.error("Token expired", ex);
         } catch (JWTVerificationException ex) {
            log.error("Token verification exception", ex);
         } catch (Exception ex) {
            log.error("Can't build user session", ex);
         }
      }
      return null;
   }
```

```
    private String getBehalfOnEmail(HttpServletRequest request) {
        return Optional.ofNullable(request.getHeader(securityConstants.getBehalfOn()))
                .orElseGet(() -> request.getParameter(securityConstants.getBehalfOn()));
    }


}
```

Лістинг 2.17 - Клас JWTAuthorizationFilter

Клас *RoleEventListener* (лістинг 2.18) відповідає за прослуховування івентів, пов'язаними з ролями від сервісу акаунтів.

```
@Slf4j
@Component
@RequiredArgsConstructor
public class RoleEventListener {

    private final UserPermissionService userPermissionService;

    @JmsListener(id = "notifications-server.user-role-changed-listener", destination = "${user.role.changed}",
containerFactory = TOPIC_JMS_LISTENER_CONTAINER_FACTORY)
    void userRoleChangedEvent(UserRoleChangedEvent event) {
        if (event.getServer() == ServerNameEnum.NOTIFICATIONS_SERVER) {
            log.info("Received role change event: {}", event);
            userPermissionService.deleteCacheForEmail(event.getEmail());
        }
    }

    @JmsListener(id = "notifications-server.default-role-changed-listener", destination = "${default.role.changed}",
containerFactory = TOPIC_JMS_LISTENER_CONTAINER_FACTORY)
    void defaultRoleChangedEvent(DefaultRoleChangedEvent event) {
        if (event.getServer() == ServerNameEnum.NOTIFICATIONS_SERVER) {
            log.info("Received default role change event: {}", event);
            userPermissionService.deleteCache();
        }
    }

    @JmsListener(id = "notifications-server.role-deleted-listener", destination = "${role.deleted}", containerFactory =
TOPIC_JMS_LISTENER_CONTAINER_FACTORY)
    void roleDeletedEvent(RoleDeletedEvent event) {
        if (event.getServer() == ServerNameEnum.NOTIFICATIONS_SERVER) {
            log.info("Received role deleted event: {}", event);
            userPermissionService.deleteCache();
        }
    }


}
```

Лістинг 2.18 - Клас RoleEventListener

Клас *UserPermissionService* (лістинг 2.19) відповідає за дозволи користувачів використовуючи сервіс акаунтів.

```
@Service
@Slf4j
public class UserPermissionService {

  private final UserServerRoleControllerApi userServerRoleControllerApi;
  private final Map<String, Set<PermissionsEnum>> roleToPermissions;
  private final LoadingCache<String, String> userRoles = CacheBuilder.newBuilder()
      .expireAfterWrite(1, TimeUnit.HOURS)
      .build(new CacheLoader<>() {
        @Override
        public String load(final String s) {
          return getUserRole(s);
        }
      });

  public UserPermissionService(
    UserServerRoleControllerApi userServerRoleControllerApi,
    @Value("classpath:/permissions.yaml") final Resource resource
  ) {
    this.userServerRoleControllerApi = userServerRoleControllerApi;
    try (InputStream inputStream = resource.getInputStream()) {
      PermissionMatrix matrix = new Yaml().loadAs(inputStream, PermissionMatrix.class);

      roleToPermissions = matrix.entrySet().stream()
        .map(
          entry -> Map.entry(
            entry.getKey(),
            entry.getValue().stream().map(PermissionsEnum::valueOf).collect(Collectors.toUnmodifiableSet())
          )
        )
        .collect(Collectors.toUnmodifiableMap(Map.Entry::getKey, Map.Entry::getValue));
    } catch (final IOException ex) {
      throw new RuntimeException("Can't read permissions", ex);
    }
  }

  public void deleteCacheForEmail(String email) {
    userRoles.invalidate(email);
  }

  public void deleteCache() {
    userRoles.invalidateAll();
  }

  public String getRoleByEmail(String email) {
    try {
      return userRoles.get(email);
    } catch (Exception e) {
      throw new ServiceErrorException("Can't find role for user with email: " + email);
    }
  }

  private String getUserRole(String email) {
    CommonResponse.StringResponse response = userServerRoleControllerApi.getUserRoleNameOnServer(email);
    if (response.getError() != null) {
      log.error("Can't get role for email: {}. Reason: {}", email, response.getError());
      return null;
    }
```

```
        return response.getResult();
    }

    public Set<PermissionsEnum> getPermissions(String role) {
        final Set<PermissionsEnum> permissions = roleToPermissions.get(role);
        if (CollectionUtils.isEmpty(permissions)) {
            throw new ServiceErrorException("Can't find permissions for role: " + role);
        }
        return permissions;
    }

    public Map<String, Object> getCurrentUserRole() {
        if (SecurityContextAccessor.isNotAuthenticated()) {
            throw new ForbiddenException();
        }

        final String role = getRoleByEmail(SecurityContextAccessor.getBehalfOnEmail());
        return ImmutableMap.<String, Object>builder()
                .put("name", role)
                .put("permissions", getPermissions(role))
                .build();
    }

    private static class PermissionMatrix extends HashMap<String, Collection<String>> {

    }
}
```

Лістинг 2.19 - Клас RoleEventListener

PermissionsEnum (лістинг 2.20) визначає перелік можливих дозволів, а клас *AuthenticatedUser* (лістинг 2.21) представляє успішно авторизованого користувача.

```
public enum PermissionsEnum implements GrantedAuthority {


    SEND_PUSH_TO_TOPIC;


    @Override
    public String getAuthority() {
        return name();
    }


}
```

Лістинг 2.20 - PermissionsEnum

```
@Builder
@Data
public class AuthenticatedUser {

    private final String token;
    private final String serverEmail;
    private final String behalfOnEmail;
    private final Set<PermissionsEnum> permissions;
    private final String role;

    public boolean hasPermission(PermissionsEnum permission) {
        return permissions.contains(permission);
    }

}
```

Лістинг 2.21 - Клас AuthenticatedUser

Клас *WebSecurity (лістинг 2.22)* визначає правила безпеки застосунку, *CORS (Cross-Origin Resource Sharing)* політику та ендпоінти для яких необхідна авторизація.

