МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

ФАКУЛЬТЕТ *ІНФОРМАТИКИ*
КАФЕДРА *ІНФОРМАТИКИ*

ДИПЛОМНА РОБОТА

Галузь знань *12 – Інформаційні технології*

спеціальність *121 – Інженерія програмного забезпечення*

на тему

Веб-сервіс для проведення електронних опитувань та голосувань у корпоративному середовищі

Виконав студент 4 курсу:
Валентий Ярослав Олегович

Науковий керівник:
старший викладач
Кобзар Олег Олегович

Київ - 2024

Індивідуальне завдання

ЗАТВЕРДЖУЮ

Асистент,

_____О. О. Кобзар
(підпис)

„____”_____ 20__ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

студенту *_Валентому Ярославу Олеговичу_* факультету *_ інформатики _4_* курсу *__БП «Інженерія програмного забезпечення»*

ТЕМА: Веб-сервіс проведення електронних опитувань та голосувань у корпоративному середовищі

Зміст текстової частини до дипломної роботи:

Зміст
Анотація
Вступ
1. Актуальність використання електронної системи виборів та проблеми, пов'язані з організацією виборів.
1.1. Важливість використання електронних засобів для підтримання життєдіяльності органів студентського самоврядування
1.2. Недоліки попередньої електронної системи виборів та інших тимчасових рішень
1.3. Вивчення поточних проблем Студентської Виборчої Комісії
1.4. Необхідність синхронізації баз даних університету та електронної системи виборів
2. Створення структури проєкту та налаштування робочого середовища
2.1. Використання стратегії Monorepo для побудови структури проєкту
2.2. Інструменти для роботи з Monorepo
2.3. Налаштування робочого середовища

Календарний план

№ п/п	Назва етапу дипломної роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	18.09.2022	
2.	Комунікація з Студентською Виборчою Комісією.	24.11.2022	
3.	Проектування електронної системи виборів.	03.02.2023	
4.	Реалізація програмної частини електронної системи виборів.	15.11.2023	
5.	Проектування сервісу для синхронізації баз даних.	04.12.2023	
6.	Реалізація програмної частини сервісу для синхронізації баз даних.	27.01.2024	
7.	Представлення реалізованої системи представникам Студентської Виборчої Комісії та зібрання відгуків.	24.04.2024	
8.	Створення презентації.	10.05.2024	

№ п/п	Назва етапу дипломної роботи	Термін виконання етапу	Примітка
9.	Коригування роботи.	12.05.2024	
10.	Захист дипломної роботи.	29.05.2024	

Студент _____

Керівник _____

“_____” _____

Зміст

Анотація

У даній роботі розглядається дослідження існуючих проблем у процесах виборів до органів студентського самоврядування на прикладі Національного Університету “Києво-Могилянська академія”, пошуки їх рішень та практична реалізація системи електронних виборів, що б дозволила проводити голосування серед учасників навчального процесу та була універсальною для різних застосувань, а також задовільняла поточні потреби Студентської Виборчої Комісії, з метою спрощення та покращення цього процесу. Окрім того робота досліджує проблему синхронізації баз даних університету та інших сервісів, якими користуються студенти, та оглядає реалізацію рішення цієї проблеми. Для виконання прикладних задач цієї роботи використовуються сучасні методи та інструменти розробки веб-додатків.

Вступ

Існування активного студентського самоврядування в університеті є ознакою його життя та процвітання, адже саме діяльні студенти в майбутньому будуть впливати на майбутнє нашої країни. Проведення демократичних та прозорих виборів до органів самоврядування є однією з головних умов його розвитку та збереження, оскільки це відображає ставлення студентства до цінності права та справедливості.

Використання електронних засобів для проведення голосувань є зручним вирішенням багатьох проблем, що пов’язані з викликами сьогодення, адже не всі учасники навчального процесу можуть бути присутні фізично в університеті, щоб проголосувати. Окрім цього, використання електронних засобів допомагає автоматизувати та полегшити процеси підрахунку голосів, доступу до інформації про голосування та поліпшити прозорість процесу. Студентська Виборча Комісія мала змогу випробувати можливості системи, що знаходиться за адресою https://vote.ukma.edu.ua/, проте дане рішення не задовольняє актуальні потреби та не є універсальною. Тому дослідження існуючих проблем у процесах виборів до органів самоврядування та створення універсальної електронної системи для проведення голосувань є актуальними.

Метою даної роботи є створення електронної системи виборів, що відповідатиме запитам Студентської Виборчої Комісії, буде універсальною та використовуватиме сучасні технології, завдяки чому підтримка подібної системи та удосконалення не складатиме труднощів у майбутньому. Також важливою частиною роботи є дослідження проблем синхронізації баз даних університету та сервісів, що використовують для поліпшення навчального процесу, створення рішення та його імплементація.

1. Актуальність використання електронної системи виборів та проблеми, пов'язані з організацією виборів.

1.1. Важливість використання електронних засобів для підтримання життєдіяльності органів студентського самоврядування

Життя нашого університету завжди тісно було пов’язане зі самоврядуванням студентів, адже це є підтвердженням того, що університет живе та розвивається. Для повноцінного функціонування студентських організацій є потреба раз на певний період часу змінювати їх склад. Через цифровізацію багатьох сфер діяльності університету, організатори та учасники цього процесу вже давно використовують електронні засоби для проведення виборів нових членів студентських організацій.

Перехід на електронні засоби для проведення виборів має багато переваг, адже це дає можливість не тільки автоматизувати цей процес, але ще й надати доступ до цих голосувань людям, які не можуть фізично брати в них участь. Це є дуже актуальним, враховуючи минулу глобальну пандемію, а також триваючу повномасштабну російсько-українську війну. Через неї багато хто зі студентів або виїхав з України, або не має змоги знаходитися в Києві і фізично відвідувати університет. Тому використання електронних засобів для проведення таких заходів є життєво необхідним для повноцінного та справедливого життя студентського самоврядування в нашому університеті. Також потрібно згадати, що подібна система дозволить значно спростити не тільки підрахунок голосів, адже він буде відбуватися автоматично, але ще й додасть гнучкості у збереженні та відображенні даних щодо результатів проведених виборів. Це дасть змогу швидко і будь-де фізично дізнатися результати виборів, якщо такі вже завершені,

оголосити проведення нових або промодерувати потрібні аспекти.

Зважаючи на вище вказані тези, можна зробити висновок, що розробка та використання подібних систем є дуже актуальними в наш час, оскільки вони вирішують багато проблем, а також спрощують цю процедуру, роблять більш доступною та відкритою.

1.2. Недоліки попередньої електронної системи виборів та інших тимчасових рішень

Для того, щоб розробити електронну систему виборів, потрібно дослідити вже існуючі рішення, які проблеми вони мають та що можна в них покращити. Частково проблематика проведення виборів в корпоративному середовищі вже розглядалася в минулій роботі, але наразі є неповною та не було імплементоване рішення, що б покращило процеси [1].

Раніше впровадження подібної системи вже виконувалося [2]. В неї була можливість створити голосування, в якому адміністратор має змогу додати певних студентів, що існують в базі даних сервісу, яких можуть обрати студенти, обрати час початку та закінчення голосування і, якщо час голосування вичерпався – система показує таблицю з кількістю голосів за кожного кандидата. Ця система мала ряд недоліків, які було б важко усунути просто доробивши її. На це є декілька причин:

- Система розроблювалася на базі ядра іншого сервісу, що створювався для системи оцінювання предметів, через що має обмеження і проблеми, виправлення яких є неможливим через особливості архітектури ядра;
- Підтримку цієї системи зупинили через важкість роботи зі стеком технологій, на якій вона була написана, а також, підкріплюючи тезу

наведену вище, через неможливість повністю покрити ті потреби виборчого процесу, які були наявні. Це й було основною причиною чому цей сервіс перестав бути актуальним, адже студентські організації просто не могли використовувати цю систему для закриття власних потреб у виборчому процесі.

Варто також вказати на недоліки цієї системи, які ускладнювали чи унеможливлювали її використання для деяких виборчих процесів у студентському самоврядуванні.

По-перше, дана система не мала змогу гнучко обмежити доступ до голосування за певними параметрами студента, наприклад, спеціальністю, роком навчання, тощо. Для правильного функціонування навчального процесу кожен рік є необхідність проводити вибори старост. Вони проходять у кожної спеціальності кожного курсу. Минула система не мала змоги проводити подібні голосування через відсутність гнучкої системи фільтрації що ще більше підтверджує, що вимога до системи надавати можливість обмежувати доступ до голосування певним студентам є актуальною і необхідно вирішити це питання.

По-друге, стара система дає змогу обрати студентам тільки одного кандидата в одному голосуванні. Проте деякі виборчі процеси потребують гнучкості, наприклад, щоб студент міг обрати двох чи трьох делегатів або від одного до п’яти, чи інші комбінації. Через цей недолік, деякі студентські організації не могли провести вибори в цій системі, оскільки вони потребували такої опції, а система не могла її надати.

Доповнюючи проблеми вказані вище, варто додати, що стек технологій, на якому була написана ця система для голосувань, застарів. Все менше людей їх використовують, наприклад, PHP активно втрачає свою популярність. В щорічному опитування Stack Overflow PHP з 30% показник використання серед

розробників у 2017 році впав до 18% у 2023 [3]. І ця тенденція спостерігається і щодо інших технологій та фреймворків, що використовуються в цій системі. Окрім цього, сайти, що були створені за допомогою PHP, складно підтримувати. Були випадки, коли перенесення інших сервісів університету із власних серверів на хмарні платформи створювало суттєві труднощі, адже сервіси та середовища для них, що розроблялися з допомогою PHP набагато складніше конфігурувати. З цього можна зробити висновок, що з часом стає все важче знаходити розробників, які мають можливість підтримувати таку систему. У зв'язку з цим, можна також зробити висновок, що для створення нової системи необхідно використовувати сучасні технології, що користуються попитом на ринку праці.

