КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня бакалавра
за спеціальністю 122: Комп'ютерні науки
на тему:

Інформаційна система автоматизації поточної роботи та голосування «ВЧЕНА РАДА»

Виконав студент 4 курсу

Олексій БІЛИК

(підпис)

Науковий керівник:
професор

Ірина ВЕРГУНОВА

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Київ - 2022

РЕФЕРАТ

Обсяг роботи 39 сторінки, 16 ілюстрацій, 27 джерел посилання.

За темою дослідження опубліковано 2 тез виступів на конференціях.

ВЕБ-ДОДАТОК, ПЛАТФОРМА, ХМАРНІ ТЕХНОЛОГІЇ, СИСТЕМА ДЛЯ ГОЛОСУВАННЯ, ФОРМУВАННЯ ШАБЛОННИХ ДОКУМЕНТІВ, КЕРУВАННЯ КОРИСТУВАЧАМИ, CI/CD

Об’єктом дослідження є створення веб-платформи з функціоналом організації засідань за порядком денним, проведенням голосувань між учасниками та автоматизації формування відповідних протоколів.

Предметом роботи є програмний продукт для організації та проведення засідань вченої ради факультету.

Метою роботи є розробка додатку для організації та проведення онлайн та оффлайн засідань вченої ради факультету.

Інструменти розроблення: інтегроване середовище розробки Intellij IDEA 2021.3, мови програмування Java 11 та JavaScript. Під час розробки також використовувались послугу хмарного провайдера Amazon Web Services і наступні фреймворки:

- Spring – розробка бекенду;
- Vue.js – розробка веб-інтерфейсу користувача.

Результат роботи: проаналізовано алгоритм проведення засідання вченої ради, виділено процеси, які підлягають автоматизації, розроблено програмний продукт «Мобільна система для голосування “Вчена рада факультету”», що забезпечує виконання всіх робіт для функціонування ради.

Програмний продукт – система “Вчена рада факультету” – може використовуватись для проведення засідань вченої ради факультетів з голосуваннями, а також університету. При незначному розширені створена платформа може використовуватись іншими навчальними та трудовими колективами, наприклад для проведення засідань трудового колективу факультету, спецрад із захисту дисертаційних робіт.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкту дослідження. Використання інформаційних систем з можливістю проведення відеоконференцій, підтримки документообігу та голосування дозволяє працівникам ефективно виконувати робочі обов’язки віддалено, підвищує комфортність створення та опрацювання документів. На сьогоднішній день такі системи розроблюються та впроваджуються в усіх розвинутих країнах. Яскравими прикладами таких систем є парламентські системи електронного голосування, багаточисленні корпоративні системи електронного документообігу, системи автоматизації нарад та засідань із голосуваннями. Проте всі ці системи є або системами автоматизованого голосування, або системами електронного документообігу.

Актуальність роботи та підстави її виконання. З початком карантину людям довелось перенести все по можливості в онлайн. Не стали виключенням і колективи вчених рад, які проводять засідання з порядком денним, який включає питання з необхідністю проведення голосування серед учасників. У випадку факультету комп’ютерних наук та кібернетики питання зв’язку вирішується використанням Zoom, для голосування використовується сервіс Google Forms з подальшою ручною обробкою лічильною комісією, план засідання розповсюджується у текстовому документі, а запрошення надсилаються всім на пошту. Після проведення засідання секретарю необхідно сформувати протоколи засідань і лічильної комісії. Хоч вони і є шаблонними, але виконується це вручну. Виходить чимало процесів, які проходять на різних платформах/сервісах або можна автоматизувати.

Значно полегшила б організацію і проведення подібних засідань платформа, функціонал якої вирішував би усі ці питання, а саме:

- Планування засідання;
- Відправлення запрошень;
- Створення порядку денного та пропонування питань для розгляду;
- Проведення голосування;

- Генерація файлів протоколів за шаблоном(протокол лічільної комісії, протокол засідання);
- Сповіщення учасників;
- Керування ролями та посадами учасників;
- Архівація та зберігання файлових даних.

Більшість з цих процесів потребують реалізації не тільки на час карантинних обмежень, але й цифровізації для проведення у офлайн режимі, наприклад з використанням смартфонів.

Мета й завдання роботи. Метою роботи є розробка повнофункціональної системи для проведення засідань вченої ради факультету, що дозволяє проводити голосування, а також вести документообіг.

Для досягнення мети роботи були поставлені завдання:

- вивчення функціонування вченої ради факультету;
- аналіз можливості автоматизації основних процесів роботи ради та їх включення до системи;
- автоматизація створення протоколів засідання;
- створення автоматизованої системи реєстрації членів вченої ради, автоматизованої системи таємного голосування та обробки результатів системою в онлайн режимі.

Об’єкт, методи й засоби дослідження або розроблення. Об'єктом дослідження є процес автоматизації функціонування вченої ради факультету. Методами та засобами розроблення є методи: створення баз групового доступу; навігації за додатками, розробленими в межах системи; розробки та організації представлень, що дозволяють створювати візуальні представлення даних; методи маршрутизації електронних форм.

Можливі сфери застосування. Розроблена система, що впроваджується для роботи на факультеті комп’ютерних наук та кібернетики, може бути впровадженою й на інших факультетах університету.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ СИСТЕМ. ЇХ ПЛЮСИ ТА МІНУСИ

1.1. Сервіс Google Forms

Після переходу у онлайн режим вчена рада факультету почала проводити процес голосування з використанням сервісу Google Forms. Секретар засідання створює створює бланк для голосування і надсилає всім на пошту посилання. Після того, як всі проголосували результати обробляються вручну і оголошуються, а після засідання на основі отриманих результатів вручну формується протокол лічильної комісії по визначеному шаблоном файлу.

Плюси:

- Зручний сучасний інтерфейс;
- Простий у використанні;
- Відомий для учасників засідання.

Мінуси:

- Необхідно вручну створювати форми голосування на основі порядку денного і надсилати учасникам;
- Результати обробляються вручну;
- Відсутність формування результуючого протоколу.

1.2. Zoom

Питання зв’язку учасники вченої ради вирішують шляхом використання сервісу відеодзвінків Zoom. Після збільшення популярності вказаного сервісу був доданий функціонал “Опитування”, в якому можна виставити на голосування від одного до декількох запитань з відповідними варіантами відповідей.

Плюси:

- Достатньо зручний інтерфейс;
- Автоматичне оброблення результатів голосування.

Мінуси:

- Потрібно використовувати додатковий сервіс(пошту) для залучення учасників засідання до голосування;
- Необхідно вручну створювати форми голосування на основі порядку денного;
- Відсутність формування результуючого протоколу;
- Неможливість використання вказаного сервісу в режимі офлайн, оскільки функціонал “Опитування” є частиною відеодзвінка, який буде недоцільно використовувати під час очної зустрічі учасників.

1.3. Megapolis. Відеозасідання

У 2020 році компанія Intecracy Group [1] презентувала комерційне рішення для проведення відеоконференцій. Система спрямована для використання під час зустріч, наслідком яких мають бути юридично значущі дії. Наприклад, це збори Ради директорів (чи засновників) у бізнес-структурах, обговорення нормативно-правових актів чи рішень у державному секторі тощо. Зазвичай такі зустрічі вимагають особистої присутності, що в умовах карантину є великою проблемою. Також можлива інтеграція з системою документообігу Megapolis.DocNet [2].

У зв’язку з комерційною складовою проекту перевірити роботу платформи та отримати досвід для порівнянн не вдалося, але з переглянутих презентацій можна зробити висновок про сучасність UX/UI інтерфейсу та автоматизацію більшості процесів, які в загальному характерні організації і проведенню засідань

робочих колективів. Слід зауважити про спрямованість більше у юридичну сторону. Відкритим залишається питання про проведення засідань у режимі оффлайн.

РОЗДІЛ 2
ЗАГАЛЬНИЙ ФУНКЦІОНАЛ ПЛАТФОРМИ

2.1. Автентифікація та авторизація

При розробці платформи було вирішено відмовитись від стандартного підходу з HttpSession і використовувати технологію JWT [3] (додаток А).

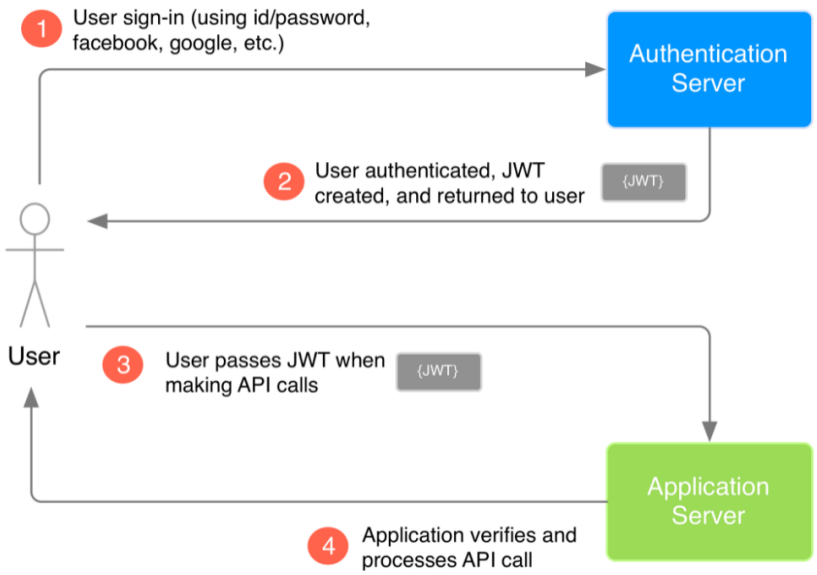


Рисунок 2.1 – Алгоритм отримання і використання JWT

Використовується JWT (рис. 2.1) для перевірки автентифікації користувача наступним чином:

1. Користувач робить запит автентифікації за допомогою пари e-mail/пароль;
2. Сервіс автентифікації перевіряє отримані данні, в разі успіху створює короткотривалий Access Token [4] та довготривалий Refresh Token [5] і надсилає їх користувачу;
3. Коли користувач робить запит на API платформи, він добавляє до нього раніше отриманий Access Token (токен доступу).