```
@EnableWebSecurity
@RequiredArgsConstructor
@EnableConfigurationProperties(SecurityConstants.class)
public class WebSecurity extends WebSecurityConfigurerAdapter {


   private final SecurityConstants securityConstants;
   private final JwtTokenDecoder jwtTokenDecoder;
   @Value("${server.swagger.enabled}")
   private final boolean swaggerEnabled;
   private final UserPermissionService userPermissionService;
   private final ObjectMapper objectMapper;


   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http.cors().and().csrf().disable();
      if (swaggerEnabled)
         http.authorizeRequests().antMatchers("/v2/api-docs", "/swagger-resources/**", "/configuration/**",
"/swagger-ui.html/**", "/webjars/**").permitAll();
      http.authorizeRequests()
            .antMatchers("/api/v1/email/template/git/webhook")
            .permitAll();
      http.authorizeRequests()
            .anyRequest().authenticated()
            .and()
            .exceptionHandling()
            .authenticationEntryPoint(
                  (HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) -
> {
                     ResponseError error = new ResponseError();
                     error.setCode(HttpServletResponse.SC_UNAUTHORIZED);
                     error.setMessage("Unauthorized");


                     writeObjectToResponse(Response.of(error), response);
                  })
            .and()
            .addFilter(new JWTAuthorizationFilter(authenticationManager(), securityConstants, userPermissionService,
jwtTokenDecoder))
            // this disables session creation on Spring Security
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
   }


   @Bean
   CorsConfigurationSource corsConfigurationSource() {
      CorsConfiguration corsConfiguration = new CorsConfiguration();
      corsConfiguration.setAllowedMethods(asList("GET", "POST", "PUT", "DELETE", "OPTIONS", "HEAD"));


      final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
      source.registerCorsConfiguration("/**", corsConfiguration.applyPermitDefaultValues());
      return source;
   }


   private void writeObjectToResponse(Response<?> object, HttpServletResponse response) throws IOException {
      response.getWriter().write(objectMapper.writeValueAsString(object));
      response.setContentType(MediaType.APPLICATION_JSON_VALUE);
      response.setStatus(HttpServletResponse.SC_OK);
   }


}
```

Лістинг 2.22 - Клас AuthenticatedUser

Клас *PushController* (лістинг 2.23) визначає ендпоінти для надсилання сповіщень через *Firebase*.

```
@RequiredArgsConstructor
@RestController
@RequestMapping("/api/v1/push")
public class PushController {


    private final PushService pushService;


    @PostMapping("/send/topic/{topic}")
    public Response<Boolean> sendPushToTopic(@PathVariable("topic") String topic, @RequestBody SendPushDto
sendPushDto) throws PushException {
        pushService.sendPushToTopic(topic, sendPushDto.title, sendPushDto.text);
        return Response.of(true);
    }


}
```

Лістинг 2.23 - Клас PushController

2.7 Висновки до розділу 2

У розділі описано структуру сервісу нотифікацій, побудову реалізацію основних функціональних класів, підхід та реалізацію оновлень шаблонів листів, формування листів, деталі надсилання листів за допомогою електронної пошти та сповіщень за допомогою *Firebase*, комунікаційний рівень для сервісу нотифікацій.

РОЗДІЛ 3. ЗАСТОСУВАННЯ ТА ТЕСТУВАННЯ

3.1 Створення шаблонів листів

Під час створення шаблонів дотримувались порад, описаних у першому розділі, для дотримання максимальної сумісності із поштовими клієнтами та мінімальної потенційної оцінки на спам. Було створено 5 шаблонів:

1) *NEW_COURSE_WORK_REQUEST.html* (лістинг 3.1) - використовується для сповіщення про створення нової заявки на тему курсової.
2) *NEW_COURSE_WORK_REQUEST_SENT.html* (лістинг 3.2) - використовується для сповіщення про успішне надсилання заявки на тему курсової.
3) *COURSE_WORK_REQUEST_APPROVED.html* (лістинг 3.3) - використовується для сповіщення про підтвердження заявки на тему курсової.
4) *COURSE_WORK_REQUEST_REJECTED.html* (лістинг 3.4) - використовується для сповіщення про відхилення заявки на тему курсової.
5) COURSE_WORK_EVALUATED.html (лістинг 3.5) - використовується для сповіщення про завершення оцінювання курсової роботи.

```
<!doctype html>
<html>
<head>
  <title>Новий запит на курсову роботу</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>
<body>
<table style="font-family: 'Helvetica Neue'; width: 600px">
  <tr>
    <td style="text-align: center;border-radius: 10px 10px 0 0; background: #7099c8;">
      <img src="https://h2020.ukma.edu.ua/wp-content/uploads/2016/01/logo-NaUKMA.png"
          alt="NATIONAL UNIVERSITY OF KYIV MOHYLA ACADEMY"
          width="100px"
          style="border-radius: 20%; padding: 25px; margin: 10px; background: white;"/>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 25px">
      <h1 style="text-align: center;">Новий запит на курсову роботу</h1>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 6px; padding-bottom: 25px; line-height: 25px">
      <p>
        Шановний викладач! Ви отримали новий запит на виконання курсової роботи.<br>
        Тема: {{parameter.topic}}.<br>
        Для детальної інформації перейдіть до вебсайту
        <a href="https://smart.ukma.edu.ua/">smart.ukma.edu.ua</a>.
      </p>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; height:100px; border-radius: 0 0 10px 10px; background: #7099c8;">
      <p style="font-size: 12px; color: #ffffff; line-height: 16px;">
        Національний університет "Києво-Могилянська академія"<br>
        вул. Сковороди 2, Київ 04070, Україна<br>
        тел.: 425-60-59<br>
        вебсайт:
        <a href="https://smart.ukma.edu.ua/" style="color: #ffffff;">smart.ukma.edu.ua</a>
      </p>
    </td>
  </tr>
</table>
</body>
</html>
```

Лістинг 3.1 - Шаблон NEW_COURSE_WORK_REQUEST.html