1.3. Вивчення поточних проблем Студентської Виборчої Комісії

Для того, щоб прийняти рішення щодо побудови архітектури для нової електронної системи виборів, необхідно було спочатку дослідити, а які саме потреби зараз має Студентська Виборча Комісія, яка є органом студентського самоврядування, який відповідає за проведення виборів до Студентської Колегії, Конгрегації Студентів, старостату, тощо. Для цього було вирішено зв'язатися з представниками Студентської Виборчої Комісії для з'ясування поточних проблем та викликів, які перед ними стоять.

Під час спілкування з представниками було висунуто ряд вимог до подібної системи, а також побажання, які, на їхню думку, варто реалізувати. З цих вимог можна зробити такий список:

- Система має надавати змогу адміністратору обмежувати вибірку студентів, які мають доступ до голосування. Наприклад, якщо проводяться вибори до Конгрегації Студентів, то до складу має обратися по делегату від кожного факультету кожного курсу. Тобто має бути надана можливість створити голосування до якого матиме

доступ тільки, наприклад, студенти першого курсу Факультету Інформатики. Така гнучкість має бути надана щодо факультету, ступеню здобувача освіти, спеціальності та курсу. Наприклад, для виборів старост має бути надана можливість вказати факультет, спеціальність та курс водночас;

- Адміністратор повинен мати можливість надати учасникам голосування змогу надавати більше ніж один голос за одне голосування. Така потреба з’являється, оскільки до певних органів учасники голосування повинні мати змогу проголосувати за декілька кандидатів;
- Учасники голосування мають мати вільний доступ до результатів проведеного голосування, дізнатися кількість голосів за кожного кандидата, тощо;
- Було висловлено побажання також імплементувати функціонал, який би дозволив проводити вибори до рад гуртожитків, використовуючи інформацію, що зберігається в базах даних електронної системи поселення в гуртожитки DMS;
- Минула система електронних виборів мала деякі проблеми з синхронізацією бази даних сервісу та системою “Оптіма”, через що інколи траплялися ситуації, що до голосування мали доступ студенти, які вже не навчаються або діючі студенти не мали змоги проголосувати. Через це виникла необхідність дослідити це питання та запропонувати вирішення проблеми;
- Представники Студентської Виборчої Комісії висловили своє побажання щодо розробки додаткового сервісу для генерації протоколів за результатами виборів старости потоку.

Враховуючи ці проблеми, необхідно знайти рішення, які б задовольнили представників Студентської Виборчої Комісії та надали студентам НаУКМА якісний сервіс, що б спростив процес електронних виборів всередині університету.

1.4. Необхідність синхронізації баз даних університету та електронної системи виборів

Для того, щоб використовувати актуальні дані про студентів в новій електронній системі виборів, необхідно впровадити новий сервіс для синхронізації бази даних системи виборів та системи “Оптіма”. Необхідно переконатися, що всі актуальні дані будуть підтягуватися з бази даних університету, а також потрібно надати можливість виконувати синхронізацію як власноруч, так і автоматично.

2. Створення структури проєкту та налаштування робочого середовища

2.1. Використання стратегії Monorepo для побудови структури проєкту

Використання монорепо (єдиний git-репозиторій для кількох програмних проектів) для створення проєктів та їх підтримки стає все більш і більш поширеною практикою, адже ця стратегія має багато суттєвих переваг над збереженням різних сервісів в різних репозиторіях. Ще в 2008 році можна було зустріти репозиторії, які містили в собі і фронтенд, і бекенд, але це зумовлено тим, що монолітність була основним підходом у побудові клієнт-серверних застосунків. З приходом мікросервісної архітектури з'явилась потреба розділяти сервіси, а також створювати репозиторій під кожен сервіс окремо. З першого погляду, це має бути зручно, адже команда розробників, що працює над конкретним сервісом, має змогу контролювати процеси, що виключно пов'язані з ним і не переживати, що робота іншої команди в цьому ж репозиторії зможе якось нашкодити їхнім наробкам чи завадити виробничим процесам. Проте з розвитком проєкту та розробкою нового функціоналу підтримка подібної інфраструктури може стати тягарем і створювати незручності для розробників.

Підхід збереження мікросервісів у різних репозиторіях може бути незручним, коли є потреба використовувати дані та кодову базу, що знаходяться в різних репозиторіях. Такі зв'язки стає важко підтримувати і задокументовувати. Також часто є ситуації, коли для серверної та клієнтської частини може використовуватися одна мова програмування, наприклад, TypeScript. Тоді стає важко перевикористовувати код, що описує спільні типи даних. За таких умов доводиться просто визначати одні і ті ж самі типи декілька

разів, що порушує паттерн DRY (Don’t Repeat Yourself). Також часто бувають ситуації, коли зі збільшенням сервісів у проєкті стає неможливо зрозуміти, звідки беруться ті, чи інші дані, чи як працює новий сервіс, адже документація, наприклад, швидко застаріває. Тоді єдиним варіантом залишається звернутися до розробника нового сервісу і дізнатися актуальну інформацію. Використання стратегії монорепо може вирішити ці проблеми, а також створити більш зручний процес розробки нового функціоналу і поліпшити забезпечення якості кодової бази.

Стратегія монорепо має багато переваг над збереженням мікросервісів у різних репозиторіях. По-перше, стає набагато зручніше підтримувати та оновлювати версії бібліотек, що використовуються в проєкті. Немає необхідності завантажувати кожен репозиторій, оновлювати версію і потім завантажувати нову версію в хмару. Достатньо завантажити один репозиторій, змінити версію в одному конфігураційному файлі і зробити всі інші відповідні дії. По-друге, вирішується проблема перевикористання кодової бази в різних сервісах, адже налаштувати імпорт модулів всередині одного репозиторію набагато простіше, ніж придумувати складні рішення для перевикористання коду, а інколи - це може бути просто неможливо. По-третє, це може поліпшити процес впровадження нових можливостей застосунку. Тепер розробники, що будуть перевіряти ваш код через Pull Request, зможуть мати повний контекст змін, адже бачитимуть зміни в кожному сервісі, що ви поліпшували.

Можна зробити висновок, що використання стратегії монорепо значно підвищує швидкість та якість розробки застосунків, допомагає перевикористовувати кодову базу, а також сприяє підвищенню якості перевірки змін, що вносяться у проєкт.

2.2. Інструменти для роботи з Monorepo

Через більше поширення стратегії організації проєктів монорепо, з'явилася необхідність створення додаткових інструментів для роботи з ним. В першу чергу необхідно сказати про роботу зі спільними залежностями в монорепо. Оскільки нова електронна система виборів розробляється за допомогою мови програмування TypeScript [4], то в обговоренні будуть згадуватися технології, які залучаються в роботі саме з цією мовою.

На даний момент є три найпопулярніших менеджерів пакетів для роботи з пакетами в середовищі Node.js: npm, yarn та pnpm [5]. Кожен з них має свої переваги та недоліки, але для використання в цьому проєкті було вирішено обрати саме pnpm, оскільки він відзначається своєю швидкістю та економністю використання простору на диску. Це пояснюється тим, що він зберігає всі залежності в глобальному сховищі, а в `node_modules` пакети використовують посилання [6] на ці залежності, що дозволяє зекономити місце не зберігаючи зайвий раз вже існуючі пакети на диску. Саме ця особливість користування простором вирізняє його серед інших менеджерів для роботи з пакетами. Також варто зазначити, що всі з перелічених менеджерів пакетів мають вбудовану підтримку монорепо структур проєкту. Це означає, що якщо ви встановите залежності в корені вашого монорепозиторія, то усі проєкти чи бібліотеки матимуть доступ до цих залежностей. Якщо з'являється потреба використати в проєкті залежність з іншою версією, ніж в корені монорепозиторія, чи потрібно обмежити доступ до неї з інших проєктів чи бібліотек, то необхідно просто встановити цю залежність всередині цього проєкту окремо в його `package.json`.

Також pnpm має додатковий функціонал для виклику команд проєктів та бібліотек, що знаходяться всередині монорепо, прямо з кореня монорепо. Для

цього потрібно спочатку створити конфігураційний файл `pnpm-workspace.yaml`, в якому потрібно вказати шляхи до підпроєктів, які використовуються всередині монорепозиторія. Після цього потрібно утворити залежності до цих підпроєктів виконавши команду `pnpm install`, що дозволить використовувати посилання на них всередині інших підпроєктів. Виконавши ці дії, можна викликати команди підпроєктів з кореня монорепо. Це можливо зробити декількома способами:

- Використовуючи параметр `--recursive, -r`. Він виконає всі вказані скрипти в кожному підпроєкті, якщо в його пакеті є такий скрипт. Це зручно, якщо є потреба виконати всі схожі команди, які є в проєкті. Наприклад, `pnpm -r run test`. Подібна команда виконає всі команди `test`, які є всередині монорепозиторія;
- Використовуючи параметр `--filter, -F`. За допомогою спеціального синтаксису він надає можливість обмежити вибірку підпроєктів, до яких буде застосовано цю команду. Якщо умови підпроєкту, а також присутності команди виконуються, то команда, що вказується в попередній команді `pnpm` виконується зсередини підпроєкту. Наприклад, `pnpm -F {apps/client} run build`. Подібна команда виконає команду `build` всередині підпроєкту, який знаходиться в підпапці `client` папки `apps`.

Є й інші способи для спеціального викликання команд підпроєктів, але розбирати їх всі немає потреби.

Окрім додаткових функцій пакетних менеджерів, поліпшити роботу з монорепозиторіями можуть й інші інструменти. Turborepo – це інструмент, який дозволяє оптимізувати виконання команд, які використовуються в монорепо,

встановивши між ними зв’язки та залежності, що в свою чергу допомагає уникнути повторень у виконання, кешувати результати виконання команд, щоб, наприклад, потім перевикористати їх у CI (Continuous Integration) проєкту. Це може значно зменшити витрати на виконання CI, поліпшити досвід розробки та структурувати код.

