МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ФІЗИКИ ТА МАТЕМАТИКИ

КАФЕДРА ІНФОРМАТИКИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ЕКОНОМІЧНОЇ КІБЕРНЕТИКИ

РОЗРОБКА ТА ПРОЕКТУВАННЯ БАЗИ ДАНИХ ДЛЯ СЕРВІСУ АНАЛІТИКИ ЕКОНОМІЧНОЇ ДІЯЛЬНОСТІ УНІВЕРСИТЕТУ

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти «бакалавр»

Виконав: студент 4 курсу 441 групи

Спеціальності 121 Інженерія

програмного забезпечення

Освітньо-професійної (наукової)

програми: Інженерія програмного

забезпечення

Цуркан Антон Федорович

Керівник: кандидат педагогічних наук,

доцент Вінник Максим Олександрович

Рецензент: кандидатка педагогічних

наук, доцентка

Кузьмич Людмила Василівна

Херсон – 2021

ЗМІСТ

ВСТУП

В останні роки намітилася тенденція збільшення обсягів бюджетних коштів, що надходять до вузів. В даний час бюджетні кошти йдуть вже не тільки на виплату заробітної плати та стипендій, а й на задоволення інших потреб вузів. Чим більше фінансових потреб тим більше проблем зі збором статистичних даних про всі грошові операції.

Зараз в бухгалтерії використовуються застарілі програми, які налічують в собі безліч таблиць, записів, транспортних файлів в\із казначейства. У такому обсязі інформації важко орієнтуватися, особливо якщо ти далекий від бухгалтерського ремесла. Зазвичай тільки досвідчені бухгалтери можуть розібратися та систематизувати данні в таких умовах.

Актуальність: В цій роботі буде розказано про розробку програми для роботи з даними для візуалізації фінансово-економічної статистики університету. Як і було сказано вище, намагатися зрозуміти данні через таблиці та голі цифри не проста задача. Додаток бере інформацію із бази даних та аналізує її згідно з умовами користувача, після чого візуалізує за допомогою діаграм, схем, блок-схем, графіків. Таким чином можна буде продемонструвати складні для сприйняття фінансово-економічні данні в більш зручному і зрозумілому вигляді, навіть для людини далекої від бухгалтерії.

Об'єктом дослідження є технології роботи з базою даних та візуалізацію інформації.

Предметом дослідження є база даних економічної діяльності університету.

Метою даної роботи написання API для бази даних економічної діяльності університету, та створення графічного звіту за допомогою BI-додатку.

Досягнення зазначеної **мети** здійснюється шляхом вирішення таких **основних завдань**:

1. Огляд бізнес-процесів на прикладі університету.
2. Огляд BI-технологій
3. Початок розробки на NestJS, створення моделей бази даних.
4. Створення Swagger API для взаємодії з базою даних.
5. Підключення сервісу Power BI для візуалізації обробленої інформації.

Структура роботи складається з вступу, двох розділів, висновку і списку літератури.

РОЗДІЛ 1

ОГЛЯД ЕКОНОМІЧНИХ СИСТЕМ ТА БІЗНЕС ПРОЦЕСІВ УНІВЕРСИТЕТУ

1.1 Аналіз бізнес процесів університету

Основи аналізу бізнес-процесів.

Основною проблемою, для якої шукалось рішення, було позначення, що описує бізнес-процеси, яке було б легко читаним і зрозумілим, для досвідчених аналітиків, а також для клієнтів і користувачів інформаційних систем, особливо для тих, хто здійснює моніторинг та керує цими процесами.

Для правильності і корисності оцінки методів аналізу бізнес-процесів, які будуть запропоновані, необхідно визначити, наскільки організація (в даному випадку Академія) підготовлена до аналізу, тому що технічно такий аналіз потрібно починати аж до підготовки опису в інформаційній формі бізнес-процесу. Був зроблений висновок, що:

- Необхідно мати можливість визначити, чи буде опис процесу (зроблений його власником) повністю відповідати реальній ситуації і, що більш важливо, буде містити всю інформацію, яку власник повинен вкласти в БПИФ (наприклад, під час аналізу деякі оброблені дані оцінюються як конфіденційні, що суперечить заяві власника з цього приводу).
- Без належного опису процесу, складеного власними співробітниками організації, зовнішній аудитор може не мати можливості належним чином розпізнати і оцінити невідповідності та неточності.

Була сформована і протестована теза про те, що якби бізнес-процес описувався методами моделювання бізнес-процесів, було б набагато простіше провести точне і ефективне оцінне інтерв'ю.

Моделювання бізнес-процесів

Позначення і методології моделювання можна розділити на дві групи:

1. Моделювання використовується для аналізу і оптимізації бізнес-процесів і бізнес-подій;
2. Моделювання для створення програмного забезпечення та інженерних цілей.

При описі процесу потрібно зосередитися на першій зі згаданих груп. Будучи об'єктно-орієнтованими, методології розробки програмного забезпечення привели до появи гнучких і потужних інструментів, таких як UML, але їх важче застосовувати в бізнес-моделюванні через ієрархічну модель організації.

Всередині першої групи графічні мови, корисні для подання концепцій, процедур, дій в рамках процесів, досить обширні. Серед найбільш важливих слід зазначити:

- DFD (Діаграма потоку даних),
- EPC (Ланцюжки процесів, керовані подіями)
- IDEF3 (Метод інтегрованого визначення 3, метод захоплення опису процесу),
- BPML v.1.1 (Мова моделювання бізнес-процесів),
- YAWL (Мова моделювання бізнес-процесів),
- BPMN (Позначення моделювання бізнес-процесів).

Недоліком графічних мов є особиста нотація, яка залежить від мови. Зазвичай будь-який стандарт містить унікальні символи, описи чи артефакти. Для організацій, що аналізують свої процеси на ефективність або можливості оптимізації, хорошим рішенням можуть бути узагальнені, менш формалізовані позначення. У разі розробки нового процесу, який, ймовірно, буде реалізований з використанням IT-систем, більш формалізовані ознайомлення становлять більш польові та корисні для переносу проекту безпосередньо на програмне забезпечення.

Прикладом такої мови може бути BPMN - його схеми, завдяки однозначності, дозволяють транскрибувати в BPEL (Мова виконання бізнес-процесів), тим самим забезпечуючи композицію, реалізацію об'єктно-орієнтованої моделі UML, що дає змогу автоматизувати створення програмного забезпечення.

У цьому випадку дослідження проводилося в Академії бізнесу в Домброва-Гурнича, і для методології моделювання бізнес-процесів був вибраний і використаний EPC. Перевагами цієї методології були особливо:

- просте використання нотації - діаграми EPC легко засвоюються,
- прозорість / ясність - діаграма читається без програмного забезпечення і зрозуміла користувачам процесів, що не володіє IT-навичками,
- масштабованість - модель дозволяє ідентифікувати основні завдання процесу і швидко визначати повноту на загальному рівні. Кожна дія може деталізувати і описати як підпроцес з іншого діаграмою EPC.

Схема EPC для цілей моделювання використовує два основних компоненти:

- Події,
- Діяльність.

Доповнені (за бажанням) об'єктами, що символізують:

- Особи або ролі, які беруть участь в діяльності,
- Документи, що є вхідними або вихідними даними процесу діяльності,
- IT-система, яка символізує концепцію передачі даних в систему або з неї,
- Місця, в яких здійснюється діяльність,

- Ризики, пов'язані з цією діяльністю.