```
<!doctype html>
<html>
<head>
  <title>Запит на курсову роботу надіслано</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>
<body>
<table style="font-family: 'Helvetica Neue'; width: 600px">
  <tr>
    <td style="text-align: center;border-radius: 10px 10px 0 0; background: #7099c8;">
      <img src="https://h2020.ukma.edu.ua/wp-content/uploads/2016/01/logo-NaUKMA.png"
          alt="NATIONAL UNIVERSITY OF KYIV MOHYLA ACADEMY"
          width="100px"
          style="border-radius: 20%; padding: 25px; margin: 10px; background: white;"/>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 25px">
      <h1 style="text-align: center;">Запит на курсову роботу надіслано</h1>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 6px; padding-bottom: 25px; line-height: 25px">
      <p>
        Вашу заявку на тему "{{parameter.topic}}" було надіслано викладачу.
      </p>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; height:100px; border-radius: 0 0 10px 10px; background: #7099c8;">
      <p style="font-size: 12px; color: #ffffff; line-height: 16px;">
        Національний університет "Києво-Могилянська академія"<br>
        вул. Сковороди 2, Київ 04070, Україна<br>
        тел.: 425-60-59<br>
        вебсайт:
        <a href="https://smart.ukma.edu.ua/" style="color: #ffffff;">smart.ukma.edu.ua</a>
      </p>
    </td>
  </tr>
</table>
</body>
</html>
```

Лістинг 3.2 - Шаблон NEW_COURSE_WORK_REQUEST_SENT.html

```
<!doctype html>
<html>
<head>
  <title>Запит на курсову роботу схвалено</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>

<body>
<table style="font-family: 'Helvetica Neue'; width: 600px">
  <tr>
    <td style="text-align: center;border-radius: 10px 10px 0 0; background: #7099c8;">
      <img src="https://h2020.ukma.edu.ua/wp-content/uploads/2016/01/logo-NaUKMA.png"
          alt="NATIONAL UNIVERSITY OF KYIV MOHYLA ACADEMY"
          width="100px"
          style="border-radius: 20%; padding: 25px; margin: 10px; background: white;"/>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 25px">
      <h1 style="text-align: center;">Запит на курсову роботу схвалено</h1>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 6px; padding-bottom: 25px; line-height: 25px">
      <p>Ваш запит на виконання курсової роботи на тему "{{parameter.topic}}" був схвалений
викладачем.</p>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; height:100px; border-radius: 0 0 10px 10px; background: #7099c8;">
      <p style="font-size: 12px; color: #ffffff; line-height: 16px;">
        Національний університет "Києво-Могилянська академія"<br>
        вул. Сковороди 2, Київ 04070, Україна<br>
        тел.: 425-60-59<br>
        вебсайт:
        <a href="https://smart.ukma.edu.ua/" style="color: #ffffff;">smart.ukma.edu.ua</a>
      </p>
    </td>
  </tr>
</table>
</body>
</html>
```

Лістинг 3.3 - Шаблон COURSE_WORK_REQUEST_APPROVED.html

```
<!doctype html>
<html>
<head>
  <title>Запит на курсову роботу відхилено</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>

<body>
<table style="font-family: 'Helvetica Neue'; width: 600px">
  <tr>
    <td style="text-align: center;border-radius: 10px 10px 0 0; background: #7099c8;">
      <img src="https://h2020.ukma.edu.ua/wp-content/uploads/2016/01/logo-NaUKMA.png"
          alt="NATIONAL UNIVERSITY OF KYIV MOHYLA ACADEMY"
          width="100px"
          style="border-radius: 20%; padding: 25px; margin: 10px; background: white;"/>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 25px">
      <h1 style="text-align: center;">Запит на курсову роботу відхилено</h1>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 6px; padding-bottom: 25px; line-height: 25px">
      <p>Ваш запит на виконання курсової роботи на тему "{{parameter.topic}}" був відхилений.</p>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; height:100px; border-radius: 0 0 10px 10px; background: #7099c8;">
      <p style="font-size: 12px; color: #ffffff; line-height: 16px;">
        Національний університет "Києво-Могилянська академія"<br>
        вул. Сковороди 2, Київ 04070, Україна<br>
        тел.: 425-60-59<br>
        вебсайт:
        <a href="https://smart.ukma.edu.ua/" style="color: #ffffff;">smart.ukma.edu.ua</a>
      </p>
    </td>
  </tr>
</table>
</body>
</html>
```

Лістинг 3.4 - Шаблон COURSE_WORK_REQUEST_REJECTED.html

```
<!doctype html>
<html>
<head>
  <title>Ваша курсова робота оцінена</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>