2.3. Налаштування робочого середовища

Для того, щоб почати розробку електронної системи виборів, необхідно підготувати робоче середовище, що сприятиме ефективній розробці, та структуру проєкту.

По-перше, необхідно підготувати проєкт до встановлення залежностей. Оскільки у якості менеджера пакетів було обрано саме `pnpm`, то для того, щоб створити конфігураційний файл, де будуть описуватися залежності, до яких матимуть доступ всі підпроєкти монорепозиторія, необхідно виконати команду `pnpm init`. Після її виконання створиться файл package.json, де описується основна інформація про проєкт: назва, опис, авторство, ліцензія та скрипти.

Оскільки для створення серверної частини використовується фреймворк NestJS, то було б логічно використати інструменти цього фреймворку, що дозволяють швидко створити необхідну структуру проєкту, налаштувати лінтери та інші конфігураційні файли. Для створення основи проєкту достатньо виконати команду `nest new <назва_проєкту>` [7]. За допомогою цієї команди створюється:

- Конфігураційний файл `tsconfig.json` для TypeScript, що надає можливість правильно компілювати TypeScript в JavaScript, встановлювати правила для перевірки правильності написання коду,

прописувати псевдоніми для шляхів, щоб спростити досвід розробки та багато іншого;

- Конфігураційний файл `.eslintrc.js`, що визначає налаштування для статичного аналізатора ESLint, який перевіряє код на наявність помилок;
- Конфігураційний файл `nest-cli.json`, який описує застосунки та бібліотеки, які знаходяться в проєкті, шляхи, взаємозв'язки між ними, додаткові налаштування для компілятора та налаштування NestJS CLI;
- Конфігураційний файл `.prettierrc`, який визначає правила для Prettier, бібліотеки, яка допомагає форматувати код.

Також ця команда додає до `package.json` необхідні залежності, що потребує NestJS та виконує команду `pnpm install` для завантаження їх з регістру пакетів. Виконання команди `nest new <назва_проєкту>` створює заготовку для звичайного проєкту, який не має всередині підпроєктів та не є монорепозиторієм. Однак, для того, щоб перетворити цей проєкт, щоб конфігурація NestJS була налаштована працювати з монорепозиторієм, достатньо ініціалізувати застосунок або бібліотеку використовуючи інструменти NestJS CLI. Для цього існує потужний функціонал команди `nest generate <app | library> <назва_застосунку/бібліотеки>`. Виконавши її, структура проєкту трохи зміниться і налаштування `nest-cli.json` пристосуються до роботи з монорепозиторієм. Якщо генерується app, то NestJS CLI створює нову директорію з назвою `apps`, всередині якої будуть знаходитися застосунки цього проєкту. Він також додасть нову папку всередині `apps` та наповнить її необхідними конфігураційними файли, включаючи `tsconfig.app.json`, та створить застосунок NestJS зі стандартним модулем,

контроллером та сервісом. Так само це працює і при виконанні цієї команди з аргументом library.

Проєкт складатиметься з трьох застосунків та двох бібліотек. В папці apps необхідно згенерувати два застосунки, один з них – `kma-vote`, який є серверною частиною електронної системи виборів, а також `sync`, який буде окремим серверним застосунком, який відповідає за синхронізацію бази даних `kma-vote` та системи “Оптіма”. Використовувати NestJS CLI для генерації базового коду для клієнтського застосунку немає необхідності, оскільки для цього буде використовуватися функціонал Next.js CLI. В папці `libs` необхідно згенерувати дві бібліотека. Перша бібліотека називатиметься `configs`, всередині якої будуть знаходитися конфігураційні файли та налаштування до серверних та клієнтського застосунку. Вона може залучати налаштування підключень до баз даних, черги BullMQ, SSL, сесії, стратегії аутентифікації та інше. Назвою другої бібліотеки є `core`. Всередині неї будуть знаходитися спільні елементи, класи та сутності, що можуть використовуватися в різних застосунках: як серверних, так і клієнтських.

2.4. Особливості модульності та використання NestJS, як фреймворка для розробки серверних застосунків

NestJS зарекомендував себе, як надійний фреймворк для створення серверних застосунків в середовищі Node.js [8]. Він встановлює достатньо жорсткі правила для розробника, що дозволяє уникнути найпростіших помилок та побудувати чисту архітектуру. Такий код легко підтримувати, тестувати та піднімати на інфраструктурі. NestJS має багато вбудованих можливостей, такі як підтримка middlewares, перехоплювачі помилок, наявність декораторів, тощо. Варто також звернути увагу, що NestJS використовує паттерн DI (Dependency

Injection) для того, щоб створювати взаємозв’язки та залежності між модулями та різноманітними провайдерами [9]. Цей паттерн є одним з прикладом використання інверсії керування або Inversion of Control (IoC). Цей принцип проєктування широко використовується фреймворками для побудови серверних застосунків у різних мовах програмування. Він відрізняється від процедурного програмування тим, що на відміну від процедурного програмування, де написаний код напряму викликає якісь бібліотеки для виконання дій, сам фреймворк відповідає за те, щоб виконувати цей код, який був написаний розробником, а не навпаки. Це дозволяє віддати контроль над діями всередині застосунки фреймворку, що може бути значно ефективнішим, а також уникати помилок, що пов’язані з фазами життєвого циклу програми.

Використання модулів, як основних будівельних та логічних блоків, допомагає розробнику відділити різні частини проєкту за їхньою функціональністю. Вони інкапсулюють в собі провайдери, сервіси та контролери, які пов’язані з цим модулем, а також надають можливість поширювати доступ до них іншим модулям. Такі особливості дозволяють писати код, який легко підтримується. Також варто зазначити, що такий підхід допомагає створювати частини програми, які мають слабкі зв’язки, а отже менше залежать одна від одної.

3. Реалізація електронної системи виборів

3.1. Опис вимог до електронної системи виборів

Для того, щоб імплементувати систему електронних виборів, що б могла задовольнити потреби Студентської Виборчої Комісії, необхідно описати повний список вимог до такої системи. Це дозволить наперед визначити майбутню архітектуру системи.

Вимоги до клієнтського веб-застосунку:

- Користуватися електронною системою виборів можуть лише студенти університету;
- Вхід до системи відбувається за допомогою сервісу Microsoft Office 365;
- Існує два типи доступу: звичайний користувач та адміністратор;
- Тільки адміністратор має можливість створювати нові голосування;
- Для створення голосування необхідно задати назву, ключ посилання (він створюється автоматично), опис голосування, мінімальну та максимальну кількість опцій, які зможе обрати той, хто голосуватиме, дати початку та закінчення голосування, можливість приховати голосування від перегляду іншими користувачами, налаштування доступу до голосування та обрання кандидатів;
- Налаштування доступу відбувається завдяки випадаючим спискам, які містять назви факультетів, спеціальностей, років навчання та ступені здобуття освіти. Адміністратор обирає потрібні налаштування. Якщо поле не задане, то воно не буде враховуватися при розрахунку доступу до електронної системи виборів;

- Усі користувачі сервісу можуть переглянути список голосувань, якщо вони не були приховані адміністратором;
- Інтерфейс відображення голосувань має можливість фільтрації голосувань. Він дає змогу показувати голосування, які мають конкретний статус, наприклад, “В процесі” або “Скасовано”, тощо. Також є можливість окремо переглянути голосування, до яких користувач має доступ або в яких він вже брав участь. Також інтерфейс має мати пагінацію, щоб не перевантажувати інтерфейс зайвою інформацією;
- Перейшовши на сторінку голосування, користувач може переглянути основну інформацію про голосування, результати голосування, якщо воно вже було завершено, або окрему секцію з можливістю проголосувати, якщо такий користувач має доступ до цього голосування та не брав в ньому ще участь;
- Адміністратор має можливість відредагувати голосування, якщо воно ще не почалося;
- Адміністратор має можливість власноруч змінювати стани голосувань використовуючи інтерфейс системи.

Вимоги до серверного застосунку:

- Використання TypeScript та Node.js;
- Комунікація з клієнтським застосунком має відбуватися за допомогою REST API;
- Для збереження даних застосунку використовується СУБД Postgres;
- Для збереження даних про сесію користувача використовується Redis;
- База даних має зберігати інформацію про голосування, варіанти виборців, результати голосувань, студентів, які на даний момент

навчаються, користувачів системи, а також голосів, які були віддані студентами на голосуванні;

- Кожна сутність в Базі Даних має мати можливість бути видобута та видозмінена використовуючи базові операції CRUD [10];
- Для генерації документації по існуючим інтерфейсам використовується Swagger;
- Сервіс має мати захист від несанкціонованого доступу використовуючи ролі, які будуть містити в собі обмежені права та застосовуватися у разі виклику певних методів контролера, якщо такі обмеження будуть застосовані;
- Сервіс має відповідати за своєчасну зміну станів голосувань, якщо час початку чи закінчення такого голосування стає меншим за теперішній час;
- Сервіс має повертати помилку, якщо доступ до ресурсу обмежено для даного запиту;
- Аутентифікація користувача має здійснюватися використовуючи наявні можливості сервісу Microsoft Office 365. Для цього має використовуватися протокол OIDC (OpenID Connect);
- Для взаємодії з СУБД Postgres використовується бібліотека TypeORM, яка виконує функції ORM (Object–relational mapping) [11];
- Сервіс має використовувати фреймворк NestJS, як основу для його функціонування та організації;
- Результати голосування мають зберігати інформацію про кандидатів у форматі JSON. Якщо використовувати зв’язки в базі даних та діставати інформацію про кандидатів за ключем, то у випадку видалення кандидата з бази даних, в майбутньому при перегляді результатів таких виборів на місці інформації про кандидата буде