4. При обробці запиту з доданим токеном доступу сервер перевіряє валідність отриманого токену і може авторизувати користувача без додаткових запитів до бази даних;
5. Після закінчення часу валідності токена доступу користувач робить запит на API платформи, додаючи до нього Refresh Token. У відповідь сервер створює новий токен доступу. Таким чином для оновлення сесії роботи з платформою користувачу не потрібно знову вводити пару e-mail/пароль.

Слід зазначити, що платформа є закритою, тобто зареєструватися можна тільки отримавши відповідне запрошення. Запросити нового учасника можуть головуючий вченої ради та секретар. Для цього заповнюється необхідна інформація про користувача, включаючи його e-mail (рис. 2.2). Після заповнення всіх полів на вказану пошту находить лист-запрошення з посиланням для реєстрації.

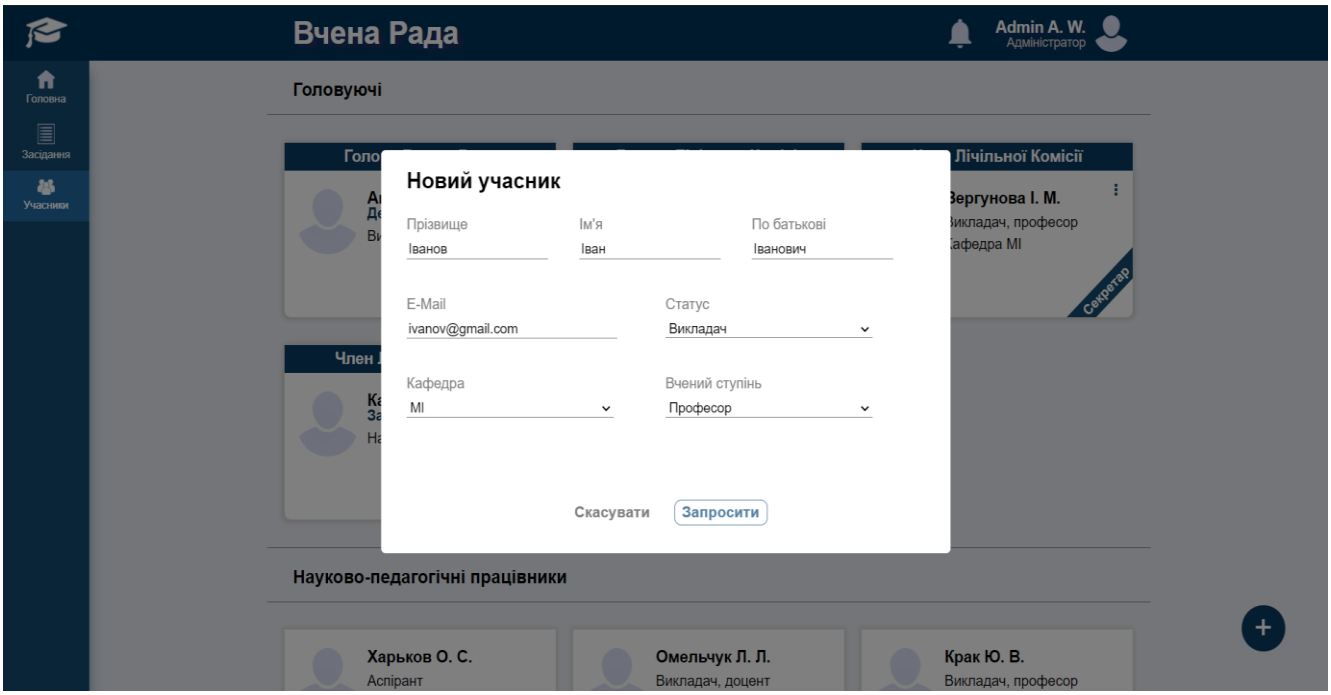


Рисунок 2.2 – Запрошення нового учасника

Також доступна можливість відновлення паролю. Для цього на пошту користувача надсилається посилання з кодом.

Сама ж реалізація виконана з використанням Spring Security. Фреймворк Spring виконує обробку запитів у вигляді конвеєру на якому запит проходить певні фільтри та “хендлери”. При підключені Spring Security підключається вже з “коробки” додатковий фільтр, який передбачає використання сесії користувача. Також Spring Security надає можливість додавати свої фільтри у згаданий вище конвеєр або замінювати вже існуючі. Використовуючи цю можливість було додано JWTAuthenticationFilter (додаток Б), який наслідує AbstractAuthenticationProcessingFilter та JWTAuthorizationFilter, який наслідує BasicAuthorizationFilter. Для перевірки пари логін і пароль(пароль зберігається в закодованому вигляді у базі даних) в SpringSecurity було зареєстровано додатковий BaseAuthenticationProvider, який реалізує інтерфейс AuthenticationProvider.

2.2. Збереження даних та взаємодія з Базою Даних

Для збереження даних було вирішено використовувати об'єктно-реляційну систему керування базами даних PostgreSQL. Таке рішення є найбільш популярним і конкурентним під час розробки додатків на Java на рівні з MySQL.

Для доступу до бази даних використовується *spring-starter-jpa*. Це обгортка від Spring над JPA(Java Persistence API) - стандартизований інтерфейс для Java ORM фреймворків. В якості реалізації цього інтерфейсу використовується Hibernate. Такий підхід дозволяє працювати з даним як з об’єктами. Також полегшується режим розробки додатку, оскільки переважна кількість запитів генерується автоматично без явного вказання, що зменшує велику кількість шаблонного коду, який необхідний, щоб коректно створити транзакцію, додати запит з параметрами, отриманий результат перевести у об’єкти.

JPA використовує підхід з “репозиторіями”. Він передбачає, що запити не описуються явно, а створюються інтефейси і вже в назвах методів цих інтерфейсів міститься вся інформація, яка необхідна, щоб згенерувати запит мовою HQL(Hibernate Query Language). HQL – це об’єктно орієнтована мова запитів, яка

дуже схожа на SQL. Основна відмінність між HQL і SQL полягає у тому, що SQL працює з таблицями у базі даних та їх стовпцями, а HQL – з об’єктами та їх полями(атрибутами класу). Виконання HQL запитів виконується через їх трансляцію у SQL.

Наприклад, щоб знайти усіх користувачів за його id створюється інтерфейс

UserRepository

і додається метод

User findByIdAndActive(UUID id, Boolean active)

Під час запуску додатка буде створено проксі клас, який буде реалізовувати вказаний інтерфейс. В результаті ми отримаємо HQL запит

```
SELECT u
FROM User u
WHERE u.id = ?1 AND u.is_active = ?2
```

Також можна створювати запити з поєднанням декількох таблиць, наприклад знаходження всіх активних користувачів, які належать до певної організації

List<User> findAllByActiveAndOrganizationName Boolean active, String name)

отримаємо HQL запит

```
SELECT u
FROM User u
LEFT OUTER JOIN Organization org on org.id = u.organization_id
WHERE u.is_active = ?1 AND ord.name = ?2
```

Хоч такий підхід і є дуже зручним, проте все таки потрібно перевіряти, які запити генерує Hibernate, переглядаючи журнали логування. Інколи все таки краще писати запити вручну, щоб бути впевненими в їхній оптимальності.

Також використовується патерн LazyLoading, що покращує швидкість виконання запитів. Під час роботи з ORM завантаження даних можна розділити на два типи: eager і lazy.

Eager Loading — це шаблон проектування, в якому ініціалізація даних відбувається відразу, під час першого запиту до бази даних.

Lazy Loading — це шаблон проектування, який використовується, щоб відкласти ініціалізацію об’єкта до тих пір, доки цей об’єкт не буде потрібний бізнес логіці додатку.

З точки зору запитів відмінність буде полягати у ключовому слові синтаксису мови HQL “Fetch”

LEFT JOIN FETCH Organization org on org.id = u.organization_id

Саме це є вказівкою для Hibernate, що з об’єднаною таблицею будуть виконані операції не тільки на рівні бази даних, а що ці запис також потрібно завантажити у вигляді об’єктів да додати їх до об’єкту *User*. В іншому випадку замість об’єктів класу *Organization* будуть створені проксі, а при першому використанні цих проксі об’єктів буде виконано додатковий запит до бази даних, в якому вже і буде завантажено інформацію про *Organization.*

2.3. Ролі користувачів в системі

В системі використовуються наступні ролі користувачів:

- Учасник вченої ради;
- Член Лічільної комісії;
- Голова лічільної комісії;
- Голова Вченої ради;
- Секретар Вченої ради;
- Адміністратор.

У вказаному списку ролі записані за порядком зростання рівня доступу. В даному випадку секретар має більше повноважень, оскільки основний функціонал

є модераторським, що за звичайного засідання і виконує секретар. За рівнем доступу визначається можливість використовувати той чи інший функціонал платформи. Наприклад: мінімальний рівень доступу для створення нового засідання - секретар Вченої ради. Переглянути актуальні ролі членів вченої ради можна на сторінці “Учасники”. Також на цій сторінці можна керувати ролями користувачів, використовуючи відповідне випадаюче меню (за наявності рівня доступу) (рис. 2.3). Якщо користувача було назначено/знято з посади – він отримає відповідне сповіщення у центрі сповіщень (рис. 2.4).