<body>
<table style="font-family: 'Helvetica Neue'; width: 600px">
  <tr>
    <td style="text-align: center;border-radius: 10px 10px 0 0; background: #7099c8;">
      <img src="https://h2020.ukma.edu.ua/wp-content/uploads/2016/01/logo-NaUKMA.png"
          alt="NATIONAL UNIVERSITY OF KYIV MOHYLA ACADEMY"
          width="100px"
          style="border-radius: 20%; padding: 25px; margin: 10px; background: white;"/>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 25px">
      <h1 style="text-align: center;">Ваша курсова робота оцінена</h1>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; padding-top: 6px; padding-bottom: 25px; line-height: 25px">
      <p>
        Ваша курсова робота на тему "{{parameter.topic}}" була оцінена викладачем.<br>
        Для детальної інформації перейдіть до вебсайту
        <a href="https://smart.ukma.edu.ua/">smart.ukma.edu.ua</a>.
      </p>
    </td>
  </tr>
  <tr>
    <td style="text-align: center; height:100px; border-radius: 0 0 10px 10px; background: #7099c8;">
      <p style="font-size: 12px; color: #ffffff; line-height: 16px;">
        Національний університет "Києво-Могилянська академія"<br>
        вул. Сковороди 2, Київ 04070, Україна<br>
        тел.: 425-60-59<br>
        вебсайт:
        <a href="https://smart.ukma.edu.ua/" style="color: #ffffff;">smart.ukma.edu.ua</a>
      </p>
    </td>
  </tr>
</table>
</body>
</html>
```

Лістинг 3.5 - Шаблон COURSE_WORK_EVALUATED.html

3.2 Перевірка шаблонів листів

Для перевірки та оцінки створених шаблонів було використано сервіс *Miltrap.* У ньому можна побачити оцінку за 5-бальною шкалою на рівень спаму, відсоток сумісності з поштовими клієнтами, зауваження з використання розмітки та стилів, наявність домену та *IP*-адреси відправника у чорних списках. Продемонструємо дані аналізу (таблиця 3.1).

Template	*Spam report*	*Market support*
NEW_COURSE_WORK_REQUEST.html	0.2/5	96.5%
NEW_COURSE_WORK_REQUEST_SENT.html	0.3/5	96.5%
COURSE_WORK_REQUEST_APPROVED.html	0.2/5	96.5%
COURSE_WORK_REQUEST_REJECTED.html	0.2/5	96.5%
COURSE_WORK_EVALUATED.html	0.3/5	96.5%

Таблиця 3.1 - Аналіз шаблонів

3.3 Розширення сервісу курсових робіт

Сервіс курсових робіт - це один із підсервісів системи *Smart*, побудований також як мікросервіс. Його основні задачі: управління заявками на курсові та власне курсовими роботами, інструменти оцінювання робіт та перевірка вмісту плагіату, створення звітів по користувачу. Основна логіка знаходиться у класах *ApplicationService* (лістинг 3.6) та *CouseWorkService* (лістинг 3.7), у них і було додано використання сервісу нотифікацій.

```
@Service
@Transactional(propagation = Propagation.REQUIRED, rollbackFor = BaseException.class)
@Slf4j
public class ApplicationService extends BaseService<ApplicationEntity, ApplicationView, Integer> {

    private final ApplicationValidationService applicationValidationService;
    private final CourseWorkRepository courseWorkRepository;
    private final InfoServerUtils infoServerUtils;
    private final EmailSender emailSender;

    public ApplicationService(ApplicationValidationService applicationValidationService,
                              CourseWorkRepository courseWorkRepository,
                              InfoServerUtils infoServerUtils,
                              EmailSender emailSender) {
        super(ApplicationEntity.class, ApplicationEntity::new);
        this.applicationValidationService = applicationValidationService;
        this.courseWorkRepository = courseWorkRepository;
        this.infoServerUtils = infoServerUtils;
        this.emailSender = emailSender;
    }

    @Override
    public Criteria<ApplicationEntity> parse(String restrict) {
        ApplicationCriteria criteria = new ApplicationCriteria(restrict);
        if (SecurityContextAccessor.hasPermission(PermissionsEnum.VIEW_APPLICATION))
            criteria.setApplication_owner(SecurityContextAccessor.getBehalfOnEmail());
        if (SecurityContextAccessor.hasPermission(PermissionsEnum.VIEW_FACULTY_APPLICATION))

criteria.setDepartments(infoServerUtils.getDepartmentsListByFacultyId(infoServerUtils.getFacultyOfUser(SecurityC
ontextAccessor.getBehalfOnEmail())));
        log.debug("We form Application criteria: \n:{}", criteria);
        if (isEmpty(criteria.getCourse_work_statuses()))
            criteria.setCourse_work_statuses(ImmutableList.of(Status.ACTIVE));
        return criteria;
    }

    @Override
    public List<Map<String, Object>> getList(Criteria<ApplicationEntity> criteria, Collection<String> fields) {
        List<Map<String, Object>> res = super.getList(criteria, fields);
        log.debug("We prepare list of applications: \n: {}", res.size());
        return res;
    }

    @Override
    public void postCreate(final ApplicationEntity entity) {
        entity.setStatus(ApplicationStatusEnum.NEW);
        entity.setTimeOfCreation(LocalDateTime.now(ZoneOffset.UTC));
        entity.setTimeOfModify(LocalDateTime.now(ZoneOffset.UTC));
        emailSender.sendEmailRequest(new EmailSendRequest(
                EmailType.NEW_COURSE_WORK_REQUEST_SENT,
                "uk",
                entity.getEmail(),
                Map.of("topic", entity.getCourseWorkTheme().getTopic())
        ));
        emailSender.sendEmailRequest(new EmailSendRequest(
                EmailType.NEW_COURSE_WORK_REQUEST,
                "uk",
                entity.getCourseWorkTheme().getCourseNumber().getEmail(),
                Map.of("topic", entity.getCourseWorkTheme().getTopic())
```

```
        ));
    }

    @Override
    public void postUpdate(final ApplicationEntity entity) {
        entity.setTimeOfModify(LocalDateTime.now(ZoneOffset.UTC));
    }

    @Override
    public boolean delete(Integer id) {
        ApplicationEntity entity = repository.findById(id)
            .orElseThrow(() -> new NoSuchEntityException(ApplicationEntity.class.getName(), "by id " + id));
        validationService.validForDelete(entity);
        repository.delete(entity);
        log.debug("We delete application: {}", entity);
        return true;
    }

    public Boolean changeApplicationStatus(Integer id, ApplicationStatusEnum status) {
        log.debug("We try to change application with id: {} to status: {}", id, status);
        ApplicationEntity applicationEntity = repository.findById(id)
                .orElseThrow(() -> new NoSuchEntityException(ApplicationEntity.class.getName(), "by id " + id));
        applicationValidationService.validForStatusChange(applicationEntity, status);
        applicationEntity.setStatus(status);
        postUpdate(applicationEntity);
        if (ApplicationStatusEnum.APPROVED == status)
            approveApplication(applicationEntity);
        repository.save(applicationEntity);
        log.debug("We change application with id: {} to status: {}", id, status);
        if (ApplicationStatusEnum.REJECTED == status) {
            emailSender.sendEmailRequest(new EmailSendRequest(
                    EmailType.COURSE_WORK_REQUEST_REJECTED,
                    "uk",
                    applicationEntity.getEmail(),
                    Map.of("topic", applicationEntity.getCourseWorkTheme().getTopic())
            ));
        }