На рисунку 1.1 приклад простої структури EPC.

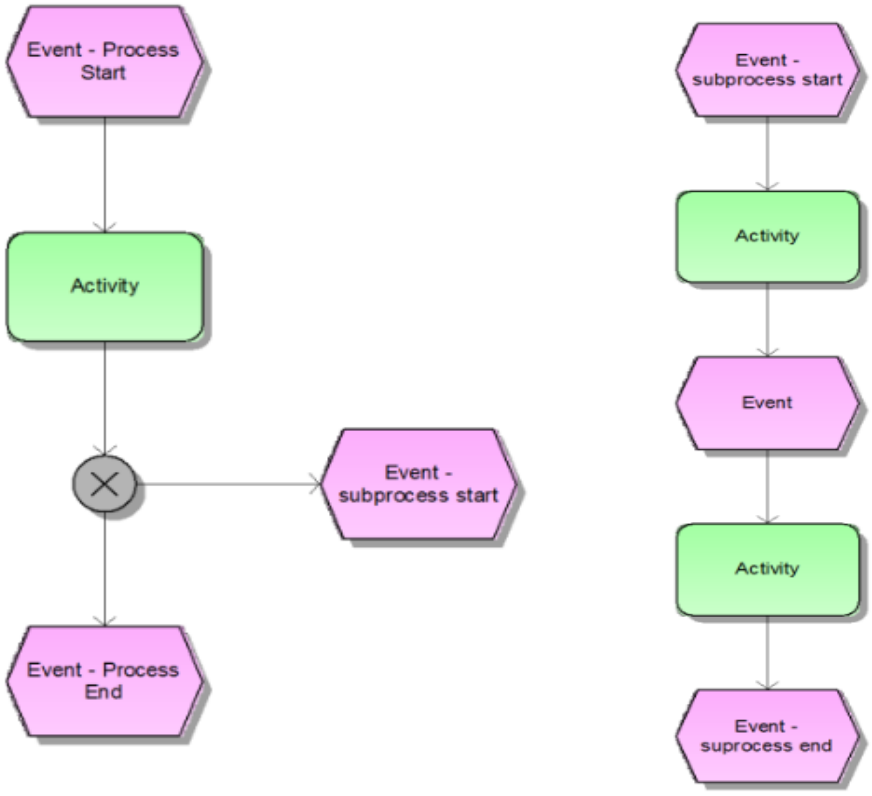


Рисунок 1.1 - Приклад простого опису процесу з використанням EPC як методу моделювання

Керівники середньої ланки дуже хорошої думки про використання EPC в якості методу аналізу процесів. Він був прийнятий дуже швидко, незважаючи на те, що це дослідження було їх першим досвідом використання цього методу. Це також виявилося дуже швидким методом усунення будь-яких недоліків в описі процесу, в Академії.

Модель процесу присвоєння диплома

Моделювання процесу за допомогою EPC-діаграми дозволило провести повний аналіз обраного процесу Академії - під час дослідження процес видачі дипломів був обраний в якості прикладу.

Процес присудження диплому вимагає різних компонентів - дані IT-системи, документальний фільм студента, зовнішні бізнес-послуги, співпраця з відповідним відділом для процесу (деканат) з іншими (факультети, фінанси) і іншими ресурсами Академії (наприклад, класи для випускних іспитів).

Процес присудження диплому - це комплекс дій, які відбуваються після того, як студент здає останній предметний, який студент повинен виконати самостійно, з моменту зарахування, закінчення дисертації в деканаті. З цього моменту деканат відповідає за перевірку статусу студента, організацію підсумкового іспиту, а після останнього іспиту - за документацію всього процесу, а також за архівування будь-яких документів. Процес закінчується видачою випускнику обох частин диплома (частина А і додаток В).

Рисунок 1.2 - Процес отримання дипломів в цілому

У загальному вигляді всі вищезгадані дії можна розділити на три великі підпроцеси (рис. 1.2). Однак тільки після докладного опису на діаграмі EPC можна оцінити ступінь складності процесу, виконати оцінку ризику або визначити, хто і які ресурси будуть задіяні в процесі.

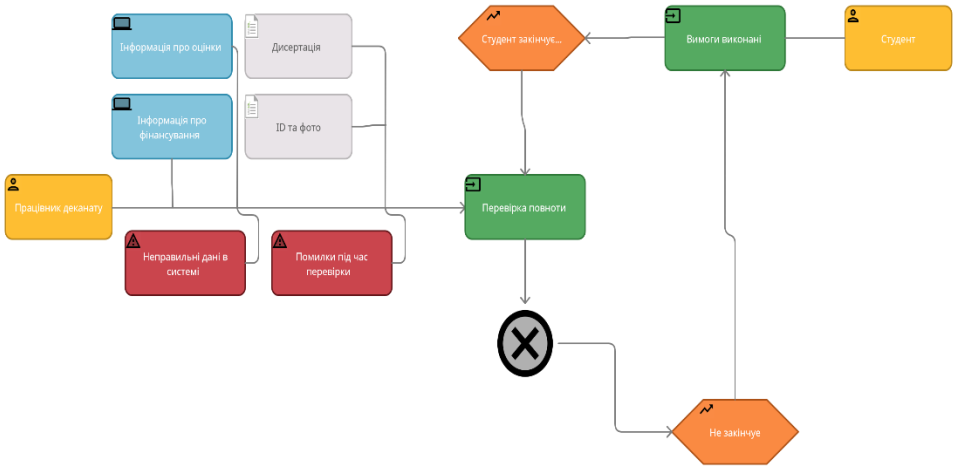


Рисунок 1.3 - Часткова діаграма EPC отримання диплому

На основі схеми бізнес-процесу, змодельованої за допомогою EPC, можна визначити і узагальнити детальну інформацію будь-якого роду.

- Співробітники організації, залучені в процес,
- Вхідні та вихідні документи в паперовій формі, а також в цифровому форматі,
- Дії, що відбуваються в процесі, в тому числі з використанням IT та логістичних компонентів,

Модель бізнес процесу, рекомендується починати аналіз бізнес-процесів з побудови EPC-діаграми. Це призводить до швидких результатів, формує основу для більш складних позначень, якщо вони будуть використовуватися, прискорює аналіз, та дозволяє більш ефективно аналізувати процес за допомогою діаграми EPC, навіть якщо це виконується власником процесу.

Переваги методу EPC:

- EPC зрозумілий для простого персоналу,
- використовувані ресурси перераховані і організовані,
- схеми зрозумілі і можуть бути менш або більш докладними,
- EPC простий у використанні.

1.2 Огляд BI-систем

Business intelligence (BI) - програмне забезпечення, створене для аналізу інформації в проекті, компанії або сторонніх даних. Інструменти які використовують BI-системи дозволяють генерувати звіти по даним, із баз даних, веб-ресурсів, хмарних сервісів, файлових та інших джерел інформації.

Використання **BI-інструментів:**

- В якості одного з інструментів управління - для контролю завдань, скорочення збитків, оптимізації процесів,
- Як систему моніторингу - для контролю процесів бізнесу, відстеження витрат, пошуку «больових точок»,
- Для вирішення PR-завдань - для демонстрації масштабів, можливостей і ключових показників бізнесу на виставках і конференціях, під час запуску нового проекту.