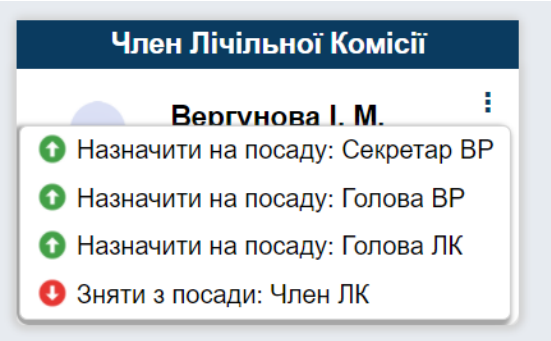


Рисунок 2.3 – Керування ролями користувача

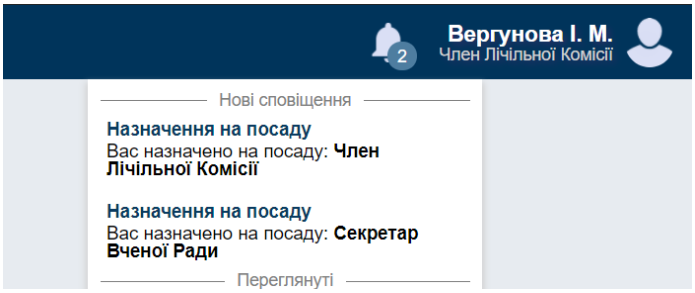


Рисунок 2.4 –Центр сповіщень користувача

2.4. Організації

Після певного часу тестового використання платформи виникла потреба у розділенні користувачів на деякі групи таким чином, щоб вони не пересікалися між собою під час керування платформою. Відповідно було додано додатковий шар ієрархії у доменній моделі платформи: організації

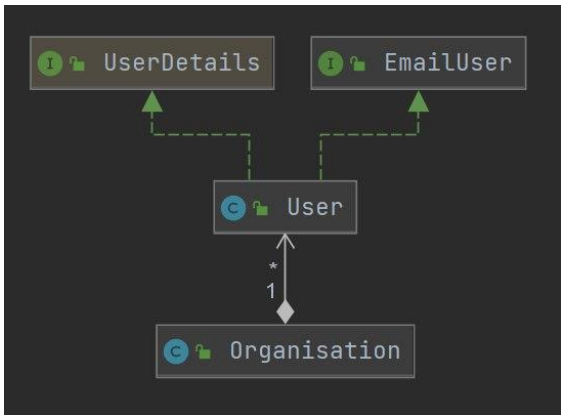


Рисунок 2.5 – UML діаграма додаткового шару доменної моделі

Нова організація реєструється в системі адміністратором і надсилається запрошення секретарю, який вже може додавати інших користувачів після активації акаунту. Слід зауважити, що можливості супер користувачів, таких як Секретар або Голова Вченої ради, активні тільки в межах організації, до якої вони належать. Також користувач може отримати доступу тільки до протоколів організації, до якої він належить. Таке рішення реалізовано на основі перевірки політики та прав доступу, тобто без фізичного розділення, що є не повністю безбечно. Варіант більш безпечного рішення буде розглянуто у пункті 5 розділу 4 цієї роботи.

2.5. Сторінка “Засідання”

На сторонці “Засідання” доступний для перегляду весь список засідань вченої ради (рис. 2.5). Також можна отримати коротку необхідну інформацію про саме засідання, включаючи дату та час проведення, посилання на відеоконференцію та статус.

Загалом у засідання може бути один з чотирьох статусів:

- Модерація (заповнення необхідних данних і попереднє формування порядку денного; доступно для перегляду тільки головуючим і секретарю);
- Заплановано;
- Онлайн (засідання зараз відбувається);

- Завершено.

Під час **Модерації** секретар вводить та корегує загальну інформацію: номер засідання, дата та час проведення, посилання на відеоконференцію. Також створює розділи порядку денного та додає питання для розгляду.

При зміні статусу засідання з **Модерація** на **Заплановано** усім членам вченої ради надсилається на пошту запрошення з короткою необхідною інформацією і посиланням на засідання на платформі. Саме засідання стає доступним для перегляду всіх учасників і кожен може запропонувати власне питання до порядку денного для розгляду.

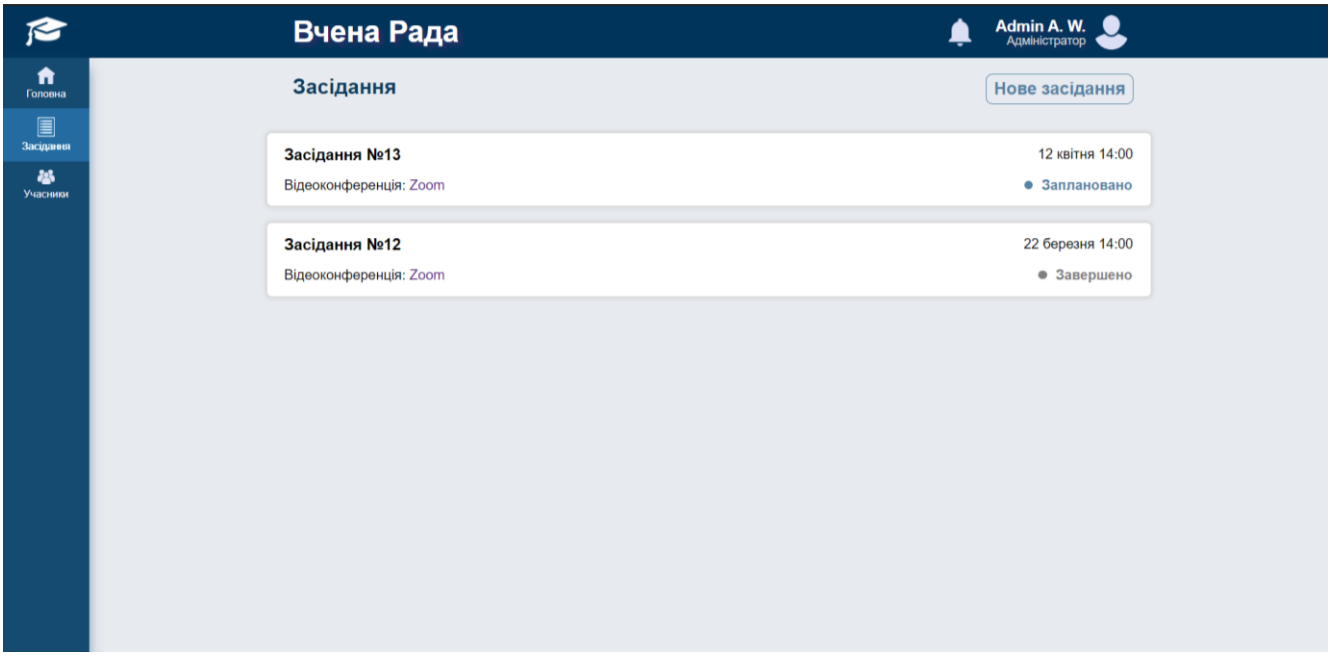


Рисунок 2.5 – Сторінка перегляду всіх засідань

РОЗДІЛ 3

ОРГАНІЗАЦІЯ І ПРОВЕДЕННЯ ЗАСІДАННЯ

3.1. Формування порядку денного

При переході на сторінку перегляду засідання змінюється протокол спілкування з сервером на WebSocket [6]. Користувач підписується на канал відповідного засідання і може бачити в режимі реального часу зміни, які вносять інші користувачі і вносити свої корегування.

На сторінці перегляду засідання можна відредагувати загальну інформації. Також на цій сторінці відбувається формування порядку денного: створюються розділи та питання для розгляду (рис. 3.1). До кожного питання, за потреби, можна додати файли(звіти, презентації, заяви тощо), які зможуть завантажити інші користувачі. Якщо розгляд створеного питання передбачає голосування серед учасників засідання, то до питання додається опція “Голосування” і в життєвий цикл розгляду питання буде включено процес проведення голосування.

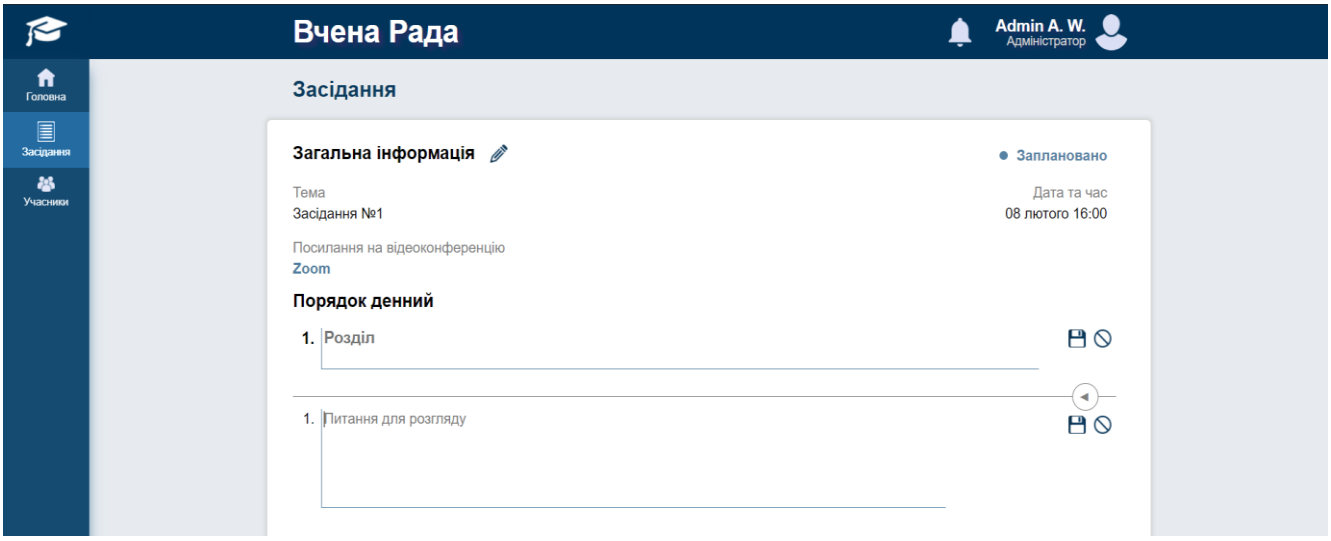


Рисунок 3.1 – Створення та редагування питання порядку денного

Зазвичай текст питання містить багато громізтких словосполучень, які часто повторюються, накштал повної назви університету чи факультету, або ж

згадування когось з учасників ради з повним записом ініціалів, вченого ступеню та посади. Тому було розроблену просто реалізацію автодоповнення (рис 3.2). Алгоритм пропонує варіанти продовження враховуючи початок фрази або абревіатури. Тобто щоб швидко написати “факультет комп’ютерних наук та кібернетик“ достатньо почати писати слово “факультет” і вибрати потрібний варіант, або ж зразу написати абревіатуру “фкнк” і зробити заміну. За необхності можна змінити відмінок фрази (стрілки вліво та вправоу меню вибору автодоповнення).