        return true;
    }

    private void approveApplication(ApplicationEntity applicationEntity) {
        rejectOtherApplicationsInThisCourseWork(applicationEntity);
        setApprovedApplicationToCourseWork(applicationEntity);
        rejectOtherStudentApplications(applicationEntity);
    }

    private void rejectOtherStudentApplications(ApplicationEntity applicationEntity) {
        ApplicationCriteria criteria = new ApplicationCriteria();
        criteria.setApplication_owner(applicationEntity.getEmail());
        criteria.setYearPeriod(applicationEntity.getCourseWorkTheme().getCourseNumber().getYearPeriod());
        List<ApplicationEntity> applicationsOfUser = criteriaRepository.find(criteria);
        if (applicationsOfUser == null || applicationsOfUser.size() == 0)
            return;
        for (ApplicationEntity application : applicationsOfUser) {
            if (application.getStatus() == ApplicationStatusEnum.NEW) {
                application.setStatus(ApplicationStatusEnum.REJECTED_BY_APPROVE);
                log.debug("We will reject application {} in other course theme because student has already approved",
```

```
application.getId());
            }
        }
        repository.saveAll(applicationsOfUser);
        for (ApplicationEntity application : applicationsOfUser) {
            if (application.getStatus() == ApplicationStatusEnum.NEW) {
                emailSender.sendEmailRequest(new EmailSendRequest(
                        EmailType.COURSE_WORK_REQUEST_REJECTED,
                        "uk",
                        application.getEmail(),
                        Map.of("topic", application.getCourseWorkTheme().getTopic())
                ));
            }
        }
    }

    private void setApprovedApplicationToCourseWork(ApplicationEntity applicationEntity) {
        applicationEntity.getCourseWorkTheme().setApprovedApplication(applicationEntity);
        emailSender.sendEmailRequest(new EmailSendRequest(
                EmailType.COURSE_WORK_REQUEST_APPROVED,
                "uk",
                applicationEntity.getEmail(),
                Map.of("topic", applicationEntity.getCourseWorkTheme().getTopic())
        ));
    }

    private void rejectOtherApplicationsInThisCourseWork(ApplicationEntity applicationEntity) {
        CourseWorkEntity courseWork = applicationEntity.getCourseWorkTheme();
        List<ApplicationEntity> rejected = new ArrayList<>();
        for (ApplicationEntity applicationOther : courseWork.getApplications()) {
            if (!applicationOther.getId().equals(applicationEntity.getId())) {
                applicationOther.setStatus(ApplicationStatusEnum.REJECTED_BY_APPROVE);
                postUpdate(applicationOther);
                rejected.add(applicationOther);
                log.debug("We will reject application: {} because other application approved", applicationOther);
            }
        }
        if (rejected.size() > 0)
            repository.saveAll(rejected);
    }

    public Map<String, Object> getApprovedApplicationByCourseWork(final int courseWorkId, final List<String>
fields) {
        final ApplicationEntity application = courseWorkRepository.findById(courseWorkId)
                .map(CourseWorkEntity::getApprovedApplication)
                .orElseThrow(() -> new NoSuchEntityException("No approved coursework application for coursework: " +
courseWorkId));
        applicationValidationService.validForView(application);
        return converter.convert(application, fields);
    }

    public Boolean studentHasApprovedApplication() {
        return applicationValidationService.checkIfUserHasApprovedApplicationForCurrentYear();
    }
}
```

Лістинг 3.6 - Клас ApplicationService

```
@Service
@Transactional(propagation = Propagation.REQUIRED, rollbackFor = BaseException.class)
public class CourseWorkService extends BaseService<CourseWorkEntity, CourseWorkView, Integer> {


   private final CourseWorkValidationService courseWorkValidationService;
   private final MarkingConfigRepository markingConfigRepository;
   private final InfoServerUtils infoServerUtils;
   private final EmailSender emailSender;


   public CourseWorkService(CourseWorkValidationService courseWorkValidationService,
                            MarkingConfigRepository markingConfigRepository,
                            InfoServerUtils infoServerUtils,
                            EmailSender emailSender) {
      super(CourseWorkEntity.class, CourseWorkEntity::new);
      this.courseWorkValidationService = courseWorkValidationService;
      this.markingConfigRepository = markingConfigRepository;
      this.infoServerUtils = infoServerUtils;
      this.emailSender = emailSender;
   }


   @Override
   public Criteria<CourseWorkEntity> parse(String restrict) {
      CourseWorkCriteria criteria = new CourseWorkCriteria(restrict);
      if (isEmpty(criteria.getStatuses()))
         criteria.setStatuses(ImmutableList.of(Status.ACTIVE));
      return criteria;
   }


   @Override
   public void postCreate(final CourseWorkEntity entity) {
      entity.setTimeOfCreation(LocalDateTime.now(ZoneOffset.UTC));
      entity.setStatus(Status.ACTIVE);
   }


   @Override
   public boolean delete(Integer id) {
      CourseWorkEntity courseWork = repository.findById(id)
         .orElseThrow(() -> new NoSuchEntityException(CourseWorkEntity.class.getName(), "by id " + id));
      validationService.validForDelete(courseWork);
      repository.delete(courseWork);
      return true;
   }


   public boolean setTeacherMark(int mark, int courseWorkId) {
      CourseWorkEntity courseWork = repository.findById(courseWorkId)
         .orElseThrow(() -> new NoSuchEntityException(CourseWorkEntity.class.getName(), "by id " +
courseWorkId));
      courseWorkValidationService.hasRightToPutTeacherMark(courseWork);
      courseWorkValidationService.validTeacherMark(mark, courseWork);
      courseWork.setTeacherMark(mark);
      courseWork.setTimeOfTeacherMark(LocalDateTime.now(ZoneOffset.UTC));
      calculateTotalMark(courseWork, TEACHER);
      repository.save(courseWork);
      emailSender.sendEmailRequest(new EmailSendRequest(
            EmailType.COURSE_WORK_EVALUATED,
            "uk",
            courseWork.getApprovedApplication().getEmail(),
            Map.of("topic", courseWork.getApprovedApplication().getCourseWorkTheme().getTopic())
```