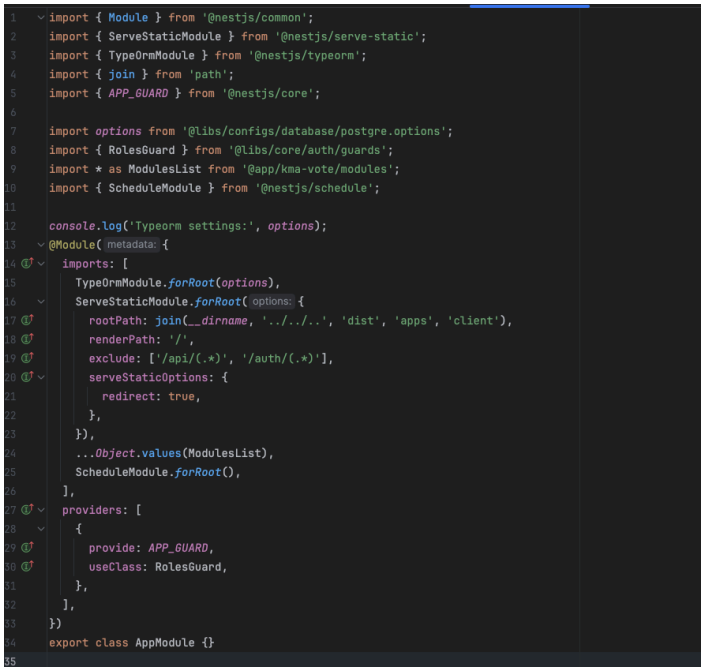
міститися пусте поле. Також результати голосування мають зберігати кількість голосів за кожного кандидати, а не вираховувати динамічно. Це теж пов’язане з тим, що когось з голосуючих може бути видалено з бази даних і, для того, щоб уникнути подібних ситуацій, необхідно зберігати такі дані напряму в базі даних.

3.2. Розробка серверної частини електронної системи виборів

3.2.1. Конфігурація сервісу та налаштування підключення до бази даних

Для початку імплементації серверного застосунку необхідно створити заготовку використовуючи NestJS CLI. Щоб це зробити, потрібно виконати команду `nest generate app kma-vote`. Після цього можна починати налаштування головних компонентів застосунку.

Всередині головного модуля `app.module.ts` відбувається налаштування та підключення всіх модулів, які використовує сервіс. Це стосується модуля для роботи з TypeORM, модуля, який відповідає за доступ до статичних сторінок сайту, які генерує клієнтська частина застосунку, модулів, які будуть реалізовані пізніше, включаючи модуль аутентифікації, та модуля, що відповідає за виконання задач, які можуть бути заплановані на виклик в певний час.



```
import { Module } from '@nestjs/common';
import { ServeStaticModule } from '@nestjs/serve-static';
import { TypeOrmModule } from '@nestjs/typeorm';
import { join } from 'path';
import { APP_GUARD } from '@nestjs/core';

import options from '@libs/configs/database/postgre.options';
import { RolesGuard } from '@libs/core/auth/guards';
import * as ModulesList from '@app/kma-vote/modules';
import { ScheduleModule } from '@nestjs/schedule';

console.log('Typeorm settings:', options);
@Module( metadata: {
  imports: [
    TypeOrmModule.forRoot(options),
    ServeStaticModule.forRoot( options: {
      rootPath: join(__dirname, '../../..', 'dist', 'apps', 'client'),
      renderPath: '/',
      exclude: ['/api/(.*)', '/auth/(.*)'],
      serveStaticOptions: {
        redirect: true,
      },
    }),
    ...Object.values(ModulesList),
    ScheduleModule.forRoot(),
  ],
  providers: [
    {
      provide: APP_GUARD,
      useClass: RolesGuard,
    },
  ],
})
export class AppModule {}
```

Рис. 1 – Налаштування головного модуля серверного застосунку

Для запуску сервісу необхідно мати стартовий файл, його назвою буде `main.ts`. Саме всередині нього відбувається створення та налаштування застосунку, додавання Swagger, ініціалізація глобальних фільтрів, middleware, тощо. Також всередині цього файлу відбувається виклик функції, яка ініціалізує аутентифікацію в застосунку разом з необхідними налаштуваннями та конфігураційними файлами. Всередині цієї функції налаштовуються middleware, що відповідають за збереження сесії в Redis, а також ініціалізації бібліотеки PassportJS, яка дозволяє налаштовувати різноманітні стратегії аутентифікації всередині застосунків.

TypeORM – це бібліотека, що спрощує взаємодію з різними СУБД. Замість того, щоб займатися з’єднанням з базою даних, написанням запитів, TypeORM бере це на себе. Єдине, що необхідно для роботи з бібліотекою – це налаштувати параметри для з’єднання з базою даних та створити класи для всіх сутностей, які є в базі даних, визначивши поля, типи даних та взаємозв’язки між сутностями. TypeORM також має інструменти для роботи з міграціями, що

дозволяють синхронізувати описані сутності в коді, та сутності в базі даних.

Для налаштування TypeORM необхідно для початку вказати параметри підключення до бази даних. Доступ можна отримати, якщо в конфігураційному файлі наявні хост, порт підключення, ім’я користувача, пароль та назва бази даних. Також TypeORM потребує додаткових налаштувань виводу інформації в консоль, а також доступу до існуючих класів сутностей, які мають бути в базі даних, та місцезнаходження файлів міграцій. Після налаштування бібліотеки, її можна повноцінно використовувати в інших модулях сервісу використовуючи метод `forFeature`, який дозволяє обрати репозиторії яких сутностей можна буде використати всередині компонентів цього модуля. Окрім сутностей, можна вказати джерело даних, тобто, якщо існує декілька баз даних, з якими має зв’язок застосунок, то необхідно вказати назву цього джерела, яка визначається в конфігураційному файлі.

```
import { DataSourceOptions } from 'typeorm';
import * as entities from '@libs/core/database/entities';

const config: DataSourceOptions = {
  type: 'postgres',
  name: 'main',
  host: env.string( key: 'POSTGRES_HOST',  defaultValue: 'localhost'),
  port: env.number( key: 'POSTGRES_PORT',  defaultValue: 3306),
  username: env.string( key: 'POSTGRES_USERNAME',  defaultValue: 'root'),
  password: env.string( key: 'POSTGRES_PASSWORD',  defaultValue: ''),
  database: env.string( key: 'POSTGRES_DATABASE'),
  synchronize: env.bool( key: 'POSTGRES_SYNCHRONIZE',  defaultValue: true),
  logging: env.bool( key: 'POSTGRES_LOGGING',  defaultValue: false),
  migrationsRun: true,
  extra: {
    charset: 'UTF-8',
  },
  entities: [...Object.values(entities)],
  migrations: ['../../../core/src/database/migrations/**/*{.ts,.js}'],
};

if (env.get( key: 'NOVE_ENV') === 'development') {
  console.log('Using DB config from the ormconfig.js file ...');
  console.log(config);
}

export default config;
```

Рис. 2 – Налаштування джерела даних в TypeORM

3.2.2. Використання бібліотеки TypeORM для взаємодії з базою даних

Після налаштування та встановлення з'єднання з базою даних, TypeORM дає можливості для різноманітних операцій з сутностями. Визначивши структуру сутності та синхронізувавши її з базою даних, є змога взаємодіяти з нею двома способами: Active Record або Data Mapper [12].

У першому випадку, клас сутності має успадкувати клас `BaseEntity`, який містить в собі методи для взаємодії з конкретним екземпляром класу, пошуку записів за певними параметрами або доступу до генератора SQL-запитів. Якщо є потреба розширити наявні метод у класу сутності, то достатньо створити статичний метод, що буде взаємодіяти з сутністю використовуючи `this` та методи наявні в базового класу.

У другому випадку, клас сутності не має успадковувати додатковий клас. Натомість для написання та використання методів для отримання екземплярів чи операціями над ними використовують окремі класи, які називають "репозиторіями". Для створення такого репозиторія для сутності достатньо використати метод класу `DataSource`, який називається `getRepository`.

Найпростіший спосіб використання TypeORM в модулях NestJS – це ін'єкція репозиторія в сервіс модуля використовуючи декоратор `@InjectRepository` в конструкторі класу, що на рівні процесів всередині застосунку надає доступ до методу `getRepository` та використовує єдиний існуючий екземпляр класу `Data Source`, який ініціалізується в головному модулі. Варто зазначити, що це використання підходу DataMapper, адже класи сутностей не успадковують клас BaseEntity.

3.2.3. Опис сутностей бази даних

Користувачами сервісу можуть бути лише студенти, тому необхідно створити сутність `User` та сутність `Student`. Сутність `Student` містить в собі інформацію про конкретного студента, яка береться з бази даних університету. Це особливий ідентифікатор студента, ПІБ, рівень та рік навчання, назва факультету, спеціальності, курсу, пошта та статус. Натомість сутність `User` містить інформацію про користувача, яку він отримує під час аутентифікації засобами Microsoft Office 365. Під час цього процесу, якщо намагається увійти в систему саме студент, то за наявності запису в базі даних про такого студента, система створить відповідний екземпляр класу `User`, якщо такого ще немає. Ця сутність має поля, які зберігають інформацію про унікальний ідентифікатор користувача в сервісі аутентифікації, пошту студента, а також ролі, які він має.

Система має дві ролі – `'admin'` та `'user'`. Якщо система не має запису в таблиці сутності `Admin` щодо конкретного студента, то під час первинної аутентифікації він отримає роль `'user'`, інакше – `'admin'`.

```
1+ usages   vichnyi-myzuka
export enum Role {
  User = 'user',
  Admin = 'admin',
}
```

Рис. 3 – Ролі користувачів у електронній системі виборів

Для зберігання інформації про голосування є сутність `Election`, де міститься назва, опис, ключ посилання, мінімальна та максимальна кількість претендентів, за яку може проголосувати студент, булеве поле для приховання голосування, дата початку та дата кінця, статус голосування, JSON поле, що містить фільтр для обмеження доступу до голосування, та метадані, які зберігають час створення та змінення голосування. Збереження інформації про кандидатів, що беруть участь у голосуванні, відбувається за допомогою створення сутності `ElectionOption`, яка містить метадані та зв'язки з `Student` та `Election`, які є зв'язками типу "багато-до-одного". Для збереження інформації про результати виборів створена сутність `ElectionResult`, що зберігає інформацію про студента, який був кандидатом, у форматі JSON, щоб уникнути втрати даних на випадок видалення цього студента з бази даних, кількість голосів, яку за нього віддали, а також метадані та зв'язки з сутністю `Election` та `ElectionOption`.