Дашборд - одна з можливостей платформ **Business Intelligence.** Це великі системи, які переводять дані бізнесу в доступну інформацію. Візуальне подання такої інформації називається **дашборд**.

На огляд були вибрані три BI-системи: Tableau, Qlik та Power BI.

Tableau

Tableau - система, яка дозволяє проводити глибокий і різнобічний аналіз, а потім результат в інтерактивній формі. Великі масиви даних збираються з різних джерел і в режимі реального часу на інформаційній панелі. При цьому це дуже гнучка, швидка і проста в освоєнні платформу.

Ключова відмінність - можливість суміщення даних з різних джерел і баз даних. Також платформа дозволяє одночасно працювати над звітом декільком користувачам. Поділитися результатом можна за посиланням, поштою або через сервер Tableau.

Ключова відмінність - можливість суміщення даних з різних джерел і баз даних. Також платформа дозволяє одночасно працювати над звітом декільком користувачам. Поділитися результатом можна за посиланням, поштою або через сервер Tableau.

Як джерело даних платформа може використовувати:

- Хмарні системи: Google BigQuery, Windows Azure тощо,
- Реляційні системи: SQL Server, DB2, Oracle тощо,
- Файлові системи Excel, CSV тощо,
- Будь-які інші джерела, які використовують програмний інтерфейс для доступу до баз даних.

Ключові особливості:

- Користувачі можуть самі створювати інструменти для дашборда і аналітики.
- Система працює з усіма пристроями, де є потоки даних, - не потрібно турбуватися про вимоги до обладнання або програмного забезпечення.
- Панелі інформації мають доступ до сховищ даних: SSAS Multi-Dimensional Cubes і DWH / DataWareHouse.
- Додатки для створення дашборда можуть створювати самі бізнес-користувачі.
- Над звітом можуть працювати відразу кілька користувачів.

Відмінні риси:

- Будь-який звіт створюється за кілька кроків.
- Ви маєте доступ до готових галузевим рішенням.
- Система встановлюється за кілька хвилин і не вимагає дорогого впровадження.
- Платформа обробляє дані будь-якого формату.
- Користувачі можуть створювати звіти будь-якої складності від простих до аналізу трендів.

- Сервіс має зрозумілий інтерфейс і швидко працює. Програма обробляє дані в режимі онлайн: фільтрує і сортує вхідну інформацію.

Продукти

- Tableau Desktop – настільна версія для окремого використання, підходить для аналітиків, консалтингових компаній та малого бізнесу. Для індивідуального використання дозволяє підключити до 6 джерел даних, для корпоративного – до 44.
- Tableau Online - хмарна платформа з безкоштовною версією за умови, що всі рішення зберігаються на загальному сервері з відкритим доступом. Також є приватна версія з платою за користування.
- Tableau Server - продукт для великих компаній, які мають свої сервера і хочуть повністю контролювати всі потоки даних і їх безпеку

Переваги

- Підтримує мобільні платформи.
- Легко інтегрується з платформами Big Data.
- Безліч вбудованих інструментів для імпорту даних з різних джерел.
- Велика кількість навчальних матеріалів: посібників, курсів, відео.
- Служба підтримки клієнтів і регулярні оновлення.

Недоліки

- Дані необхідно попередньо структурувати.
- Робота з Tableau Server вимагає консультацій IT-фахівця.
- Низький рівень захисту на рівні даних.
- У Tableau Server немає історії змін.

- Система не може замінити спеціалізовані додатки для створення фінансових звітів.

Qlik

Qlik - BI-платформа, яка відрізняється широким асоціативним пошуком і можливістю обробки обчислень в оперативній пам'яті.

Ключові особливості:

- Єдиний алгоритм аналітики працює у всіх версіях.
- Продукти надають можливість швидкої, інтерактивної візуалізації даних.
- Сервіс можна використовувати разом з іншими членами команди, можна ділитися будь-якими створеними додатками.
- Готові візуалізації можна використовувати для вибірок і досліджень.

Відмінні риси:

- Інтерфейс має високу інтерактивністю.
- Запити обробляються безпосередньо в оперативній пам'яті, а не в СУБД.
- Дані представляються у вигляді асоціативної моделі.

Продукти

Платформа представлена двома версіями. Обидві системи працюють на одній платформі і за єдиною логікою, але відрізняються позиціонуванням по відношенню до кінцевого споживача.

- QlikView - корпоративна система, яка вимагає впровадження і обслуговування IT-фахівцями.
- Qlik Sense - інструмент для персонального дослідження даних. Можна користуватися в десктоп і хмарної версії, працює на всіх пристроях, де є браузер. Призначений для приватних аналітиків і управлінців.

Переваги

- Імпорт з багатьох джерел.
- Дозволяє просто фільтрувати дані в будь-який візуалізації.
- Доступна спільна розробка.
- Швидка швидкість завантаження і обробки даних, створення графіків і таблиць.
- Зручно створювати будь-які фільтри діаграми, таблиці, графіки.

Недоліки

- Неможливо об'єднувати результати в закладках.
- При фільтрації система може комбінувати дані, навіть якщо це не потрібно.
- У користувачів, які не мають технічної бази можуть виникнути складності.
- Складно використовувати як інструмент для всієї компанії.
- Важко відправляти звіти.

Power BI

Power BI - набір інструментів бізнес-аналітики від Microsoft для аналізу і наочного уявлення цінної інформації.

Сервіс дозволяє використовувати сторонні додатки і безліч джерел даних. З його допомогою можна створювати кастомізовані візуалізації. Десктоп-версія дозволяє стандартизувати і очищати дані.

Ключові особливості:

- До системи можна підключити різні джерела даних: сторонні додатки, хмарні сервіси, потокові дані, книги Excel.
- Через API можна підключити до сервісу власні додатки.
- Інтерактивні дашборда доступні на будь-яких пристроях і відображають дані в реальному часі.

- Користувачі можуть поділитися інформацією декількома доступними способами.
- Сервіс працює на всіх платформах: хмарної, настільної і мобільної.

Відмінні риси:

- Система повністю сумісна з усіма продуктами MIcrosoft: MS Excel, Azure Cloud Service і SQL Server.
- Користувачі, які коли-небудь користувалися Windows, легко освоять інтерфейс програми.
- BI-платформа спочатку створювалася для розширення функціональності MS Excel.

Версії:

- Power BI Desktop - засіб для створення звітів і гібридний додаток для візуалізації даних з безліччю функцій.
- Power BI Mobile - додаток для швидкого доступу з будь-яких пристроїв з оновленням даних в режимі реального часу.
- Power BI Service - сервіс, який дозволяє автоматично оновлювати дані і безпечно публікувати звіти для всіх користувачів.

Всю інформацію по проекту можна відстежувати на інформаційній панелі Power BI - вона доступна на всіх пристроях і відображає актуальні дані по найважливішим метрикам.