```
        ));

        return true;
    }

    public boolean setCommissionMark(int mark, int courseWorkId) {
        CourseWorkEntity courseWork = repository.findById(courseWorkId)
            .orElseThrow(() -> new NoSuchEntityException(CourseWorkEntity.class.getName(), "by id " +
courseWorkId));
        courseWorkValidationService.hasRightToPutCommissionMark(courseWork);
        courseWorkValidationService.validCommissionMark(mark, courseWork);
        courseWork.setCommissionMark(mark);
        courseWork.setTimeOfCommissionMark(LocalDateTime.now(ZoneOffset.UTC));
        calculateTotalMark(courseWork, TotalMarkingType.COMMISSION);
        repository.save(courseWork);
        emailSender.sendEmailRequest(new EmailSendRequest(
                EmailType.COURSE_WORK_EVALUATED,
                "uk",
                courseWork.getApprovedApplication().getEmail(),
                Map.of("topic", courseWork.getApprovedApplication().getCourseWorkTheme().getTopic())
        ));

        return true;
    }

    public boolean setPlagiarismPercentage(double percentage, int courseWorkId) {
        CourseWorkEntity courseWork = repository.findById(courseWorkId)
                .orElseThrow(() -> new NoSuchEntityException(CourseWorkEntity.class.getName(), "by id " +
courseWorkId));
        courseWorkValidationService.hasRightToPutPlagiarismPercentage(courseWork);
        courseWorkValidationService.validPlagiarismPercentage(percentage);
        courseWork.setPlagiarismPercentage(percentage);
        repository.save(courseWork);
        return true;
    }

    private void calculateTotalMark(CourseWorkEntity entity, TotalMarkingType type) {
        Integer entityFacultyId =
infoServerUtils.getFacultyIdByDepartmentId(entity.getCourseNumber().getDepartmentId());
        Optional<MarkingConfigEntity> configDB = markingConfigRepository.findFirstByFacultyId(entityFacultyId);
        if (configDB.isEmpty() || configDB.get().getMarkingType() == null) {
            if (type == TEACHER)
                entity.setTotalMark(entity.getTeacherMark());
            else
                entity.setTotalMark(entity.getCommissionMark());
        } else {
            TotalMarkingType typeConfig = configDB.get().getMarkingType();
            if (type == TEACHER && typeConfig == TEACHER)
                entity.setTotalMark(entity.getTeacherMark());
            else {
                int mark = 0;
                switch (type) {
                    case SUM:
                        mark = entity.getCommissionMark() + entity.getTeacherMark();
                        break;
                    case TEACHER:
                        mark = entity.getTeacherMark();
                        break;
                    case COMMISSION:
```