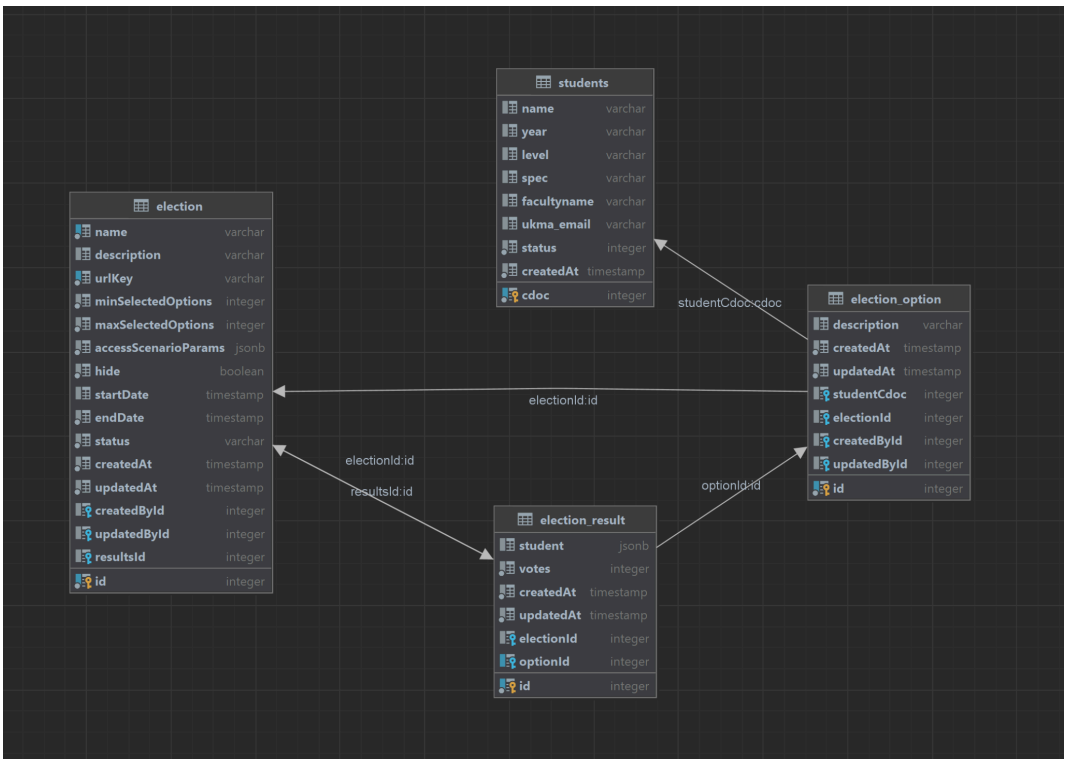


Рис. 4 – Зв’язки між сутностями Election, ElectionOption, ElectionResult та Student

Для збереження інформації про голоси студентів, які взяли участь у виборах, використовується сутність `Vote`. Вона зберігає метадані про час, коли було створено та оновлено голос, зв’язок зі студентом, який проголосував та зв’язок “багато-до-багатьох” з сутністю `ElectionOption`, який встановлюється через додаткову таблицю `VoteOptionsElectionOption`, головний ключ якої складається з ключа з ідентифікатором `Vote` та ключа з ідентифікатором `ElectionOption`. За допомогою цього стає можливий множинний вибір кандидатів у голосуванні.

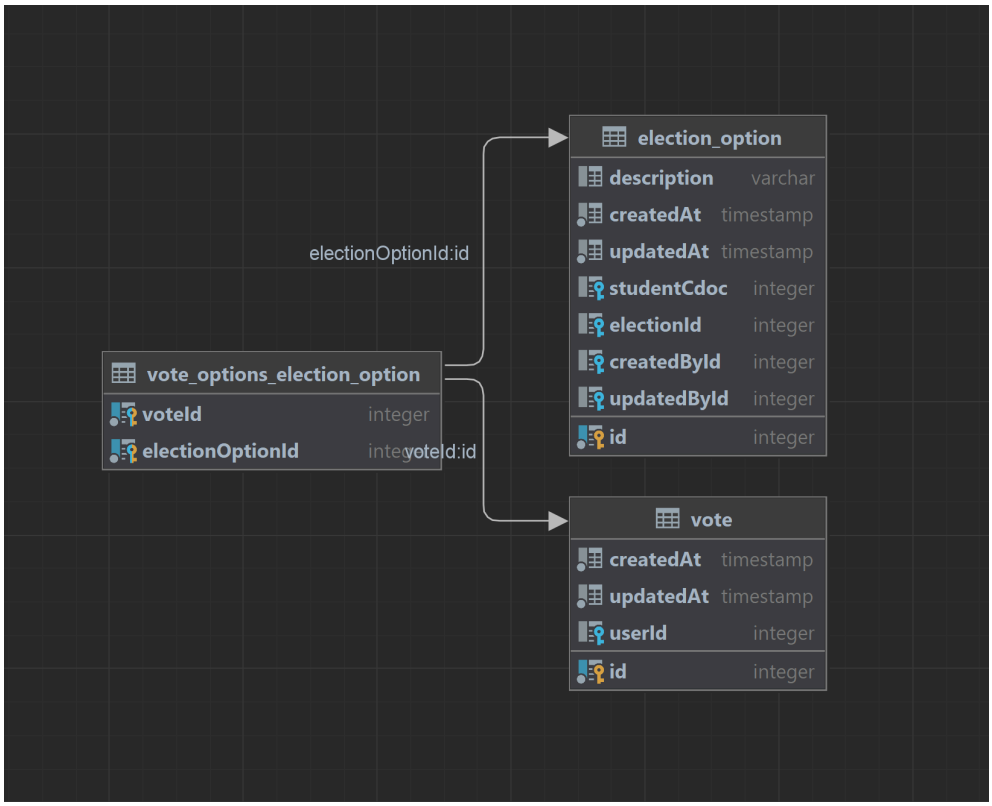


Рис. 5 – Зв’язки між сутностями Vote, ElectionOption та додатковою таблицею VoteOptionsElectionOption

3.2.4. Реалізація модулів, що відповідають за взаємодію з сутностями виборів, голосів та студентів

Модуль аутентифікації `AuthModule` має в собі відповідний сервіс та контролер. Контролер містить в собі методи, які повертають екземпляр сутності `User`, його ролі, також метод для виходу з системи, GET та POST запити, які викликаються безпосередньо під час процесу аутентифікації. Всередині них відбувається виклик методів бібліотеки PassportJS, а також методу сервісу `processAzureProfile`, який займається обробкою даних, що отримуються у разі успішної аутентифікації з сервісу Microsoft Office 365. Сервіс AuthService містить в собі вищезгаданий метод для опрацювання даних аутентифікації та метод перевірки на наявність у певної пошти прав адміністратора. Використовуючи вбудовані репозиторії TypeORM, метод `processAzureProfile` шукає, чи існує такий користувач з подібною поштою. Якщо так, то просто повертає його, інакше створює нового користувача та надає йому відповідні ролі.

Управлінням процесами, що пов’язані з голосуваннями, займається модуль `ElectionModule`. Він містить в собі три сервіси та три контролери, що відповідають за керування та отримання екземплярів голосувань, кандидатів та можливих варіантів для фільтрації доступу до голосувань. Сервіс `ElectionService` містить в собі різні методи для пошуку конкретних голосувань, перевірку на можливість проголосувати в певному голосуванні даним користувачем, базові CRUD операції над голосуваннями та методи для зміни станів голосування.

Голосування може мати 5 різних станів: Not Started (не розпочато), In Progress (в процесі), Cancelled (відмінено), Pause (на паузі) та Completed

(завершено). Кожен з цих станів створює певні обмеження для користувачів та адміністраторів. Наприклад, користувачі можуть голосувати лише, якщо голосування знаходиться в стані “In Progress” або адміністратор може редагувати голосування лише тоді, коли воно знаходиться в стані “Not Started”. Це необхідно, щоб обмежити користувачів від поведінки, що може зашкодити процесу голосування чи зробити його недійсним. Зміни станів можуть відбуватися як і за участі самої програми, наприклад, якщо час початку голосування стає меншим за теперішній час, то якщо голосування ще не має стану In Progress, сервіс має самостійно змінити стан голосування, або в ручному режимі викликаючи відповідні методи сервісу через контролер.