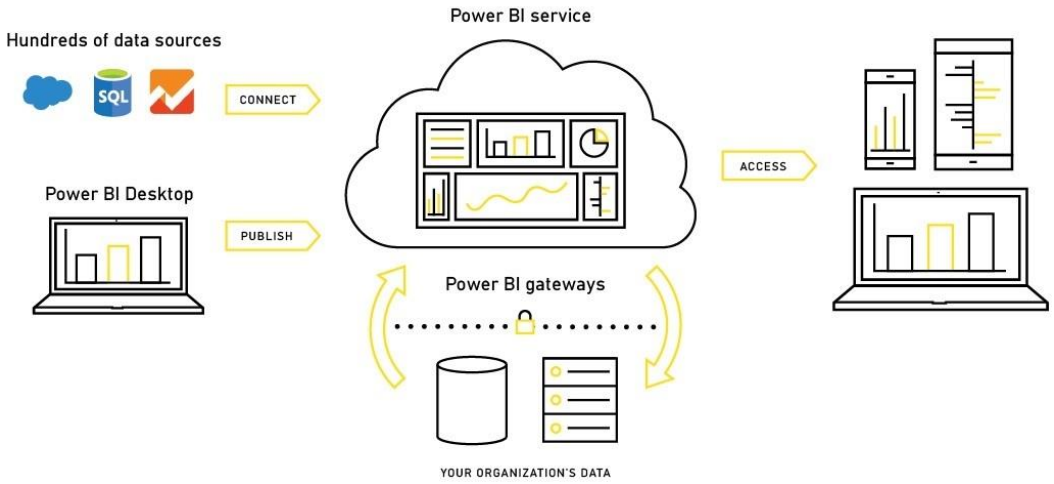


Рисунок 1.4 - Схема робота Power BI

Power BI дозволяє об'єднати всі дані компанії. До одних і тих же інформаційних панелей можна підключити безліч джерел, як локальних, так і хмарних. Наприклад, бази даних SQL Server і моделі Analysis Services.

Power BI Desktop:

- Дозволяє створити підключення до даних, колекцію запитів і звітів, до яких можна надати загальний доступ.
- У продукт вбудовані підсистема обробки запитів, система візуалізації та моделювання даних.
- За допомогою засобів візуальної аналітики можна створювати наочні, інтерактивні звіти.
- Платформа дозволяє виокремлювати цінну інформацію з масиву даних.
- Підготовка та моделювання даних займає всього кілька кроків. Деталізована аналітика використовує функції групування, прогнозування, кластеризації і швидких заходів.

Переваги:

- Можна підключити практично будь-яке офлайн або онлайн джерело даних.
- Вбудовані бібліотеки візуалізацій.
- Повна інтеграція з продуктами Microsoft.

- Підходить для широкого кола користувачів.
- Автоматичне оновлення інформації в реальному часі.

Недоліки:

- Мало інструментів для очищення й обробки даних.
- Складнощі з імпортом і обробкою великих обсягів даних.
- У користувачів, які не знайомі з MS Excel, можуть виникнути труднощі при складних операціях.
- Програма орієнтована на Excel, але часто великі компаній використовують свої сервера і не користуються даним софтом.
- При оновленні через API дані закачуються з самого початку, що уповільнює роботу програми.

РОЗДІЛ 2

НАПИСАННЯ API ЗА ДОПОМОГОЮ ФРЕЙМВОРКУ NESTJS

2.1 NestJS

Nest(NestJS) - це основа для створення ефективних, масштабованих серверних додатків Node.js. Він використовує прогресивний JavaScript, побудований та повністю підтримує TypeScript (але все ще дозволяє розробникам кодувати чистий JavaScript) і поєднує елементи OOP (об'єктно-орієнтоване програмування), FP (функціональне програмування) та FRP (функціональне реактивне програмування).

Під капотом Nest використовує надійні фреймворки HTTP Server, такі як Express (за замовчуванням), і, за бажанням, може бути налаштований на використання Fastify!

Nest забезпечує рівень абстракції над цими загальними фреймворками Node.js (Express / Fastify), але також надає їх API безпосередньо розробнику. Це надає розробникам свободу використовувати безліч сторонніх модулів, доступних для базової платформи.

Філософія

В останні роки, завдяки Node.js, JavaScript став «мовою франка» в Інтернеті як для зовнішніх, так і для бекенд-додатків. Це породило чудові проекти, такі як Angular, React та Vue, які покращують продуктивність розробників та дозволяють створювати швидкі, верифіковані та розширювані фронтенд-програми. Однак, хоча для Node існує безліч чудових бібліотек, помічників та інструментів (жоден із серверних JavaScript), жоден з них ефективно не вирішує головну проблему - Архітектуру.

Nest надає нестандартну архітектуру додатків, яка дозволяє розробникам та командам створювати високотестовані, масштабовані, вільно пов'язані та легко підтримувані програми. Архітектура сильно натхнена Angular.

Встановлення

Щоб побудувати проект за допомогою Nest CLI, треба виконати такі команди. Це створить новий каталог проекту та заповнить каталог початковими основними файлами Nest та допоміжними модулями, створивши звичайну базову структуру для проекту. Створення нового проекту за допомогою Nest CLI рекомендується для початківців.

```
$ npm i -g @nestjs/cli
$ nest new project-name
```

Рисунок 2.1 - Консольні команди для створення нового проекту

Буде створено каталог проекту, встановлені модульні вузли та кілька інших шаблонних файлів, а також створено каталог src/, який заповниться кількома основними файлами.

Рисунок 2.2 - Каталог нового проекту

Таблиця 2.1 - Файли нового проекту

app.controller.ts	Базовий контролер з одним маршрутом.
app.controller.spec.ts	Блок тестів для контролера.
app.module.ts	Кореневий модуль програми.
app.service.ts	Базовий сервіс з одним методом.
main.ts	Головний файл програми, який використовує основну функцію NestFactory для створення екземпляра програми Nest.

Main.ts включає функцію асинхронізації, яка запускає наш додаток:

```
import { NestFactory } from '@nestjs/core';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';
import { AppModule } from './app.module';
import { loggerMiddleware } from './middlewares/logger.middleware';
import { logger } from './logger';

async function bootstrap() {
  const app = await NestFactory.create(AppModule, {
    logger
  });

  const options = new DocumentBuilder()
    .setTitle('KSU-Analytics-Test')
    .setDescription('KSU-Analytics-Test API description')
    .setVersion('1.0')
    .addTag('ksu')
    .build();
  const document = SwaggerModule.createDocument(app, options);
  SwaggerModule.setup('api', app, document);

  app.use(loggerMiddleware);

  await app.listen(3000);
}
bootstrap();
```

Рисунок 2.3 - Файл Main.ts

Для створення екземпляра програми Nest використовується основний клас NestFactory. NestFactory надає кілька статичних методів,

які дозволяють створювати екземпляр програми. Метод create() повертає об'єкт програми, який відповідає інтерфейсу INestApplication. У наведеному вище прикладі main.ts просто запускається HTTP-прослуховувач, що дозволяє додатку чекати вхідних HTTP-запитів.

Проект встановлений у платформі Nest CLI, створює початкову структуру проекту, яка заохочує розробників дотримуватися узгодження про збереження кожного модуля у своєму виділеному каталозі.

Платформа

Nest прагне бути платформою-агностиком. Незалежність від платформи дозволяє створювати багаторазові логічні частини, якими розробники можуть скористатися для кількох різних типів додатків. Технічно Nest може працювати з будь-яким фреймворком Node HTTP після створення адаптера. Існує дві платформи HTTP, що підтримуються нестандартно: express та fastify. Ви можете вибрати той, який найкраще відповідає вашим потребам.