```
                mark = entity.getCommissionMark();
                break;
            }
            entity.setTotalMark(Math.min(100, mark));
        }
    }
}
```

Лістинг 3.7 - Клас CourseWorkService

Для класу *ApplicationService* було додано виклики сервісу нотифікацій для наступних ситуацій:

1) після створення нової заявки студенту та викладачу будуть надіслані листи за шаблонами *NEW_COURSE_WORK_REQUEST_SENT* (рисунок 3.1) та *NEW_COURSE_WORK_REQUEST* (рисунок 3.2) відповідно.
2) у випадку відхилення заявки викладачем, студенту буде надіслано лист за шаблоном *COURSE_WORK_REJECTED* (рисунок 3.3).
3) у випадку підтвердження заявки, студенту з підтвердженою заявкою буде надіслано лист за шаблоном *COURSE_WORK_APPROVED* (рисунок 3.4), а усім іншим студентам, які також створили заявки на дану тему, буде надіслано лист за шаблоном *COURSE_WORK_REJECTED*.

Для класу *CourseWorkService* було додано виклики сервісу нотифікацій для наступних ситуацій:

1) після встановлення оцінки викладачем, студенту буде надіслано лист за шаблоном *COUNSE_WORK_EVALUATED* (рисунок 3.5).
2) після встановлення оцінки комісією, студенту буде надіслано лист за шаблоном *COUNSE_WORK_EVALUATED*.

Рисунок 3.1 - Лист за шаблоном NEW_COURSE_WORK_REQUEST_SENT

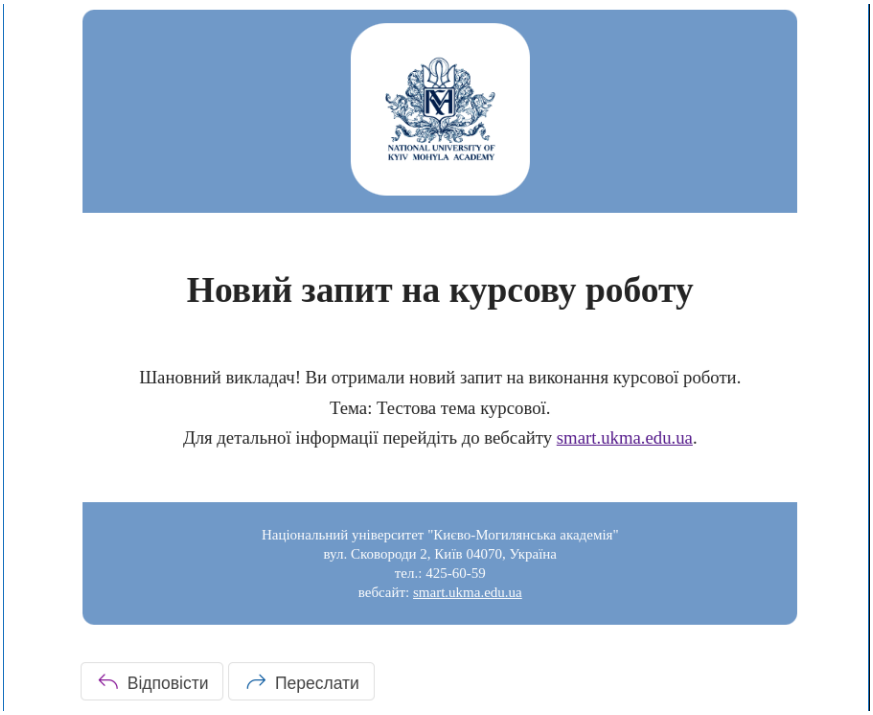

Рисунок 3.2 - Лист за шаблоном NEW_COURSE_WORK_REQUEST

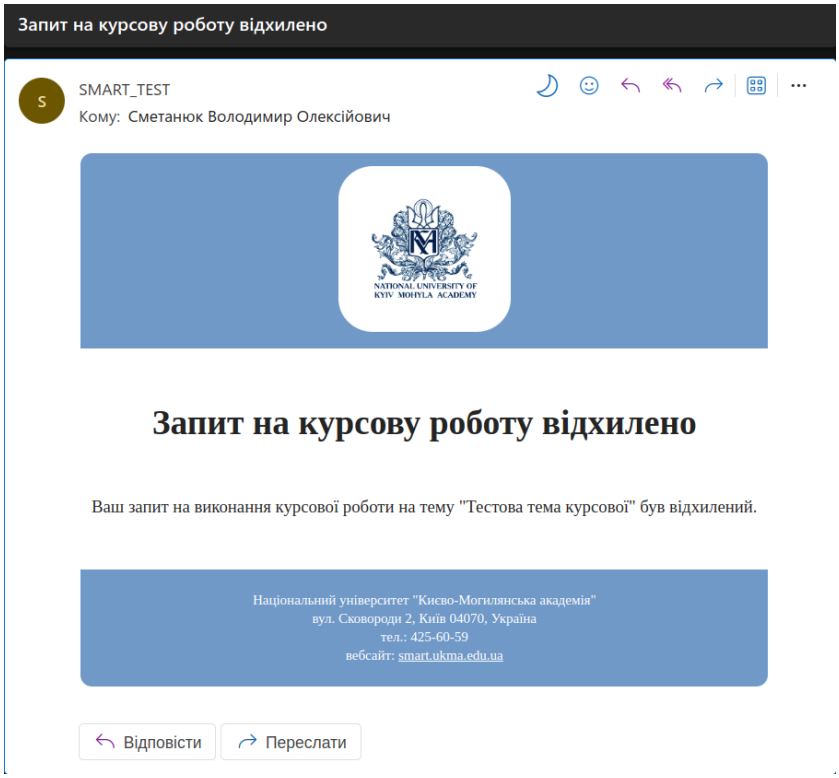

Рисунок 3.3 - Лист за шаблоном COURSE_WORK_REJECTED

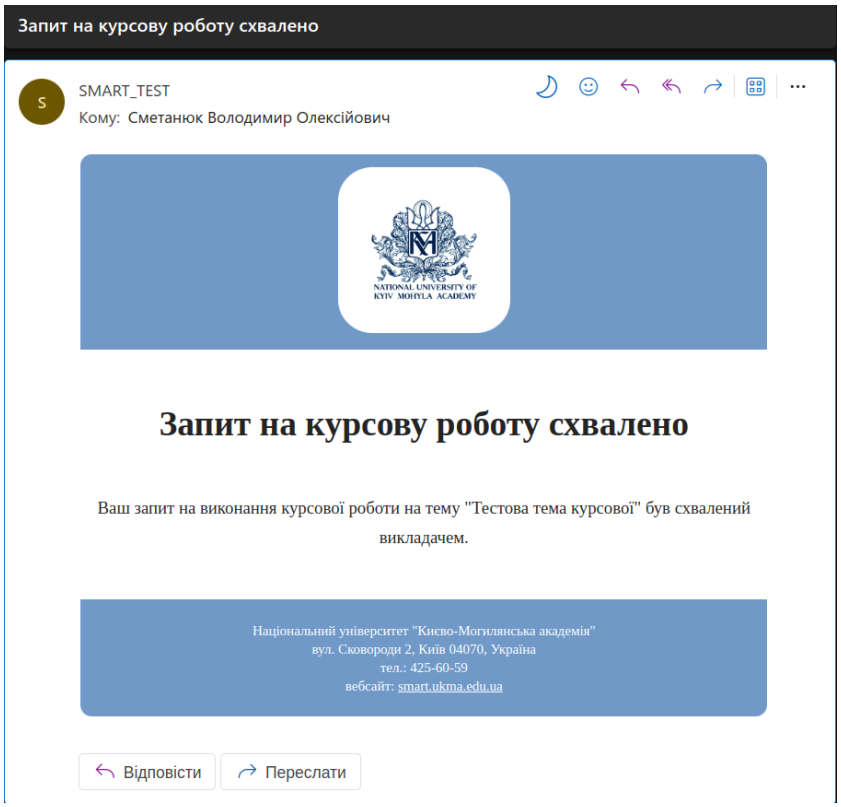

Рисунок 3.4 - Лист за шаблоном COURSE_WORK_APPROVED

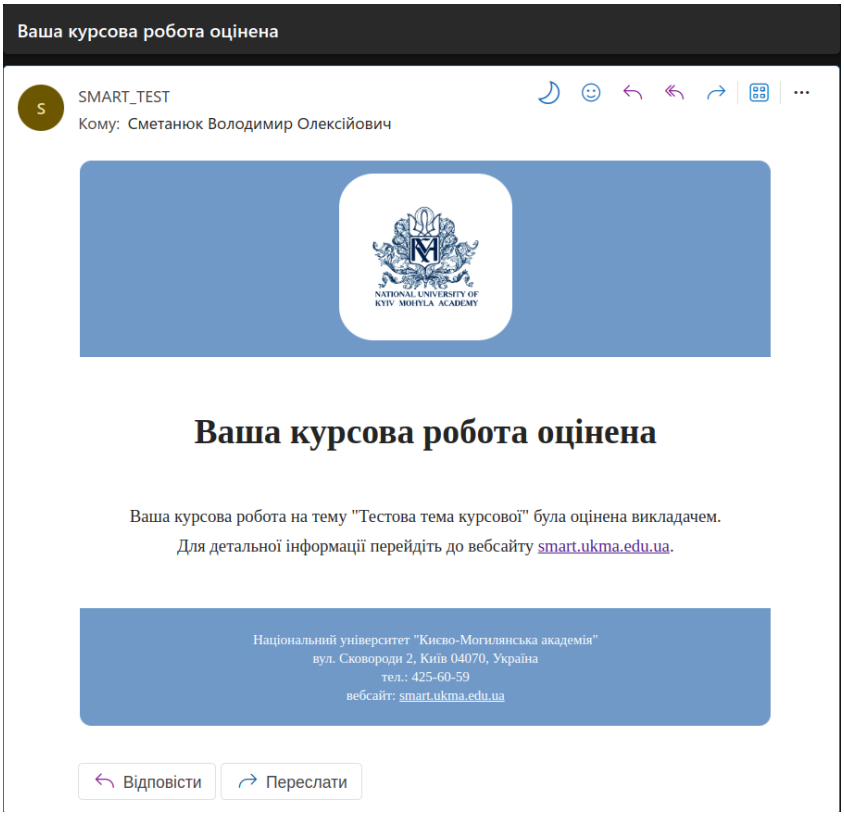

Рисунок 3.5 - Лист за шаблоном COURSE_WORK_EVALUATED

3.4 Висновки до розділу 3

У розділі продемонстровано створення шаблонів листів з використанням порад для найкращої сумісності та уникнення спам-фільтрів, наведених у розділі 1, протестовано створені шаблони за допомогою сервісу *Mailtrap*, та наведено результати у таблиці, продемонстровано розширення сервісу курсових робіт та результати роботи з використанням сервісу нотифікацій.

ВИСНОВКИ

У даній роботі було розглянуто основні архітектурні підходи до побудови застосунків, інструменти та фреймворки на мові програмування *Java* для роботи з базами даних, міграціями, веб-серверами, захистом застосунків, *SMTP* протоколом, *Firebase.* Також було досліджено деталі та принципи роботи спам-фільтрів, рекомендації щодо правильного налаштування процесів для роботи з Email та побудови листів та шаблонів.

Розглянуті технології та поради було застосовано для побудови сервісу нотифікацій. Фреймворк *Spring Boot* було використано як базовий інструмент для побудови сервісу. Було розроблено схему бази даних та використано міграції, побудовано інструменти для роботи з *Email, Firebase* та *Git* клієнтом для оновлення листів. Розроблено контролери для авторизації та роботи з *Firebase* сповіщеннями, а також обробник черги для надсилання листів через електронну пошту для рівня комунікації.

У результаті створено мікросервіс для роботи зі сповіщеннями та розроблено шаблони листів, пов’язані з курсовими роботами. Протестовано створені шаблони за допомогою сервісу *Mailtrap* та розширено сервіс курсових робіт використанням сервісу нотифікацій.