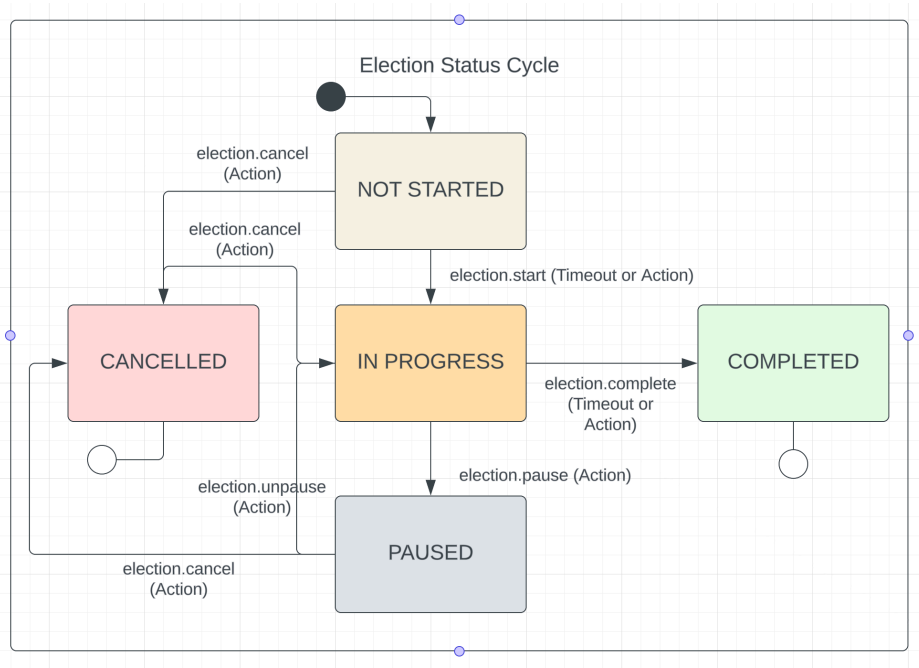


Рис. 6 – Діаграма зміни станів голосування

Сервіс `FilterOptionsService` має методи, що повертають актуальну інформацію про існуючі факультети, спеціальності, курси навчання та ступені, яка необхідна для виконання фільтрації доступу до голосування по певних параметрах. Це також необхідно для демонстрації можливих варіантів фільтрації

на клієнтському сервісі під час створення та редагування голосування.

Для керування голосами, що віддають користувачі електронної системи виборів, існує модуль `VotesModule`, що містить в собі відповідний сервіс та контролер. Сервіс має метод для голосування за певних кандидатів на голосуванні та метод, що повертає екземпляри сутності `Vote`, що пов'язані з певним голосуванням. Всередині методу для голосування відбувається валідація вхідних параметрів, перевірка на можливість користувача проголосувати в даному голосуванні та перевірка на те, чи голосування знаходиться в стані "In Progress".

3.2.5. Створення окремого модуля для виконання завдань, що потребують повторення прив'язаного до часу

Оскільки у голосувань має бути можливість змінювати стани голосування автоматично тоді, коли час початку чи кінця голосування настає, необхідно створити модуль `TasksModule`, який буде займатися такими завданнями. Він містить в собі сервіс `TasksService`, що має три методи. Перший метод використовує два інших, щоб виконувати перевірку на потрібні умови кожну хвилину за допомогою декоратора `@Cron`. Метод `updateEndingElections` займається видобутком всіх голосувань, які мають статус "In Progress" та мають менший час кінця голосування за теперішній, і використовує методи сервісу `ElectionService` для зміни стану голосування в "Completed". Таку ж дію, лише навпаки, виконує метод `updateStartingElections`.

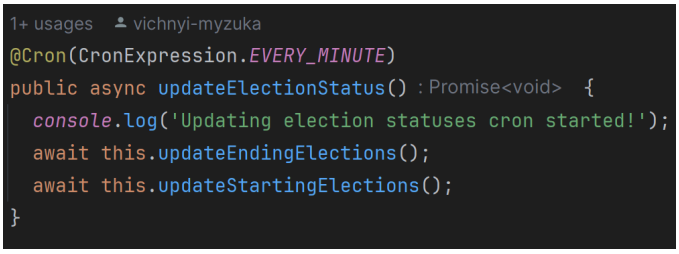


Рис. 7 - Метод сервісу TasksService для зміни станів голосувань

3.3. Розробка клієнтської частини електронної системи виборів

3.3.1. Конфігурація клієнтського сервісу

Для створення клієнтського сервісу електронної системи виборів було обрано фреймворк Next.js, який допомагає створювати веб-застосунки на основі бібліотеки React [13]. Застосунок будується використовуючи React-компоненти, в той самий час Next.js займається збиранням проєкту в бандл [14], оптимізацією та надає додаткові інструменти та бібліотеки для побудови веб-застосунків.

Створення базового налаштування проєкту відбувається за допомогою інструменту Next.js CLI, що дозволяє, використовуючи консоль, легко і швидко розгорнути застосунок. Під час виклику команди `npx create-next-app@latest` в консолі, Next.js CLI пропонує обрати налаштування для проєкту: чи буде використовуватися TypeScript, ESLint, Tailwind CSS та інші налаштування для шляхів, що стосуються структури проєкту. Оскільки Next.js потребує особливих налаштувань для ESLint, що відрізняються від тих, які були застосовані до серверного застосунку, то слід додати ESLint використовуючи Next.js CLI. Те ж саме стосується і TypeScript, бо це є у вимогах до сервісу. Окрім генерації базової структури та файлів, в корені сервісу створюється конфігураційний файл `next.config.js`, де розташовані налаштування для Next.js, які використовуються в проєкті.

Структура проєкту містить в собі папку `public`, де зберігаються статичні файли, та папка `src`, в якій знаходиться основні директорії сервісу, а саме:

- `api` – містить файли, що мають методи для роботи з серверною застосунком;
- `components` – містить React-компоненти, з яких будуються сторінки;
- `guards` – містить файли, що мають методи, які запобігають переходженням на інші сторінки, якщо не виконуються певні умови;
- `hooks` – містить файли, що мають методи, які виконують дії, що пов’язані з життєвим циклом програми;
- `pages` – містить структуру сторінок веб-застосунку;
- `storage` – містить файли, які пов’язані зі сховищем Effector;
- `styles` – містить стилі для сторінок;
- `theme` – містить файл, який визначає кольорову тему для бібліотеки MUI;
- `utils` – містить файли, які мають допоміжні методи.

3.3.2. Структура сторінок веб-додатку та реалізовані користувацькі можливості

Для виконання вимог до електронної системи виборів необхідно створити повноцінний веб-додаток, який дозволить користувачу дізнатися основну інформацію про використання, можливість переглядати та голосування, брати в них участь, а адміністратору створювати голосування, змінювати їхні стани та редагувати.

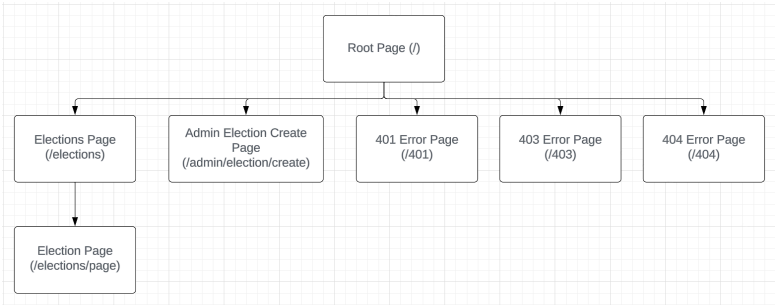


Рис. 8 - Структура сторінок веб-додатку

Початкова сторінка містить в собі короткий опис системи електронних виборів, щоб користувачі, що опинилася в додатку, змогли зрозуміти, що їм робити далі. Біля опису знаходиться кнопка, натиснувши на яку, користувач системи може увійти в неї використовуючи Microsoft Office 365. Це єдиний спосіб аутентифікації доступний в сервісу, що дозволяє впевнитися, що користувач є точно студентом університету.

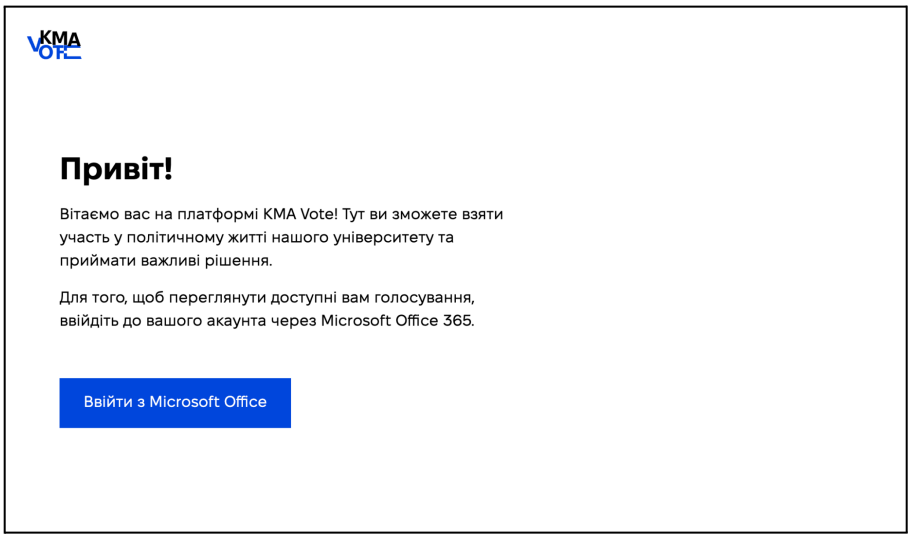


Рис. 9 – Початкова сторінка застосунку

Після успішної аутентифікації, користувач потрапляє на головну сторінку з голосуваннями. В залежності від того, яку роль має користувач, він зможе побачити додаткові елементи управління, як, наприклад, кнопку створення

нового голосування, чи додаткові дії у списках голосувань. Головним елементом сторінки є список усіх голосувань з перемикач для фільтрації. Фільтрами можуть бути як стани голосувань, так і, наприклад, голосування, в яких користувач може взяти участь чи брав раніше. Натиснувши на назву голосування, користувач переходить на сторінку цього голосування. Виконуючи певні дії щодо голосувань, адміністратор буде бачити спливаючі діалоги, в яких міститься інформація про успішну чи невдалу дію. Також, для того, щоб виконати якусь дію щодо голосування, адміністратор побачить спливаюче вікно, де системи просить підтвердити свою дію. Це зроблено, щоб уникати випадкових кліків.