Таблиця 2.2 - HTTP Сервери

platform-express	Express - це добре відомий мінімалістичний веб-фреймворк для вузла. Це бібліотека, готова до виробництва, з великою кількістю ресурсів, впроваджених спільнотою. За замовчуванням використовується пакет @ nestjs / platform-express. Багато користувачів добре обслуговують Express, і їм не потрібно вживати жодних заходів, щоб увімкнути його.
platform-fastify	Fastify - це високопродуктивний та низький накладні витрати, який зосереджений на забезпеченні максимальної ефективності та швидкості.

Незалежно від платформи, він використовує власний інтерфейс програми. Вони розглядаються відповідно як NestExpressApplication та NestFastifyApplication.

Коли передається тип методу NestFactory.create(), як у прикладі нижче, об’єкт програми матиме методи, доступні виключно для цієї конкретної платформи. Однак треба звернути увагу, що не потрібно вказувати тип, якщо ви насправді не хочете отримати доступ до базового API платформи.

```
const app = await NestFactory.create<NestExpressApplication>(AppModule);
```

Рисунок 2.4 - Передача методу NestFactory.create()

Запуск програми

Після завершення процесу встановлення ви можете запустити таку команду в командному рядку ОС, щоб запустити програму для прослуховування вхідних HTTP-запитів:

```
$ npm run start
```

Рисунок 2.5 - Консольна команда для запуску програми

Ця команда запускає програму із прослуховуванням сервера HTTP через порт, визначений у файлі src/main.ts. Після запуску програми можна відкрити браузер і перейти до «http://localhost:3000/». На екрані буде напис «Hello World!».

Controller(Контролер)

Контролери відповідають за обробку вхідних запитів та повернення відповідей клієнту.

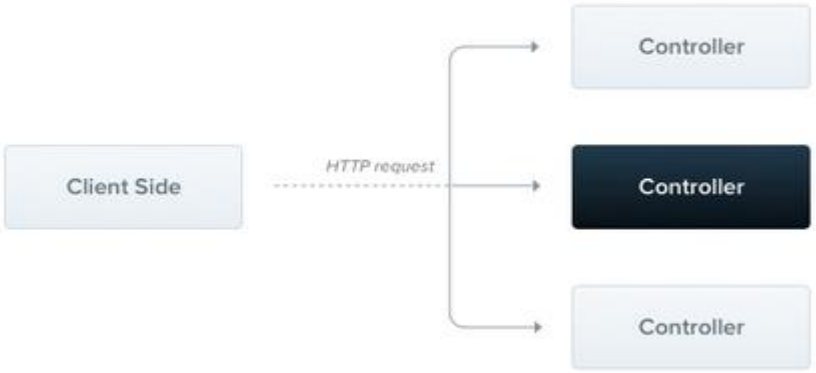


Рисунок 2.6 - Схема роботи контролера

Призначення контролера - отримувати конкретні запити на додаток. Механізм маршрутизації контролює, який контролер отримує які запити. Часто кожен контролер має більше одного маршруту, і різні маршрути можуть виконувати різні дії.

Для того, щоб створити базовий контролер, використовуються класи та декоратори. Декоратори пов'язують класи з необхідними метаданими і дозволяють Nest створювати карту маршрутів (прив'язувати запити до відповідних контролерів).

Providers(Провайдери)

Провайдер- основне поняття Nest. Багато базових класів Nest можуть розглядатися як провайдери - послуги, сховища, фабрики, помічники тощо. Основна ідея провайдера полягає в тому, що він може вводити залежності; це означає, що об'єкти можуть створювати різні взаємозв'язки між собою, а функцію "підключення" екземплярів об'єктів можна в основному делегувати системі виконання Nest. Провайдер - це просто клас, анотований за допомогою декоратора @Injectable().

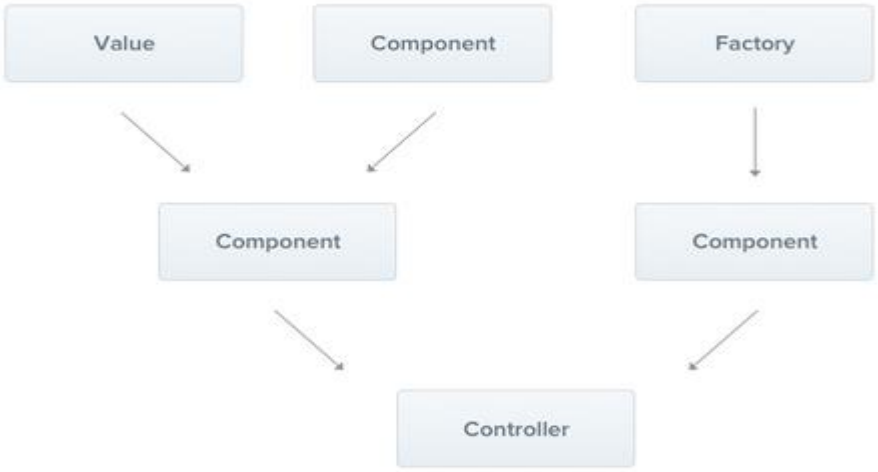


Рисунок 2.7 - Схема роботи провайдера

Контролери повинні обробляти HTTP-запити та делегувати більш складні завдання провайдерам. Провайдери - це звичайні класи JavaScript з декоратором @Injectable (), що передує їх декларації класу.

Modules(Модулі)

Модуль - це клас, котрий коментується декоратором @Module(). Декоратор @Module() надає метадані, які Nest використовує для організації структури програми.

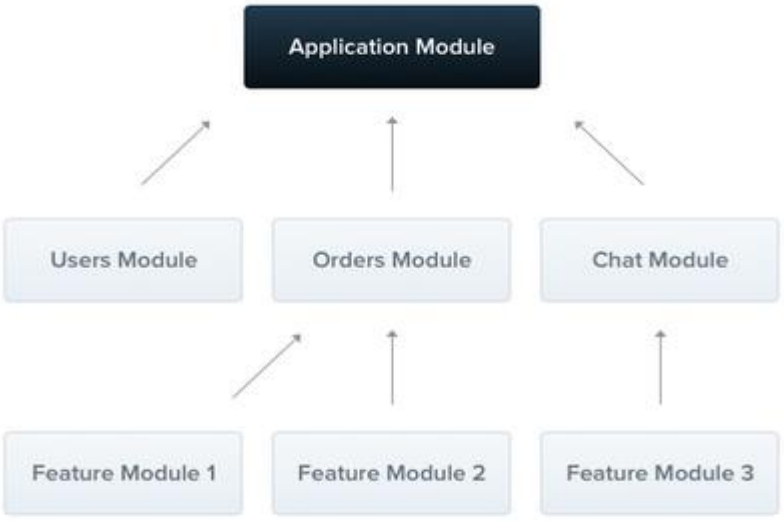


Рисунок 2.8 - Схема роботи модулей

Кожен додаток має принаймні один модуль, кореневий модуль. Кореневий модуль є відправною точкою, яку Nest використовує для побудови графу програми - внутрішньої структури даних, яку Nest використовує для взаємозв’язків модуля та провайдера. Таким чином, для більшості програм отримана архітектура використовуватиме безліч модулів, кожен з яких містить тісно пов’язаний набір можливостей.