Такий функціонал текстового поля є достить зручним і корисним, коли секретареві під час розгляду питання необхідно також коротко записувати, що було сказано під час виступу, ухвалено або вирішено. Для цього у користувача з ролью Секретар при роботі з порядком денним у кожного питання є додаткові текстові поля з автодоповненням: Виступили і Ухвалили. Ці данні також будуть використані при автоматичному формуванні протоколу засідання.

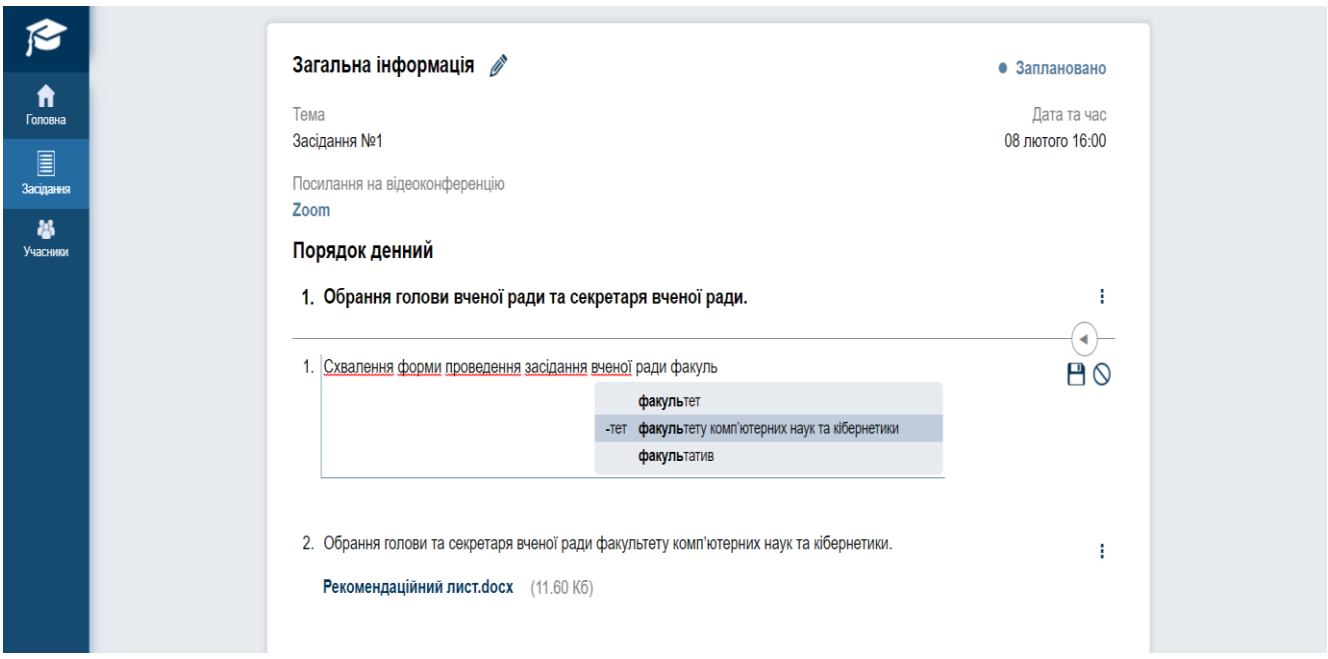


Рисунок 3.2 – Створення та редагування питання порядку денного

3.2. Проведення засідання у форматі онлайн

У час, зазначений як початок засідання, секретар розпочинає засідання і статус змінюється на “онлайн”, про що свідчить надпис зеленого кольору у правому верхньому кутку.

Після зміни статусу активується реєстрація користувачів на засідання (рис. 3.3). Реєстрація відбувається автоматично при відвідуванні сторінки порядку денного. Отримана інформація про зареєстрованих учасників також буде використовуватись під час формування протоколів.

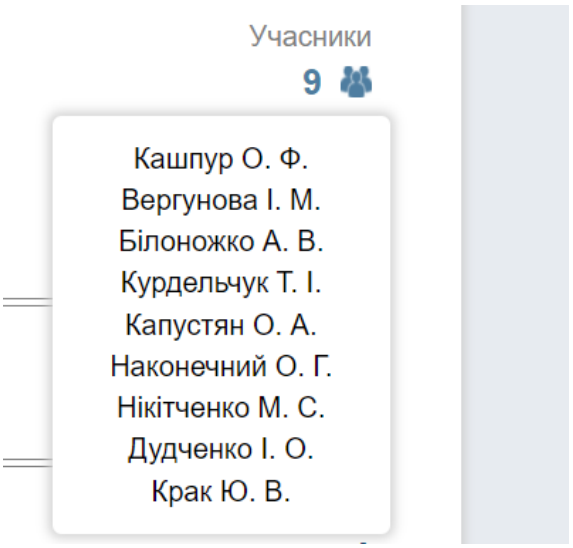


Рисунок 3.3 – Список зареєстрованих учасників засідання

Після початку розгляду питання з’являється відповідна іконка(синє око), яка позначає активне питання і активний розділ (рис. 3.4). У секретаря ж додаються поля з автодоповненням, які необхідно заповнити для протоколу. Вже розглянуті питаня позначаються зеленою галкою.

У питань з опцією голосування справа додається відповідна інформація про перебіг процесу голосування (рис. 3.5) та його результати (рис. 3.6).