Для майбутнього покращення сервісу можна збільшити кількість та типи шаблонів листів та додати використання у інші сервіси системи *Smart.*

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Монолітна архітектура ПЗ. - QALight. QALight. URL: https://qalight.ua/baza-znaniy/shho-take-monolitna-arhitektura/.
2. Microservice architecture style - Azure Architecture Center. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices.
3. Microservice architecture style - Azure Architecture Center. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices.
4. Introduction to gRPC. gRPC. URL: https://grpc.io/docs/what-is-grpc/introduction/.
5. AMQP is an open Internet (or “wire”) Protocol standard for message-queuing communications. | AMQP. Home | AMQP. URL: https://www.amqp.org/product/overview.
6. Java Message Service (JMS). Oracle | Cloud Applications and Cloud Platform. URL: https://www.oracle.com/java/technologies/java-message-service.html.
7. Java HTTP Server. Package com.sun.net.httpserver Description. URL: https://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/index.html?com/sun/net/httpserver/package-summary.html.
8. Переваги та недоліки використання Spring Boot. JavaRush. URL: https://javarush.com/ua/groups/posts/uk.3380.kava-breyk-75-perevagi-ta-nedolki-vikoristannja-spring-boot-funkc-dlja-rjadkv-u-java.
9. What Is Java Spring Boot? | IBM. IBM in Deutschland, Österreich und der Schweiz. URL: https://www.ibm.com/topics/java-spring-boot.
10. Spring Data JPA :: Spring Data JPA. Spring | Home. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/.
11. Your relational data. Objectively. - Hibernate ORM. Hibernate. URL: https://hibernate.org/orm/.
12. Database Migrations with Flyway | Baeldung. Baeldung. URL: https://www.baeldung.com/database-migrations-with-flyway.

13. Servlet Authentication Architecture :: Spring Security. Spring | Home. URL: https://docs.spring.io/spring-security/reference/servlet/authentication/architecture.html.
14. Authorization Architecture :: Spring Security. Spring | Home. URL: https://docs.spring.io/spring-security/reference/servlet/authorization/architecture.html.
15. Method Security in GraalVM Native Image :: Spring Security. Spring | Home. URL: https://docs.spring.io/spring-security/reference/native-image/method-security.html.
16. Firebase | Google's Mobile and Web App Development Platform. Firebase. URL: https://firebase.google.com/.
17. Understand Firebase projects | Firebase Documentation. Firebase. URL: https://firebase.google.com/docs/projects/learn-more.
18. What is a Spam Filter & Spam Filtering?. Fortinet. URL: https://www.fortinet.com/resources/cyberglossary/spam-filters.
19. Forsey C. How to Bypass the Toughest Email Spam Filters [Infographic Checklist]. HubSpot Blog | Marketing, Sales, Agency, and Customer Success Content. URL: https://blog.hubspot.com/marketing/bypass-email-spam-filters.
20. SPF, DKIM & DMARC Explained: How To Set Them Up And Combat Fake Emails. Snovio Labs. URL: https://snov.io/blog/how-to-set-up-spf-dkim-dmarc/.