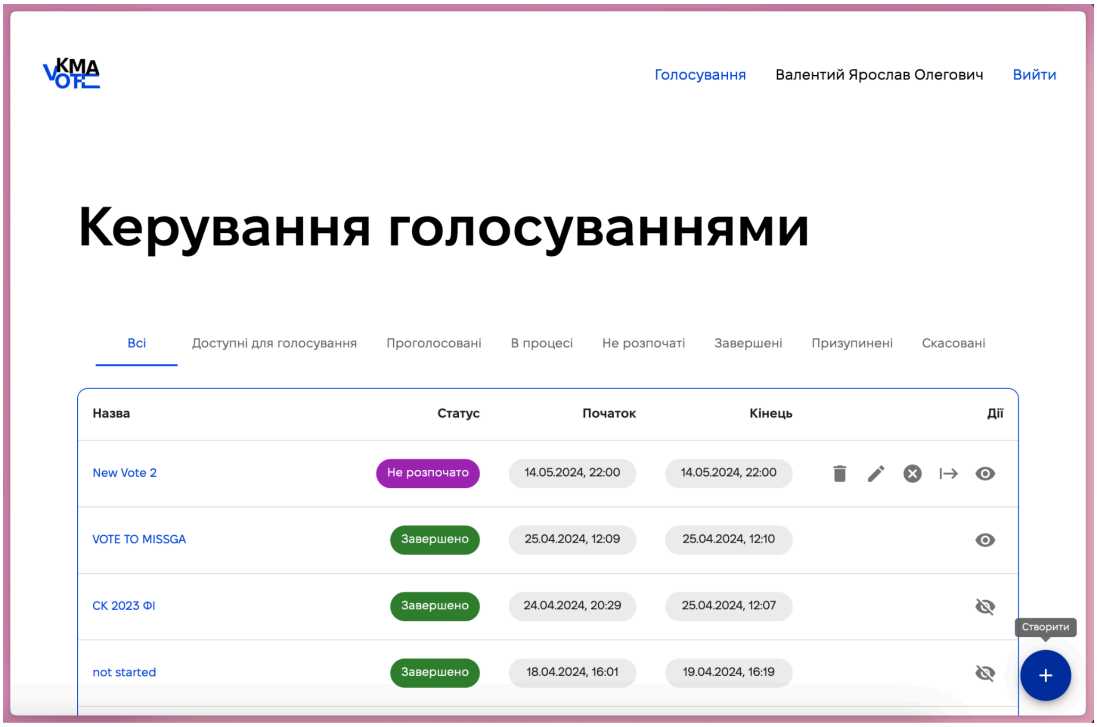


Рис. 10 – Сторінка з переліком голосувань

Натиснувши на кнопку з підписом “Створити”, адміністратор перейде на сторінку додавання нових голосувань. На цій сторінці для створення голосування можна заповнити поля назви, опису, максимальної та мінімальної кількості опцій вибору для людей, що будуть голосувати, дати початку та кінця

голосування, приховання голосування від користувачів. Також ця форма містить в собі секцію для налаштування фільтрації доступу до голосування, де використовуються випадні списки у якості елементів керування.

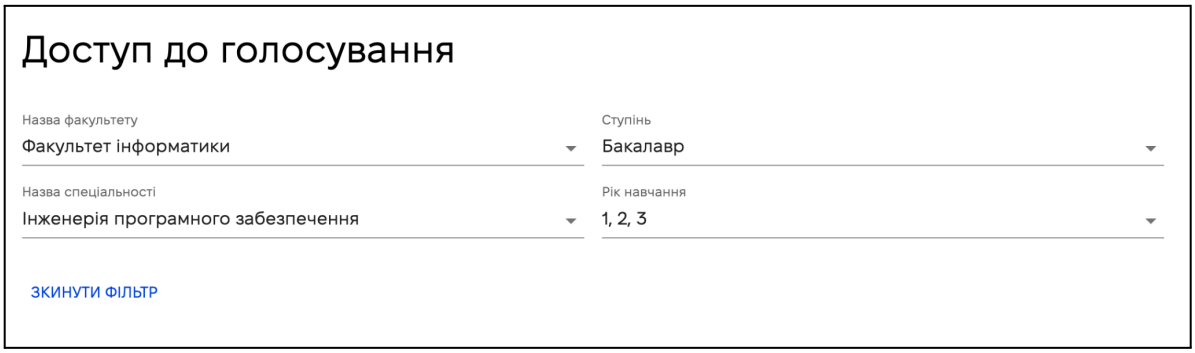


Рис. 11 – Секція налаштування фільтрації доступу до голосування

Окрім цього є секція обрання кандидатів, де знаходиться поле пошуку та таблиця з пагінацією для зручного вибору кандидатів. Після заповнення всіх обов’язкових полів, з’являється можливість натиснути кнопку “Створити голосування”, після натиску якої за успішного заповнення форми та проходженню валідації зі сторони клієнта та серверної частина, буде створено нове голосування.

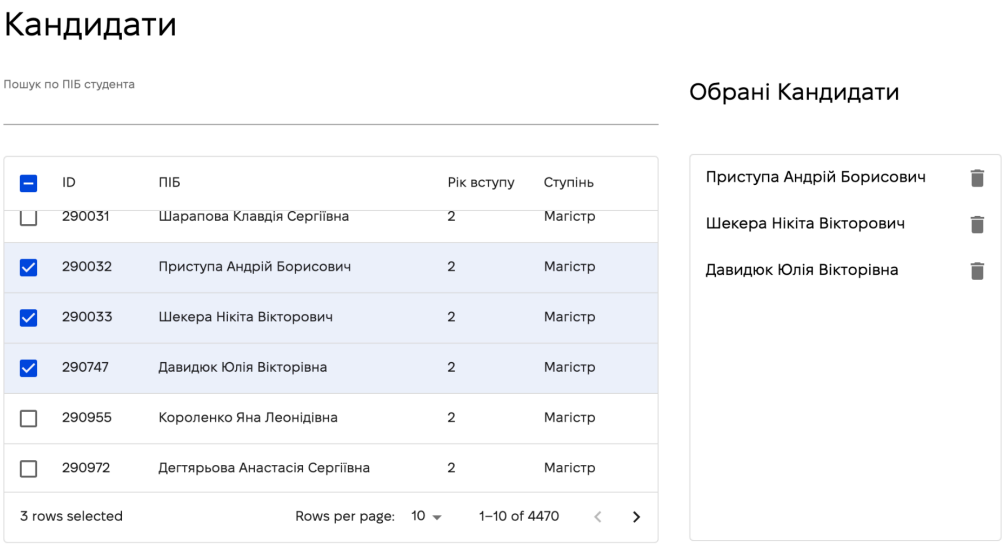


Рис. 12 – Секція обрання кандидатів

На сторінці перегляду голосування користувачі можуть переглянути детальну інформацію про голосування, прочитати опис та побачити, хто може проголосувати в даному голосуванні. Якщо голосування ще не почалося чи користувач не має доступу до цього голосування (або вже проголосував), то секція з вибором кандидатів не буде йому відображатися. Якщо голосування триває та користувач потрапляє у вибірку студентів, які мають право голосу, то якщо студент ще не голосував, він зможе бачити секцію сторінки, де знаходиться таблиця з списком кандидатів, де біля кожного ПІБ знаходиться прапорець. Якщо користувач натисне на прапорець, то він обере кандидата, за якого хоче проголосувати. Кількість прапорців, які може обрати користувач, обмежується налаштуваннями голосування. Якщо користувач обрав максимальну дозволену голосування кількість прапорців, то всі інші стануть не активними до дії і їх не можна буде натиснути. Щоб віддати голос, користувач мусить натиснути на кнопку “Проголосувати”. Якщо валідація і запит проходить, то користувача переносить на сторінку з голосуваннями та з’являється повідомлення про те, що голос був зарахований. Після цього

користувач більше не може брати участь в даному голосуванні, але може переглянути свій вибір на сторінці голосування. Після закінчення голосування користувач можуть дізнатися результати голосування переглянувши їх у таблиці з кандидатами на сторінці голосування. Ніхто не може дізнатися перебіг голосування до його закінчення задля збереження справедливості проведеного голосування. Адміністратор може лише бачити кількість відданих голосів загалом. Також адміністратор має можливість змінювати статус голосування використовуючи спливаючі кнопки. Вони розташовані в правому нижньому кутку екрану та закріплені під скролінгу. Викликаються вони за допомогою наведення курсору на відповідну область. Там також знаходиться кнопка, яка дозволяє викликати вікно редагування голосування. Воно містить форму схожу на ту, яка була на сторінці створення голосування. Єдиною відмінністю є відсутність можливості відредагувати назву голосування. Також редагування голосування доступне лише тоді, коли голосування ще не було розпочато. Це зроблено для того, щоб уникнути ситуацій втрати інформації або її пошкодженню шляхом невдалого користування системою.

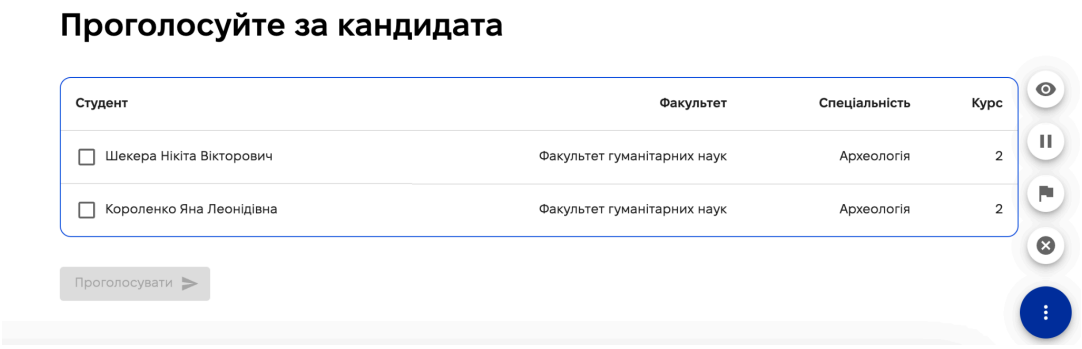


Рис. 13 – Таблиця для голосування за кандидатів та демонстрація додаткових можливостей адміністратора на сторінці голосування

Окрім сторінок, які дозволяють користувачам та адміністраторам взаємодіяти з електронною системою виборів, також є сторінки, що відповідають певним кодам помилок, які можуть виникати під час користуванням сервісу. Наприклад, 404, коли посилання, за яким намагається доступитися користувач не існує, чи 401, коли користувач не має права доступу до певного ресурсу, тощо. Також присутня переадресація на початкову сторінку, якщо відвідувач сервісу намагається відвідати сторінки, що потребують авторизації. Або навпаки, користувача переадресовує на сторінку з голосуваннями, якщо він увійшов, але намагається відвідати початкову сторінку авторизації.