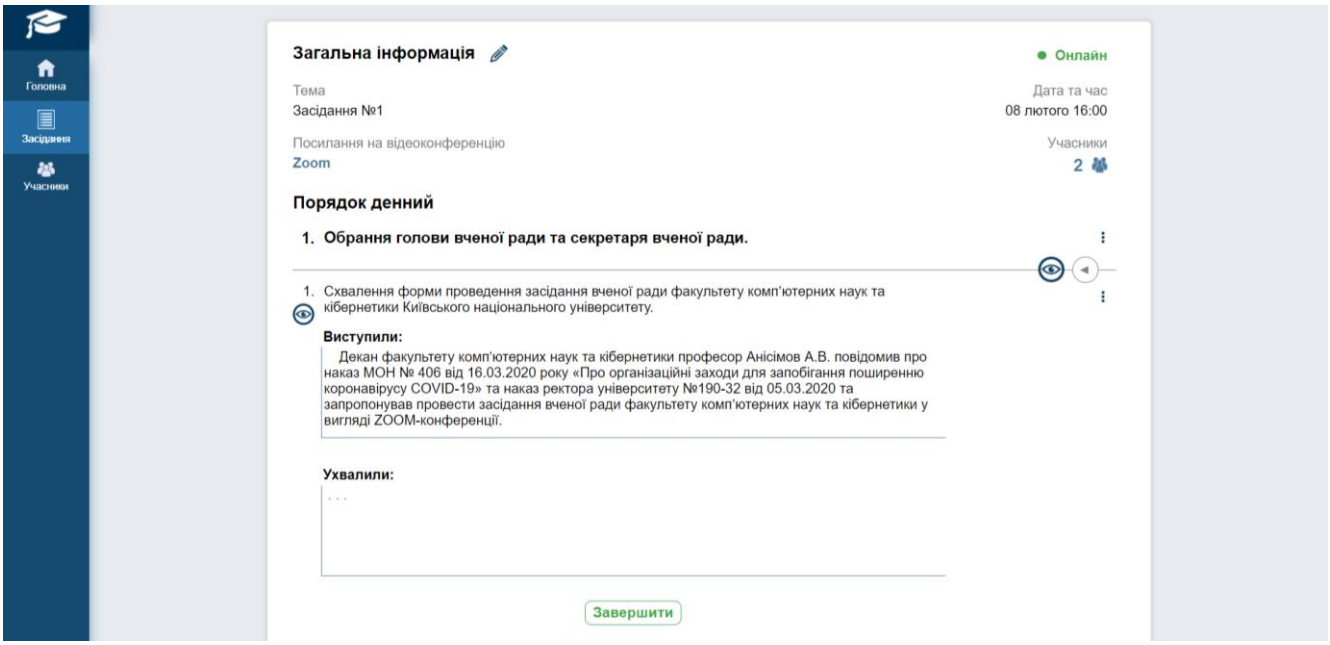


Рисунок 3.4 – Розгляд питання, інтерфейс секретаря

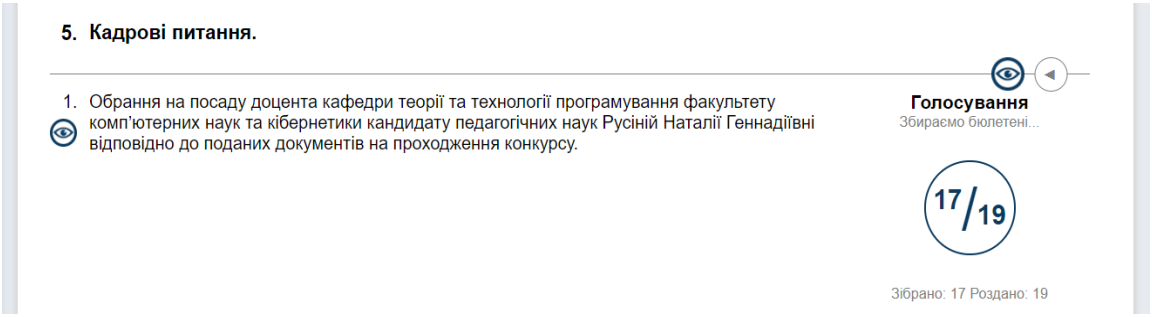


Рисунок 3.5 – Розгляд питання з голосування, інтерфейс учасника

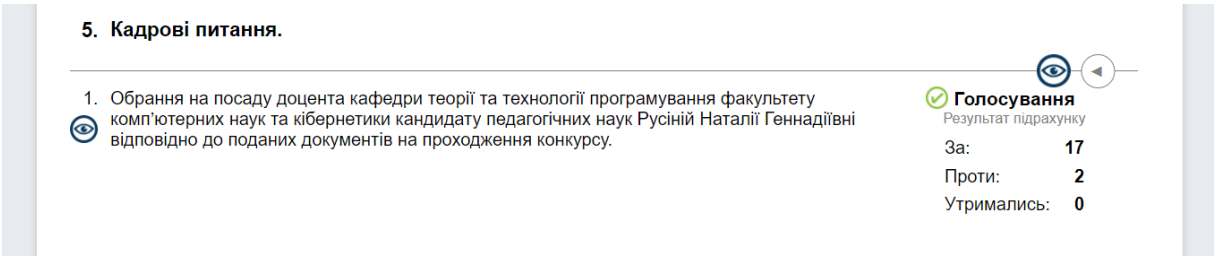


Рисунок 3.6 – Розгляд питання з голосування, результати, інтерфейс учасника

Коли ж заслухали необхідну інформацію по питанню і секретар розпочав процес голосування у кожного учасника автоматично відкривається вікно голосування (рис. 3.7).

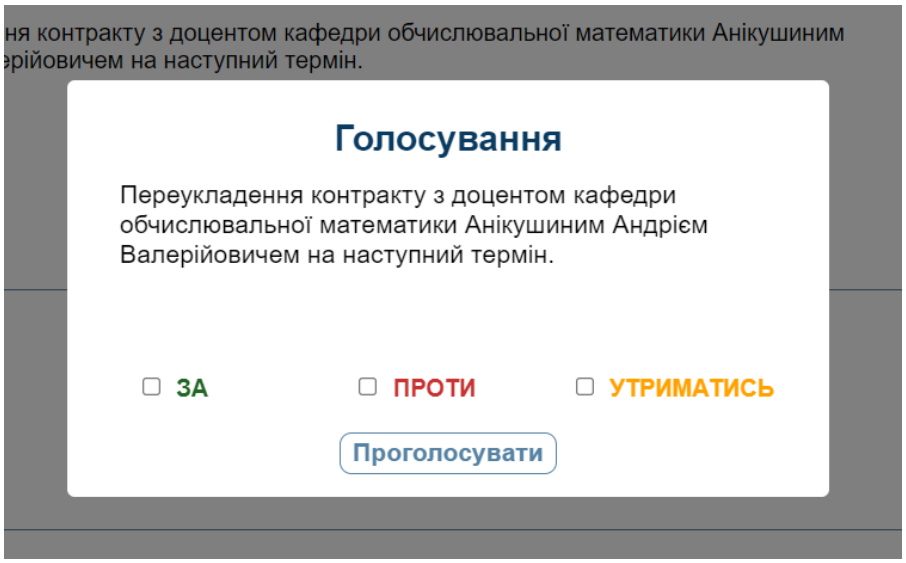


Рисунок 3.7 – Інтерфейс голосування

Голосування закінчується автоматично після збору всіх бюлетнів, або достроково за вимогою секретаря. Після цього результати голосування затверджуються або відбувається повторне голосування(за потреби).

Слід зауважити, що не можна розпочати розгляд іншого питання, не закінчивши попереднє. Це стосується і голосувань. Необхідно пройти весь ланцюжок станів питання та голосування.

3.3. Протоколи засідання

Після завершення формуються необхідні протоколи. У результаті має бути один файл з протоколом засідання, в якому записана загальна інформація за шаблоном, а також перелік розділів і питань, які були розглянуті під час засідання. Також до кожного питання у протоколі необхідно додати короткий текстовий запис виступу та що у результаті ухвалили. Якщо для розгляду питання проводилось голосування потрібно додати інформацію про результат голосування у вказаному форматі. Під час генерації такого протоколу всі дані автоматично підтягуються з порядку денного засідання і не потребує ніякої додаткової ручної обробки. Для кожного питання, яке передбачало проведення голосування необхідно сформувати

окремий файл – протокол лічільної комісії з відповідним порядковим номером. Усі файли формуються у форматі .docx.

Отримані протоколи може завантажити та переглянути кожен користувач платформи:

- протокол засідання – внизу сторінки порядку денного
- протокол лічільної комісії – під результатами голосування відповідного питання

Для формування файлів за шаблоном потрібне використання певного template engine [7]. У випадку роботи з файлами типу .doc це може бути запропоноване рішення від Aspose [8]. Приклад синтаксису:

```
<<[s.getName()]>> says: "<<[s.getMessage()]>>."
```

Важливим плюсом є те, що шаблон записується відразу у .doc файл. Оскільки синтаксис шаблонізатора доволі простий, то за потреби можна легко відкоригувати шаблон у випадку зміни норм заповнення протоколу. Внести корегування міг би і секретар без втручання адміністрації платформи. Проте дане рішення не надає необхідної гнучкості у редагуванні стилів наповнення файлу, що не дозволяє відтворити необхідний шаблон.

Наступним рішенням можна спробувати спочатку генерувати потрібний файл у вигляді .html з використанням шаблонізатора Thymeleaf [9]. Такий підхід дозволяє забезпечити необхідний формат і стилізацію протоколів завдяки гнучкості вказаного шаблонізатора. Тоді отриманий .html можна конвертувати у .doc файл, використовуючи бібліотеку docx4j [10]. Проте після конвертації з’являються проблеми зі стилізацією і створений файл потребує додаткової обробки, чого не допускає наша умова повної автоматизації цього процесу.

Відповідно було вирішено написати власну реалізацію схожого шаблонізатора з мінімальним потрібним функціоналом. Підхід з синтаксисом запозичено у вже знайомого Thymeleaf. Як і в першому варіанті, код шаблону

пишеться у .doc файл, але вже зразу з потрібною стилізацією, яка зберігається під час обробки шаблонізатором. Приклад синтаксису:

```
Результати голосування:
“За” _____${voting.agree}_____
“Проти” _____${voting.disagree}_____
“Утримались” _${voting.abstained}_____
```

3.4. Таємне голосування

Щоб система голосування відповідала стандартним вимогам до систем таємного голосування [11] необхідно, щоб виконувались наступні критерії:

- ніхто, крім виборця, не повинен знати про свій вибір
- голосувати можуть лише зареєстровані учасники (вони можуть проголосувати лише один раз)
- рішення виборця ніким не може бути змінено

Кожен зареєстрований на засіданні член Вченої ради повинен бути впевнений, що його голос зараховано, він може змінити свою думку та зробити свій новий вибір у короткий термін. Крім того, є аутентифікація оператора, і можна дізнатися, хто брав участь у голосуванні, а хто ні. Обслуговування системи не повинно вимагати великих ресурсів і має бути відмовостійким у разі технічних несправностей, ненавмисних і злочинних дій.

Серед існуючих протоколів заслуговує на увагу протокол двох агентів. Основна ідея цього протоколу – замінити одне виборче агентство двома, щоб вони контролювали один одного.

Нехай матимемо валідатора, в обов’язки якого входить підготовка списків і допуск чи недопуск учасника до голосування. Він створює набір ідентифікаційних тегів (відповідає кількості зареєстрованих учасників) і надсилає по одному тегу по безпечному каналу кожному зареєстрованому учаснику без зберігання інформації про те, кому належить цей тег.

Існує багато різних варіантів, заснованих на протоколі двох агентів. Наприклад, з деякими ускладненнями це такі протоколи:

- протокол Fujioka-Okamoto-Ohta [12], який ускладнює стандартний протокол, використовує маскування шифрування і частково вирішує проблему змови між двома агенціями. Він дозволяє перевірити, чи є документ автентичним і підписаний авторизованим користувачем, але не розкриває, які дані він містить;
- протокол Sensus [12] (модифікація протоколу Fujioka-Okamoto-Ohta), характерною особливістю якого є те, що при отриманні зашифрованого повідомлення від учасника, який проголосував, воно негайно додається до списку опублікованих результатів, а підписаний бюлетень повертається до учасника як квитанцію. Це дає можливість не чекати, поки всі інші учасники проголосують, а завершити голосування (в той же час це стає додатковим доказом того, що певний учасник брав участь у голосуванні);
- протокол He-Su [12] , в якому ключ зареєстрованого учасника підписується сліпим підписом (але використовується зазначена хеш-функція і метод сліпого шифрування повинен бути визначений заздалегідь). Це також дозволяє виборцям змінити свою думку до кінця голосування та ще більше виключає можливість змови між реєстратором та агентством. Але це вимагає досить багато ресурсів.

3.5. Оффлайн засідання

З певним часом проведення засідань вченої ради знову повернеться до формату очних зустрічей. Проте можна буде продовжити використання платформи і в такому режимі. Тоді потреба у сервісі відеозв’язку відпадає, а для участі у засіданні кожному учаснику достатньо мати лише під рукою смартфон або планшет. Верстка веб-клієнта вже адаптована для використання на вище зазначених пристроях (рис. 3.8). За потреби можна також зробити показ сторінки з порядком денним за допомогою проектора на екрані.