Декоратор @Module() бере один об'єкт, властивості якого описують модуль:

Табл. 2.3 - Взаємодія модулей з іншими частинами програми

providers	провайдери, які будуть створені за допомогою інжектора Nest, і які можуть бути спільними, принаймні, у цьому модулі
controllers	набір контролерів, визначених у цьому модулі, які мають бути створені в екземплярі
imports	список імпортованих модулів, які експортують постачальників, які потрібні в цьому модулі
exports	підмножина постачальників, що надаються цим модулем, і повинна бути доступна в інших модулях, що імпортують цей модуль

Модуль інкапсулює провайдери за замовчуванням. Це означає, що неможливо вводити постачальників, які не є безпосередньо частиною поточного модуля та не експортуються із імпортованих модулів. Таким чином, можна розглядати експортовані провайдери з модуля як загальнодоступний інтерфейс модуля або API.

Middleware(Проміжне програма)

Middleware – це функція, яка викликається перед обробником маршруту. Функції проміжного програмного забезпечення мають доступ до об’єктів запиту та відповіді та next() функції Middleware у циклі запиту-відповіді програми. Наступна функція Middleware зазвичай позначається змінною з іменем next.

Рисунок 2.9 - Схема роботи Middleware

Проміжне програмне забезпечення Nest за замовчуванням еквівалентно експрес-проміжному ПЗ. Далі описується можливості проміжного програмного забезпечення:

Функції проміжного програмного забезпечення можуть виконувати такі завдання:

- виконати будь-який код.
- внести зміни до запиту та об’єктів відповіді.
- закінчити цикл запит-відповідь.
- викликати наступну функцію Middleware в стеку.

якщо поточна функція проміжного програмного забезпечення не закінчує цикл запиту-відповіді, вона повинна викликати next(), щоб передати керування наступній функції Middleware. В іншому випадку запит буде залишено повішеним.

Exception filters(Фільтри винятків)

Nest постачається із вбудованим шаром винятків, який відповідає за обробку всіх необроблених винятків у програмі. Коли виняток не обробляється кодом вашої програми, він перехоплюється цим шаром, який потім автоматично надсилає відповідну зручну відповідь.

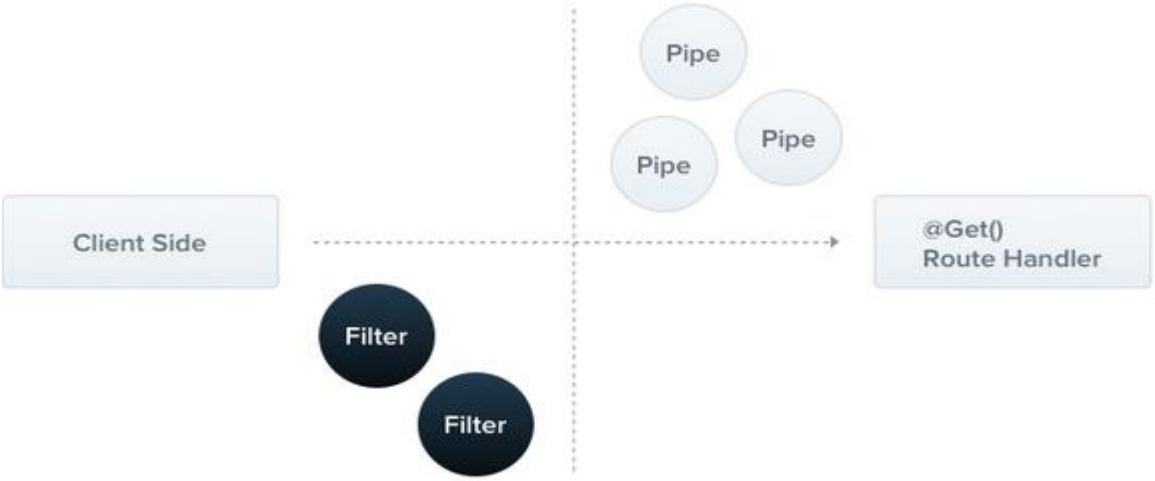


Рисунок 2.10 - Схема роботи фільтрів

Звичайно цю дію виконує вбудований глобальний фільтр винятків, який обробляє винятки типу HttpException (та його підкласи). Коли виняток не розпізнано (не є ні HttpException, ні класом, який успадковується від HttpException), вбудований фільтр винятків генерує таку відповідь JSON за замовчуванням:

```
{
  "statusCode": 500,
  "message": "Internal server error"
}
```

Рисунок 2.11 - Виняток не розпізнано

Pipes(труба)

Pipe - це клас, котрий коментується декоратором @Injectable(). Pipes повинні реалізовувати інтерфейс PipeTransform.

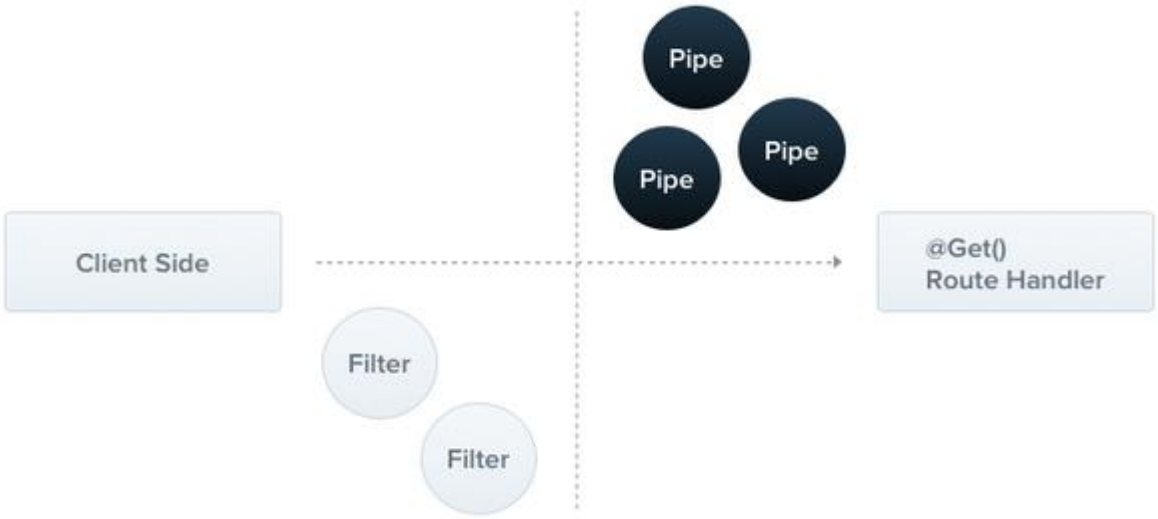


Рисунок 2.12 - Схема роботи труб

Труби мають два типові випадки використання:

- **transformation**: перетворення вхідних даних у бажану форму (наприклад, із рядка в ціле число);
- **validation**: оцінює вхідні дані і, якщо вони дійсні, просто передає їх без змін; в іншому випадку видає виняток, коли дані неправильні.

В обох випадках канали працюють на аргументах, що обробляються обробником маршрутизації контролера. Nest вставляє трубу безпосередньо перед викликом методу, і труба отримує аргументи, призначені для методу, і оперує ними. Будь-яка операція перетворення або перевірки відбувається в той час, після чого обробник маршруту викликається з будь-якими (потенційно) перетвореними аргументами.

Interceptors(Перехоплювачі)

Interceptors - це клас, котрий коментується декоратором @Injectable (). Перехоплювачі повинні реалізовувати інтерфейс NestInterceptor.