4. Реалізація сервісу для синхронізації бази даних університету та бази даних електронної системи виборів

4.1. Конфігурація сервісу та налаштування джерел пов'язаних з базами даних для бібліотеки TypeORM

Налаштування сервісу для синхронізації баз даних є дуже схожим до того, яке було під час створення серверної частини для електронної системи виборів, адже вона теж використовує NestJS у якості фреймворка та теж є серверним додатком.

Всередині головного модуля `main.module.ts` відбувається налаштування підключень до бази даних університету та бази даних електронної системи виборів використовуючи бібліотеку TypeORM. Окрім цього імпортуються модулі для аутентифікації, щоб обмежувати доступ тим, хто має право користуватися сервісом для синхронізації, модуль синхронізації, в якому містяться необхідний контролер та сервіси, та налаштування модуля, що пов'язаний з бібліотекою BullMQ. Також головний модуль надає у використання іншим модулям програми створений екземпляр класу `Queue`, який створюється для роботи з чергами.

Запуск сервісу відбувається всередині файлу `main.ts`. В цьому файлі створюється екземпляр класу `NestApplication`, який визначає додаток, налаштовується Swagger, Redis для збереження сесій, а також створюється черга для синхронізації, використовуючи інструменти бібліотеки BullMQ та бібліотеку `@bull-board`, щоб додати UI-інтерфейс для перегляду та керування чергами.

Окрім налаштування коректного зв'язку з базою даних університету, також необхідно бути в одній мережі з нею, оскільки інакше доступитися до IP

бази даних буде неможливо. Як рішення, під час розробки сервісу було прийняте рішення використовувати VPN, щоб мати можливість доступу до бази даних університету. Після налаштування сервісу на інфраструктурі університету, за умови, що вони знаходитимуться з СУБД в одній підмережі, такої проблеми виникнути не має.

4.2. Використання бібліотеки BullMQ для вирішення проблеми синхронізації баз даних

BullMQ – це бібліотека для Node.js для роботи з чергами для різноманітних задач, які зберігаються в Redis [15]. Першопочаткова проблема способів синхронізації баз даних університету та інших сервісів, що створювалися раніше, була синхронність запиту. Коли адміністратор натискав у інтерфейсі кнопку для синхронізації, то надсилався запит серверу, але відповідь на цей запит приходила лише після того, як виконувалася синхронізація, а це могло тривати і годину. На відміну від цього підходу, під час запиту на синхронізацію BullMQ не запускає одразу синхронізацію, а створює задачу в черзі і надсилає назад відповідь, що задача була створена. Якщо черга вільна, то відбувається виконання синхронізації і після цього може відбутися певна демонстрація того, що синхронізація завершена. Для цього є окремий інтерфейс для роботи з BullMQ. Якщо черга зайнята, то задача буде чекати своєї черги, доки попередня не закінчиться або її не відмінять. Також BullMQ має можливість створювати додаткові Workers і виконувати завдання паралельно, але в даній задачі це не потрібно і може створювати проблеми, оскільки роботу з базами даних ліпше зберігати синхронною.

Після реєстрації черги всередині модуля `SyncModule`, всі інші сервіси всередині мають до неї доступ. Сервіс `SyncService` має всередині методи для створення задачі та завершення всіх поточних, які використовуються

відповідним контролером для керування сервісом за допомогою протоколу HTTP. Також у метод створення задачі можна додати змінну з об'єктом фільтрації, який є екземпляром класу `StudentFilter`, того самого, що використовується для фільтрації доступу до голосування. За допомогою нього можна обрати окрему вибірку студентів, які будуть синхронізуватися. Інколи бувають випадки, коли необхідно швидко синхронізувати студента якось курсу певного факультету. Завдяки додавання можливості обмежити розміри синхронізації, вирішення подібної задачі займає небагато часу.

Для визначення того, що буде відбуватися всередині завдання, яке знаходиться всередині черги BullMQ, необхідно створити окремий клас `SyncProcessor`, який буде визначатися декоратором `@Processor('sync', { concurrency: 1 })`, для знаходження відповідної черги. Оскільки цей провайдер знаходиться всередині модуля `SyncModule`, то може ввести потрібні екземпляри класу `EntityManager` для управління сутностями баз даних університету та сервісу для синхронізації.

Алгоритм синхронізації працює таким чином:

- За допомогою екземпляру класу `QueryBuilder`, що має зв'язок з базою даних університету, з урахуванням заданих обмежень фільтрації вибірки студентів, відбувається операція `SELECT` з отриманням інформації про 500 студентів;
- Якщо кількість студентів, які повернулися внаслідок операції, не дорівнює нулю, то відбувається виклик методу `performSync`, який отримує на вхід дані студентів. Всередині цього методу виконується операція `UPSERT` для оновлення даних про студентів в базі даних системи електронних виборів; Перший крок алгоритму повторюється;

- Якщо кількість студентів, які повернулися внаслідок операції, дорівнює нулю, то виконання завдання завершується і черга звільняється.

```
      }
    }
  });

  let currentResults: SqlServerStudent[];

  console.log(job.id + ' ' + JSON.stringify(job.data) + ' is processing');
  do {
    currentResults = await query.offset(this.currentOffset).getMany();
    console.log('Current offset is ' + this.currentOffset);
    this.currentOffset += this.STUDENTS_LIMIT;
    if (currentResults.length > 0) {
      await this.performSync(currentResults);
    }
  } while (currentResults.length > 0);
}

async performSync(optimaStudents: SqlServerStudent[]): Promise<void> {
  this.counter += optimaStudents.length;
  await this.mainEntityManager
    .getRepository(BaseStudent)
    .upsert(optimaStudents, conflictPathsOrOptions: ['cdoc']);
}

@OnWorkerEvent( eventName: 'completed')
onCompleted(job: SyncJob): void {
  console.log(job.id + ' is done, results length: ' + this.counter);
  this.counter = 0;
  this.currentOffset = 0;
}
}
```

Рис. 14 – Код, що реалізує алгоритм синхронізації баз даних

Провівши спробу використання сервісу, вдалося провести синхронізацію даних 26681 студентів за 5 хвилин 56 секунд. Якщо проводити обмежену фільтрацію по факультетам, то цей показник буде ще меншим. Також є можливості поліпшити використання даної системи в майбутньому дозволивши паралельні задачі у випадку, якщо вибірки студентів, по яким проводиться синхронізація, будуть різними і не перетинатися, що дозволить уникнути зайвих операцій.

Висновки

Спілкування з представниками Студентської Виборчої Комісії дало розуміння того, які недоліки мала минула система електронних виборів та які потреби є на даний момент. За допомогою цієї інформації були сформульовані вимоги до нової системи електронних виборів, обрано стек технологій та опис функціоналу, що має бути присутнім. Було розроблене та імплементоване практичне рішення, що використовуватиметься в майбутніх виборах до студентських органів самоврядування. Також можливості системи були продемонстровані представникам Студентської Виборчої Комісії, які дали свій відгук на створене рішення, а також пропозиції щодо розробки додаткового функціоналу в майбутньому. В межах роботи також було досліджене питання проблеми синхронізації баз даних університету та сервісів, які розроблялися для покращення різноманітних процесів, та імплементоване практичне рішення.

Список літератури

1. Побудова та розробка веб-сервісу для оптимізації бізнес-процесів на прикладі НаУКМА / Валентий Я.О. – Міністерство освіти і науки України, Національний Університет «Києво-Могилянська академія» – 2022 – 11 с.
2. Створення системи для проведення опитувань серед студентів НаУКМА / Кобзар О.О. – Міністерство освіти і науки України, Національний Університет «Києво-Могилянська академія» – 2017 – 38 с.
3. Why PHP Usage Has Declined by 40% in Just Over 2 Years - TheNewStack - https://thenewstack.io/why-php-usage-has-declined-by-40-in-just-over-2-years/
4. TypeScript - https://www.typescriptlang.org/
5. PNPM Vs. NPM Vs. Yarn: Which JavaScript Package Manager Should You Choose - DhiWise - https://www.dhiwise.com/post/pnpm-vs-npm-vs-yarn-which-javascript-package-manager
6. Hard link - Wikipedia - https://en.wikipedia.org/wiki/Hard_link
7. Workspaces – CLI - NestJS - https://docs.nestjs.com/cli/monorepo
8. Why I choose NestJS over other Node JS frameworks - Medium - https://medium.com/monstar-lab-bangladesh-engineering/why-i-choose-nestjs-over-other-node-js-frameworks-6cdbd083ae67
9. Providers - NestJS - https://docs.nestjs.com/providers#dependency-injection
10. Create, Read, Update and Delete - Wikipedia - https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

11. Object-relational mapping - Wikipedia - https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping

12. Active Record vs Data Mapper - TypeORM - https://typeorm.io/active-record-data-mapper

13. Introduction - Next.js - https://nextjs.org/docs

14. Javascript Bundling - Adobe - https://developer.adobe.com/commerce/frontend-core/guide/themes/js-bundling/#:~:text=JavaScript%20bundling%20is%20an%20optimization,the%20number%20of%20page%20requests.

15. What is BullMQ? - BullMQ - https://docs.bullmq.io/