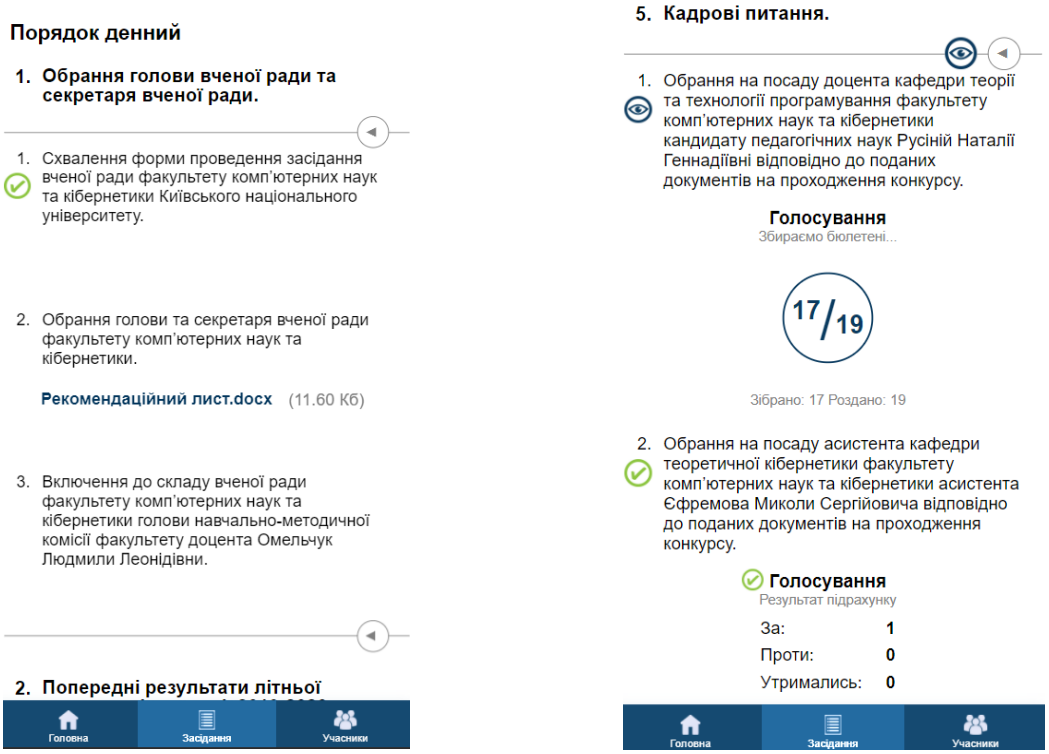


Рисунок 3.8 – Мобільна версія

РОЗДІЛ 4

CI/CD ЯК ЧАСТИНА ПРОЦЕСУ РОЗРОБКИ ВЕБ ПЛАТФОРМИ ТА ВИКОРИСТАННЯ ХМАРНИХ ТЕХНОЛОГІЙ

4.1. Неперервна інтеграція та неперервне розгортання

Невід'ємною частиною процесу розробки програмного забезпечення, а саме веб платформи у даному випадку, є CI/CD [13] – це комбінація неперервної інтеграції (continuous integration) та неперервного розгортання (continuous delivery або continuous deployment) програмного забезпечення. CI/CD поєднує в собі розробку, тестування та розгортання застосунку(рис. 4.1).

Такий підхід дозволяє:

- мінімізувати ймовірність відтворення вже знайдених дефектів, оскільки на початку кожної ітерації відбувається запуск юніт [14] та інтеграційних тестів [15], формується звіт за результатами виконання, перехід до наступного кроку відбувається тільки за умови успішного виконання усіх тестів;
- автоматизувати значну кількість процесів, що однозначно економить час і захищає від типових механічних помилок, які можуть бути допущені, наприклад, під час розгортання нової версії мікросервісу.

Також неявним плюсом є те, що за необхідності можна створювати нові середовища (dev, stage, prod) з мінімальними труднощами, оскільки наявність CI/CD передбачає відсутність прямого вказання ресурсів і наявність змінних, а також використання певних шаблонів. Таким чином середовища відрізняються між собою тільки певними id/auth даними, необхідними для ідентифікації/авторизації, та певними префіксами у назвах ресурсів (dev, test, stage, prod).

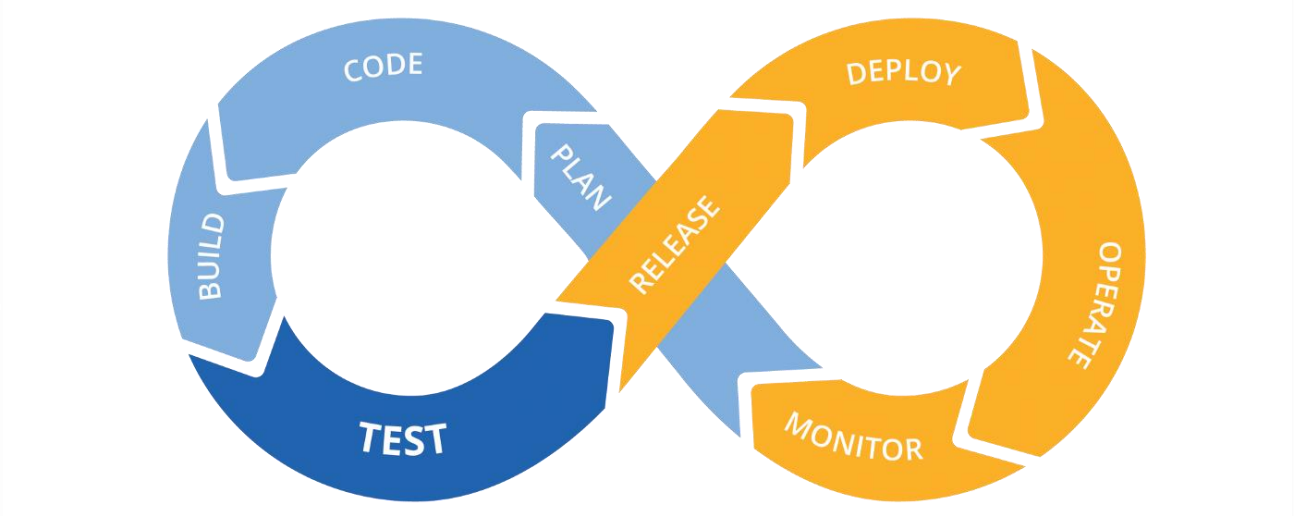


Рисунок 4.1 – Етапи неперервної інтеграції та неперервного розгортання

Зазвичай використовується наступна схема:

- для роботи над певною задачею створюється нова гілка у системі контролю версій;
- після кожного нового коміта у цю гілку(або групи комітів) спрацьовує певний тригер на підключеній платформі CI/CD (буде розглянуто далі) і виконується скрипт на віртуальній машині, у процесі виконання якого запускаються тести і створюється звіт з результатами їх виконання;
- під час злиття гілки, в якій велась робота над задачею, з головною гілкою репозиторію спрацьовує інший тригер на підключеній платформі CI/CD, який передбачає створення збірки (build) проекту та початку процесу розгортання нової версії (для dev або test оточення);
- створення релізу (доставка та розгортання нових версій на stage та prod оточення) зазвичай налаштовуються через злиття основної гілки проекту з гілками stage/prod, або ж як окрема задача, яка запускається вручну з вказанням певних параметрів.

Зараз існує багато інструментів та платформ для CI/CD, які користуються популярністю серед розробників:

1. Jenkins;
2. Circle CI;
3. TeamCity;
4. GitLab;
5. GitHub Actions.

Загалом усі запропоновані рішення схожі у основних аспектах та мають передбачене безкоштовне використання, правда з певними лімітами в плані витраченого часу і сховища. Наприклад, безкоштовна версія GitHub Actions надає 500Mb сховища і 2 000 хвилин на виконання протягом одного місяця (з підпискою GitHub Pro ці обмеження будуть збільшені до 1Gb і 3 000 хвилин відповідно).

Якщо код проекту зберігається на GitHub, то відповідно найпростішим рішенням буде GitHub Actions, оскільки це рішення є нативним та не вимагає додаткової інтеграції з іншими системами чи платформами, що полегшує процес підключення і подальшого підтримання.

Також певні інструменти пропонують і провайдери хмарних технологій. Наприклад, AWS (Amazon Web Services) надають послугу під назвою CodeBuild [16], яка дозволяє запускати тести, упаковувати додатки, створювати Docker images та публікувати їх у AWS ECR (Elastic Container Registry) звідки вже в подальшому можна буде використати створений образ при розгортанні нової версії мікросервісу. Google Cloud в свою чергу надає повноцінну serverless CI/CD платформу [17], яка дозволяє проганяти тести, створювати білди та розгортати нові версії.

4.2. Інфраструктура платформи та процес розгортання

Спочатку для розгортання додатку використовувався один з найпростіших підходів: орендувався VPS(віртуальний виділений сервер) з фіксованою платою за

місяць користування, на сервері встановлювався вебсервер Nginx [18], який приймай вхідний трафік і розподіляв його на три категорії:

- Frontend;
- Backend API;
- Backend WebSockets.

Для першої категорії Nginx сам обробляв запит і виконував роздачу статичних файлів(.html, .css, .js та зображення). Такий варіант став можливий завдяки тому, що фронтенд частина була розроблена у вигляді SPA(Single-page application) [19], мабуть найпопулярніше рішення у сучасному світі фронтенду, реалізується завдяки таким фреймворкам як Vue.js, React або ж Angular.