Рисунок 2.13 - Схема роботи перехоплювачів

Interceptors мають набір корисних можливостей, натхненних технологією Аспектно-Орієнтованого Програмування (AOP). Вони дають можливість:

- прив’язати додаткову логіку до / після виконання методу;
- перетворює результат, повернутий із функції;
- перетворити виняток, викинутий із функції;
- розширити основні функції поведінки;
- повністю замінити функцію залежно від конкретних умов.

Кожен interceptor реалізує метод intercept(), який приймає два аргументи. Перший - екземпляр ExecutionContext. ExecutionContext успадковує від ArgumentsHost. Там ми побачили, що це обгортка навколо аргументів, які були передані вихідному обробнику, і містить різні масиви аргументів залежно від типу програми.

2.2 База даних

База даних була спроектована та написана на мові MSSQL. Вона налічує в собі інформацію про студентів, їх форму навчання, фінансування факультетів, форму оплати, степінь навчання. Далі можна побачити всі таблиці:

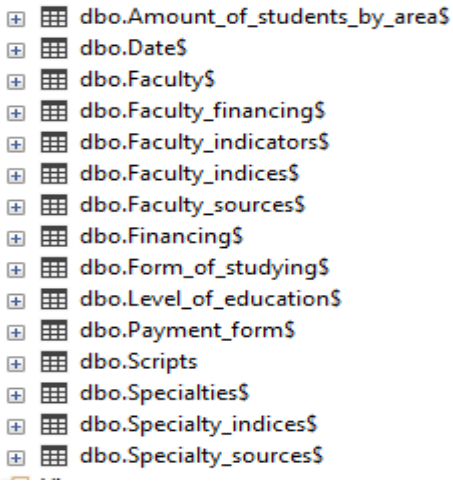


Рисунок 2.14 - Таблиці в базі даних

Робота з базою даних проходила за допомогою додатку Microsoft SQL Server Management Studio.

SQL Server Management Studio (SSMS) - це інтегроване середовище для управління будь-інфраструктурою SQL. Використовуйте SSMS для доступу, налаштування, адміністрування та розробки всіх компонентів SQL Server, Бази даних SQL Azure і Azure Synapse Analytics, а також управління ними. Середа SSMS надає єдину повнофункціональну службову програму, яка поєднує в собі велику групу графічних інструментів з рядом відмінних редакторів сценаріїв для доступу до служби SQL Server для розробників і адміністраторів баз даних всіх професійних рівнів.

Для зв’язку бази даних на SSMS та проекту на NestJS використовувався TypeORM.

TypeORM, безумовно, є найкращим Object Relational Mapper (ORM), доступним для node.js. Оскільки він написаний на TypeScript, він досить добре працює з фреймворком Nest.

Щоб розпочати роботу з цією бібліотекою, потрібно використати наступну команду:

```
npm install --save typeorm mssql
```

Рисунок 2.15 - Консольна команда для інсталяції TypeORM

Після установки потрібно створити конфіг по якому можна буде підключитися до бази відкритої на SSMS:

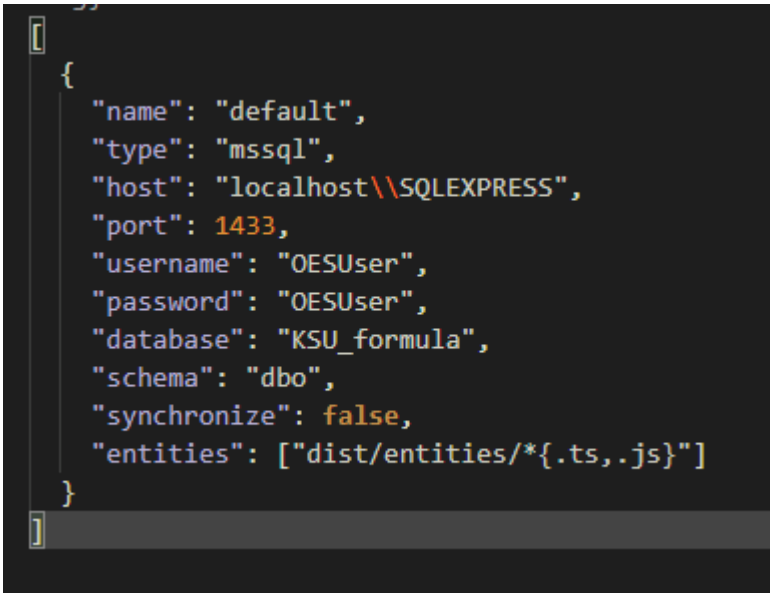


Рисунок 2.16 - Конфіг для зв’язку з базою даних

Далі потрібно створити об’єкти бази даних з якими можна буде працювати в проекті на NestJS, ці об’єкти називають entitie. Далі наведено приклад одного такого об’єкта який був створений на основі таблиці Amount_of_students_by_area:

Див. у додатку (А).

2.3 Swagger

Специфікація OpenAPI - це мовно-агностичний формат визначення, що використовується для опису RESTful API. Nest надає спеціальний модуль, який дозволяє генерувати таку специфікацію, використовуючи декоратори.

Для початку потрібно використати наступну команду та інсталювати потрібні залежності:

```
npm install --save @nestjs/swagger swagger-ui-express
```

Рисунок 2.17 - Консольна команда для інсталяції SwaggerAPI

Після завершення процесу встановлення в файл main.ts потрібно ініціалізувати Swagger, використовуючи клас SwaggerModule:

```
const document = SwaggerModule.createDocument(app, options);
SwaggerModule.setup('api', app, document);
```

Рисунок 2.18 - Ініціалізація Swagger в файлі main.ts

DocumentBuilder допомагає структурувати базовий документ, який відповідає специфікації OpenAPI. Він надає декілька методів, що дозволяють встановлювати такі властивості, як заголовок, опис, версія тощо. Для того, щоб створити повний документ (із визначеними усіма маршрутами HTTP), використовується метод createDocument() класу SwaggerModule. Цей метод приймає два аргументи, екземпляр програми та об’єкт параметрів Swagger. Крім того, можна надати третій аргумент, який повинен мати тип SwaggerDocumentOptions. Детальніше про це в розділі Параметри документа.

Після створення документа можна викликати метод setup(). Він приймає:

- шлях до інтерфейсу Swagger;
- екземпляр програми;
- об'єкт документа, згаданий вище.

Далі можна побачити зображення розробленого у рамках цієї роботи Swagger'a. Він налічує в собі по п’ять до декількох таблиць із вище представленої бази даних. Після запуску програми потрібно перейти по посиланню: «http://localhost:3000/api».

Див. у додатку (Б).

2.4 Робота с Power BI Desktop

Завантажити додаток можна на офіційному сайті «https://powerbi.microsoft.com/ru-ru/».

Було обрано саме Power Bi Desktop, тому що після підключення джерела даних можна переглянути модель бази даних та налаштувати її під свої потреби. Також різноманіття графічних інструментів дозволить візуалізувати необхідні дані.

Див. у додатку (В)

Після запуску додатку потрібно підключитися до бази даних та додати таблиці з інформацією до проекту. У верхньому лівому кутку потрібно натиснути кнопку «Get data».