У випадку потрапляння до другої категорії Nginx виконував роль Проксі-сервера та перенаправляв запити на вебсервер Tomcat.

Рисунок 4.2 – Nginx у ролі Проксі-сервера

Tomcat містив контейнер з самим додатком, який вже і обробляв запити. Не велику особливість мала третя категорія, оскільки для коректної роботи Nginx також виконував обробку заголовків запитів, а після цього вже передав запитав на обробку у Tomcat. Такий підхід дозволяє звільнити Tomcat від обробки легких, але багаточисельних запитів щодо статичних ресурсів і працювати лише, фактично,

над запитами, які стосуються бізнес логіки платформи. Також Nginx має вже готове рішення щодо кешування вже “з коробки”.

4.3. Використання хмарних технологій

Оскільки дана платформа не має характеру рівномірного навантаження було вирішено перейти на використання хмарних технологій. Як провайдера було обрано AWS (Amazon Web Services). Відповідно виникли певні зміни у інфраструктурі платформи.

Фронтенд частина платформи після збірки завантажується разом з статичними ресурсами у бакет на AWS S3. Доступ до фронтенд ресурсів отримується за допомогою AWS CloudFront [20] – мережі доставки контенту з мінімальною затримкою, яку надає Amazon Web Services для роботи з Single-page application.

Для розгортання Бекенд частини використовується рішення AWS ECS(Elastic Container Service) [21]. Оскільки такий підхід передбачає запуск додатка у контейнері, тобто кожний новий запуск відбувається у новому контейнері, а після закінчення виконання ресурси очищуються, то для мікросервісу спочатку необхідно створити ECS Task Definition [22]. Фактично це запис, який містить в собі набір параметрів, необхідних для роботи мікросервісу, інформацію про те де міститься збірка або Docker image [23] самого мікросервісу, а також налаштовані характеристики контейнера, як от значення виділеної пам’яті, або образ операційної системи.

Також AWS надає можливість використовувати Бази Даних як сервіс, а не розгортати їх разом з додатком, тому для даного додатку була використана PostgreSQL, яку надає AWS окремим компонентом.

Оскільки виникає потреба безпечної передачі даних, необхідних для підключення мікросерсіву до БД використовується AWS Secret Manager, який дозволяє безпечно зберігати пару логін/пароль і передати їх в середині мережі клауду. Мікросервіс під час запуску спочатку робить запит у AWS Secret Manager, а після цього вже підключається до БД, отримавши необхідну інформацію для доступу. Такий підхід дозволяє уникнути передачі пари логін/пароль і адресу хосту БД у вигляді параметрів під час запуску інстанса, що є не дуже безпечним варіантом.

Розгортання у AWS ECS також накладає обмеження на роботу з файловою системою, оскільки фактично всі створені файли будуть “тимчасовими” і в майбутньому до них уже не вдасться отримати доступ. Оскільки робота платформи передбачає зберігання файлових протоколів, то необхідно використовувати AWS S3 – сервіс-сховище даних, тобто так званий файловий хостинг. Для роботи з S3 необхідна також додаткова таблиця у базі даних, щоб мати можливість зворотного зв’язку і отримання файлів. Взаємодія з такими компонентами як S3 та SecretManager відбувається з використанням наданого AWS SDK [24].

Також важливим моментом під час роботи з AWS, який варто вказати, є CloudFormation [25][26]. CloudFormation – це сервіс AWS, який надає можливість створити інфраструктуру в Amazon Web Services, описавши її тестом у форматі YAML або JSON. У файлі описуються ресурси, які потрібно створити і залежності між ними. Також важливим пунктом під час створення інфраструктури на базі AWS є поняття політики (policies) і прав доступу. Якщо налаштовувати їх всі вручну, то виникає проблема при необхідності створення нового оточення(або оновлення існуючого), адже тоді прийдеться шукати всі політики, які були створені і переносити їх на нове оточення(наприклад з dev на prod). CloudFormation також вирішує цю проблему.

Таким чином завдяки CloudFormation відпадає і необхідність в документації, оскільки все вже задокументовано в коді. Під час створення нового стеку або оновлення для вже існуючого усі ресурси(починаючи від простих прав доступу закінчуючи базами даних та конфігурацією API Gateway) оброблюються автоматично. А основне – AWS сам знає, в якій послідовності краще створювати та видаляти ресурси. Завдяки цьому уникаються такі ситуацій, коли видаляється частина інфраструктури, а про його залежності забули. CloudFormation прослідкує за всім сам. У випадку ж невдалого оновлення відбувається відкочування інфраструктури до останнього стабільного стану.

4.4. Реалізація CI/CD

Після переходу на AWS отримуємо три репозиторії:

- senatus-cloudformation;
- senatus-ui;
- senatus-api.

Кожен репозиторій має свій алгоритм CI/CD:

senatus-cloudformation:

- авторизуватись у AWS;
- завантажити оновлений шаблон на S3 у відповідний бакет;
- виконати оновлення стеку cloudformation;

senatus-ui:

- встановити залежності, необхідні для створення білда;
- запустити тести, використовуючи vue-cli, створити, звіт;
  у разі успіху перейти на наступний крок;
- створити білд, використовуючи vue-cli;

- авторизуватись у AWS;
- завантажити білд разом з статичними ресурсами на S3 у відповідний бакет, з якого виконує роздачу CloudFront;

senatus-api:

- встановити залежності, необхідні для створення білда;
- запустити тести, використовуючи maven, створити, звіт;
  у разі успіху перейти на наступний крок;
- створити білд, використовуючи maven;
- створити новий docker image;
- авторизуватись у AWS;
- додати створений образ у регістр AWS (Elastic Container Registry);
- оновити TaskDefinition, вказавши у ньому тег створеного образу (короткий хеш крайнього коміта);
- оновити сервіс у ECS Cluster, вказавши id нового TaskDefinition (розпочнется процес розгортання нової версії мікросервісу на основі створеного раніше образу).

З наявних платформ та найбільш популярних рішень для CI/CD (перелік було наведено у першому пункті цього розділу) було вирішено використовувати GitHub Actions як нативне рішення, а також через відсутність потреби в інтеграції з іншими сервісами.

Своє флоу (послідовності виконання команд) описується у файлах формату .yml і додаються у корінь проекту у папку .github/workflows. Вже існує багато готових заготовок скриптів починаючи з самого простого як от переключення між гілками чи отримання тегів так більш складних – авторизація на AWS чи оновлення TaskDefinition. Такі заготовки можна вільно перевикористовувати у своїх флоу. Також можна створювати і свої заготовки, зберігаючи їх в окремому репозиторії, що дозволяє зменшити кількість шаблонного коду, якщо платформа складається,

наприклад, з декількох мікросервісів, які мають однаковий процес тестування/збірки/розгортання.

4.5. Архітектура Multi-Tenancy

Після переходу на хмарну платформу Amazon Web Services стає можливою реалізація архітектури Multi-Tenancy [27]. Це архітектура, в якій один екземпляр програмного додатка обслуговує кількох тенантів. Кожен клієнт називається тенантом. Тенантам може бути надано можливість налаштовувати деякі частини програми, наприклад колір інтерфейсу користувача (UI) або бізнес-правила(але вони не можуть налаштовувати код програми).

В архітектурі з багатьма тенантами кілька екземплярів програми працюють у спільному середовищі. Ця архітектура може працювати, оскільки кожен тенант інтегрований фізично, але логічно розділений; це означає, що один екземпляр програмного забезпечення працюватиме на одному сервері, а потім обслуговуватиме кількох тетантів. Таким чином, додаток в архітектурі з багатьма клієнтами може спільно використовувати виділений екземпляр конфігурацій, даних, керування користувачами та інші властивості.

Додатки з кількома орендарями можуть використовувати одних і тих же користувачів, дисплеї, правила - хоча користувачі можуть певною мірою налаштувати їх - і схеми баз даних, які орендарі також можуть налаштувати.

З використанням архітектури Multi-Tenancy можна розмежувати організації не тільки на рівні політики або прав доступу, але й на фізичному рівні, а також додати можливість змінювати бізнес-логіку в залежності від потреб кожної конкретної організації, що виводить платформу на зовсім інший рівень.

ВИСНОВКИ

У даній роботі було проаналізовано алгоритм проведення засідання вченої ради факультету, виділено процеси, які підлягають автоматизації, автоматизовано ключові етапи організації та проведення засідань вченої ради факультету і, як результат, розроблено веб-додаток.

Під час роботи були вдало вирішені такі проблеми:

- Автоматизація роботи з поштою та сповіщеннями;
- Взаємодія колективу користувачів у режимі real time;
- Формування файлів протоколу на основі прийнятих шаблонів та даних проведення засідання.

Також розглянуто використання хмарних технологій та CI/CD під час розробки, а створені моделі флоу CI/CD можуть бути перевикористаними у інших проектах.

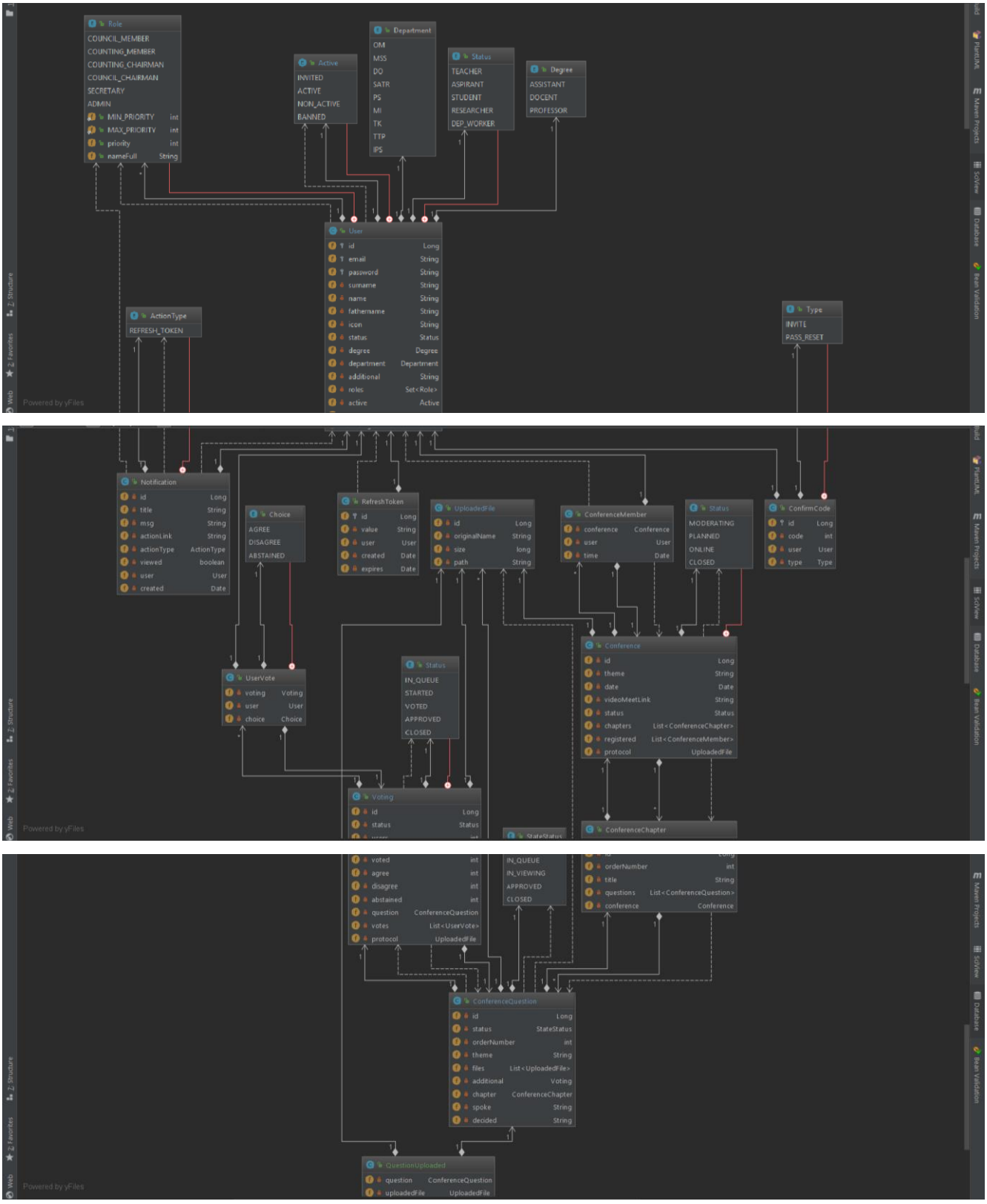
Розроблений програмний продукт – система для голосування “Вчена рада факультету” – може бути розширена для проведення відповідних засідань вченої ради на інших факультетах, а також засідань вченої ради університету.

У випадку використання системи можна відмовитись від звичайним чином збереження протоколів у .doc форматі. Перегляд протоколів буде відбуватися відразу на сторінці платформи. Такий підхід полегшить роботу з витягами та їх створенням. При необхідності використання протоколу як додатку достатньо лише прикріпити посилання на відповідну сторінку платформи, а не цілий файл.

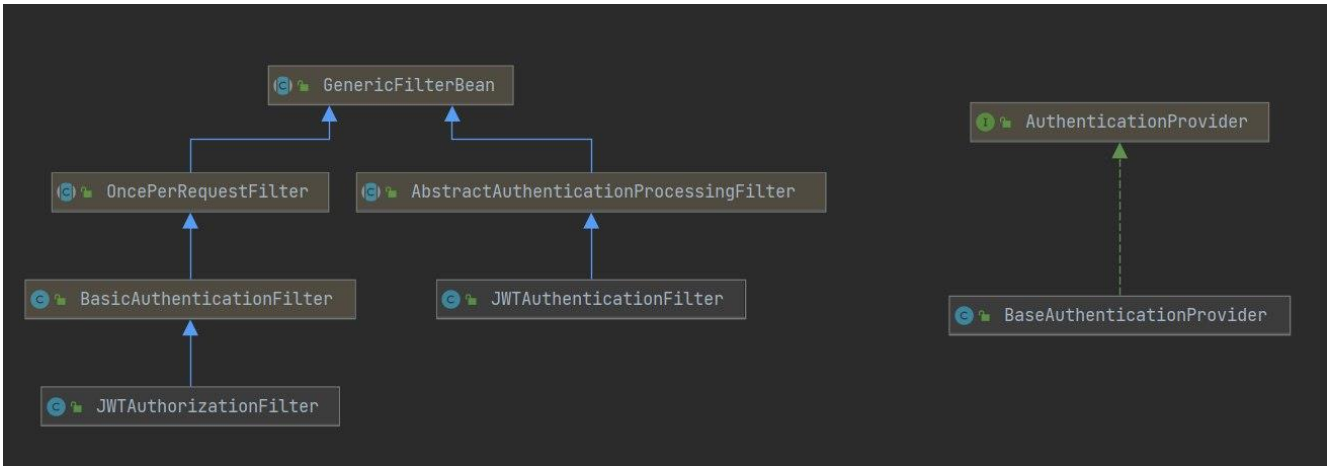
При певному доопрацюванні створена платформа може використовуватись іншими навчальними та трудовими колективами, наприклад для проведення засідань трудового колективу факультету. Можливо варто додати систему плагінів, або імплементувати архітектуру Multi-Tenancy, оскільки засідання хоч в основному проходять і за однаковим шаблоном, але є моменти, які потребують персонального підходу і реалізації для кожної окремої організації.

Створений продукт вирішує проблеми проведення онлайн засідань, але також адаптований і для оффлайн версії під час очних зустрічей. Тоді для участі у засіданні кожному учаснику достатньо мати лише під рукою смартфон або планшет.

ДОДАТОК А. UML діаграма доменної моделі



ДОДАТОК Б. Security layer



СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Intecracy Group презентує новітню систему відеозасідань. – 2020. – Ел. Ресурс. Режим доступу: https://softline.org.ua/news/intecracy-group-prezentuie-novitniu-systemu-videozasidan.html.
2. Система електронного документообігу Megapolis.DotNet. Ел. Ресурс. Режим доступу: https://inbase.com.ua/ua/soft/megapolis-docnet.html.
3. Офіційна сторінка JWT. Ел. Ресурс. Режим доступу: https://jwt.io.
4. Офіційна документація OAuth. Access Token. Ел. Ресурс. Режим доступу: https://auth0.com/docs/tokens/access-tokens.
5. Офіційна документація OAuth. Refresh Token. Ел. Ресурс. Режим доступу: https://auth0.com/docs/tokens/refresh-tokens.
6. Протокол WebSocket. Ел. Ресурс. Режим доступу: https://uk.wikipedia.org/wiki/WebSocket.
7. Template engine. Ел. Ресурс. Режим доступу: https://en.wikipedia.org/wiki/Template_processor.
8. Aspose Template Engine demo. Ел. Ресурс. Режим доступу: https://blog.aspose.com/2020/01/14/generate-word-documents-from-templates-dynamically-using-java/.
9. Офіційна документація Thymeleaf. Ел. Ресурс. Режим доступу: https://www.thymeleaf.org/documentation.html.
10. Офіційний сайт docx4j. Ел. Ресурс. Режим доступу: https://www.docx4java.org/trac/docx4j.
11. D.R. Patel, Information security: Theory and Practice, PHI Learning, Pvt. Ltd., 2008.
12. Протоколи Fujioka-Okamoto-Ohta, Sensus, He-Su. Ел. Ресурс. Режим доступу: https://uk.wikipedia.org/wiki/Протоколи_таємного_голосування.
13. What is CI/CD . Ел. Ресурс. Режим доступу: https://www.redhat.com/en/topics/devops/what-is-ci-cd.

14. Модульне тестування (unit testing). Ел. Ресурс. Режим доступу: https://uk.wikipedia.org/wiki/Модульне_тестування.
15. Інтеграційне тестування (integration testing). Ел. Ресурс. Режим доступу: https://uk.wikipedia.org/wiki/ Інтеграційне_тестування.
16. Amazon Web Services CodeBuild. Ел. Ресурс. Режим доступу: https://aws.amazon.com/en/codebuild/.
17. Google Cloud: Cloud Build. Ел. Ресурс. Режим доступу: https://cloud.google.com/build.
18. Офіційна сторінка Nginx. Ел. Ресурс. Режим доступу: https://nginx.org/en.
19. Single-page application. Ел. Ресурс. Режим доступу: https://en.wikipedia.org/wiki/Single-page_application.
20. Amazon Cloud Front. Ел. Ресурс. Режим доступу: https://aws.amazon.com/en/cloudfront.
21. Amazon Elastic Container Service. Ел. Ресурс. Режим доступу: https://aws.amazon.com/en/cloudfront.
22. Amazon ECS task definitions. Ел. Ресурс. Режим доступу: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definitions.html.
23. Docker. Docker Images. Ел. Ресурс. Режим доступу: https://docs.docker.com/engine/reference/commandline/images.
24. AWS SDK для Java. Ел. Ресурс. Режим доступу: https://aws.amazon.com/en/sdk-for-java/.
25. AWS CloudFormation. Ел. Ресурс. Режим доступу: https://aws.amazon.com/en/cloudformation/.
26. AWS CloudFormation: найкращі практики та способи їх використання. Ел. Ресурс. Режим доступу: https://www.intellias.ua/blog/cloud-formation-practices.
27. What is multi-tenancy. Ел. Ресурс. Режим доступу: https://www.techtarget.com/whatis/definition/multi-tenancy.