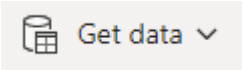


Рисунок 2.19 - Кнопка «Get data»

Відкриється вікно де потрібно вибрати джерело інформації, в даній роботі це база даних написана на MS SQL, розташована на СКБД(система керування базами даних) Microsoft SQL Server.

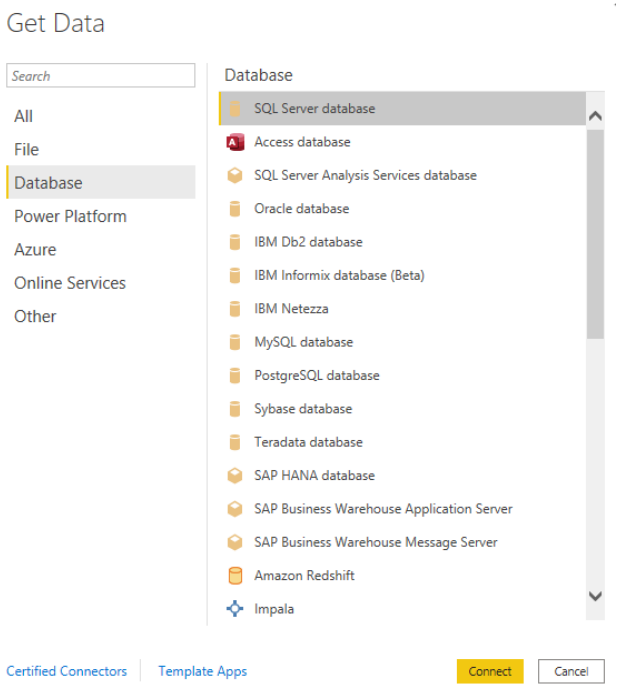


Рисунок 2.20 - Вікно вибору джерела інформації

Далі потрібно ввести дані о базі даних, для того щоб отримати доступ до інформації.

Рисунок 2.21 - Заповнення реквізитів бази даних

По закінченню потрібно вибрати таблиці, які будуть фігурувати в майбутній візуалізації, можна вибрати всі таблиці, навіть якщо деякі не будуть використовуватися. При завантаженні із SQL Server зберігаються зв’язки між таблицями.

Тепер можна переглянути модель бази даних, кожну таблицю окремо, та всю доступну в базі інформацію.

Див. у додатку (В)

Створення звіту починається з вибору типу діаграми та таблиць з яких будуть завантажуватися данні. При створенні звіту модна використовувати зрізи та фільтри для візуального відображення конкретної інформації із основної маси.

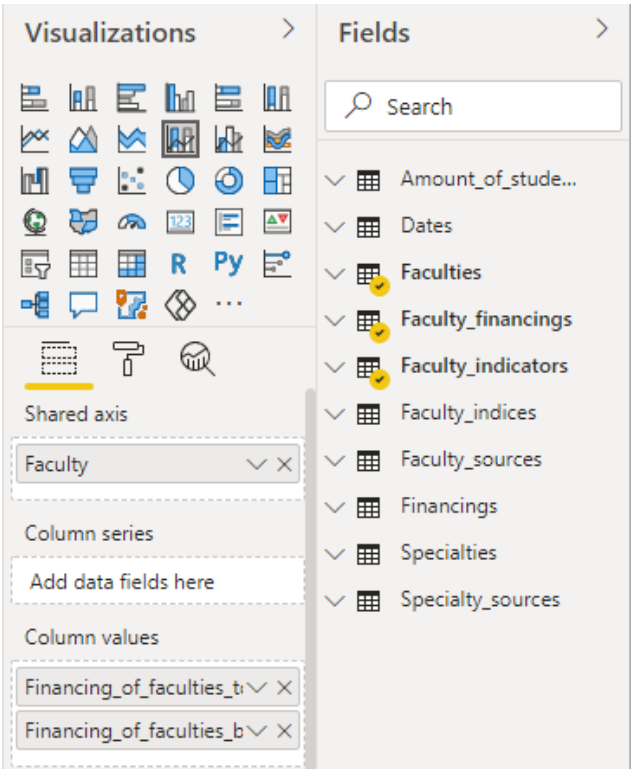


Рисунок 2.22 - Вікно візуалізації

Створений звіт на основі бази Економічної діяльності університету.

Див. у додатку (Г).

ВИСНОВОК

У ході кваліфікаційної роботи були виконані наступні завдання:

- Дослідженні бізнес-процеси в університеті, на прикладі системи EPC, яка була представлена та детально розібрана.
- Був проведений огляд трьох BI-систем Tableau, Qlik та Power BI, та вибрана найбільш зручна для реалізації завдання.
- Було розпочато розробку на основі фреймворку NestJS, на мові TypeScript. Були оглянуті основи фреймворку та розроблені моделі бази даних економічної діяльності університету.
- Підключено до проекту на NestJS, модуля Swagger API, та написання методів для взаємодії з об’єктами бази даних економічної діяльності університету. Була розроблена більша частина API.
- Успішно підключена база даних до сервісу візуалізації інформації Power BI Desktop, та створення звіту по даним із бази даних економічної діяльності університету

Можна вважати, що мета даної роботи була повністю виконана. Робоча версія програми та графічний звіт існує на комп’ютері виконавця роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Концепція реформування економічної діяльності у сфері вищої освіти України.(2017). search.ligazakon.ua.fromhttp://search.ligazakon.ua/l_doc2.nsf/link1/NT4045.html
2. Kwiek, M. (2017). De-privatization in higher education: a conceptual approach. Higher Education, 74 (2) , 259—281. https://link.springer.com/article/10.1007/s10734-016-0047-3
3. Polster, C. (2005, August 31). Privatizing Canada's Public Universities. Canadian Politics, Vol. 39, 5. https://canadiandimension.com/articles/view/privatizing-canadas-public-universities-claire-polster
4. Abate, A. S., & Lakshmipathi, R. (2009, October 9). The Role of Private Financing in Higher Education and the Changing Trends. Retrieved. fromhttp://dx.doi.org/10.2139/ssrn.1485993
5. Бухгалтерська програма «1С: бухгалтерія» http://1c.ua/ua/
6. NestJS документація https://docs.nestjs.com/
7. Typeorm-model-generator https://www.npmjs.com/package/typeorm-model-generator
8. Що таке SQL Server Management Studio (SSMS)? https://docs.microsoft.com/ru-ru/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15
9. Swagger https://swagger.io/
10. Аналіз бізнес-процесів на прикладі вищого навчального закладу https://www.researchgate.net/publication/256196756

11. BI-системи
https://www.uplab.ru/blog/review-systems-for-creating-dashboards/
12. Power Bi Desktop
https://powerbi.microsoft.com/ru-ru/desktop/
13. BI-Системи (2)
https://asu-analitika.ru/20-samyh-populjarnyh-instrumentov-biznes-analitiki-bi
14. API
https://docs.microsoft.com/ru-ru/azure/architecture/best-practices/api-design
15. API (2)
https://docs.microsoft.com/ru-ru/azure/media-services/latest/media-services-apis-overview
16. Моделі даних
https://poznayka.org/s41543t1.html
17.Моделі даних (2)
https://geoknigi.com/book_view.php?id=589
18.Node.js/documentation
https://nodejs.org/uk/docs/
19.TypeScript/documantation
https://www.typescriptlang.org/docs/
20.Git/documentatiotn
https://git-scm.com/doc

ДОДАТКИ

Додаток А

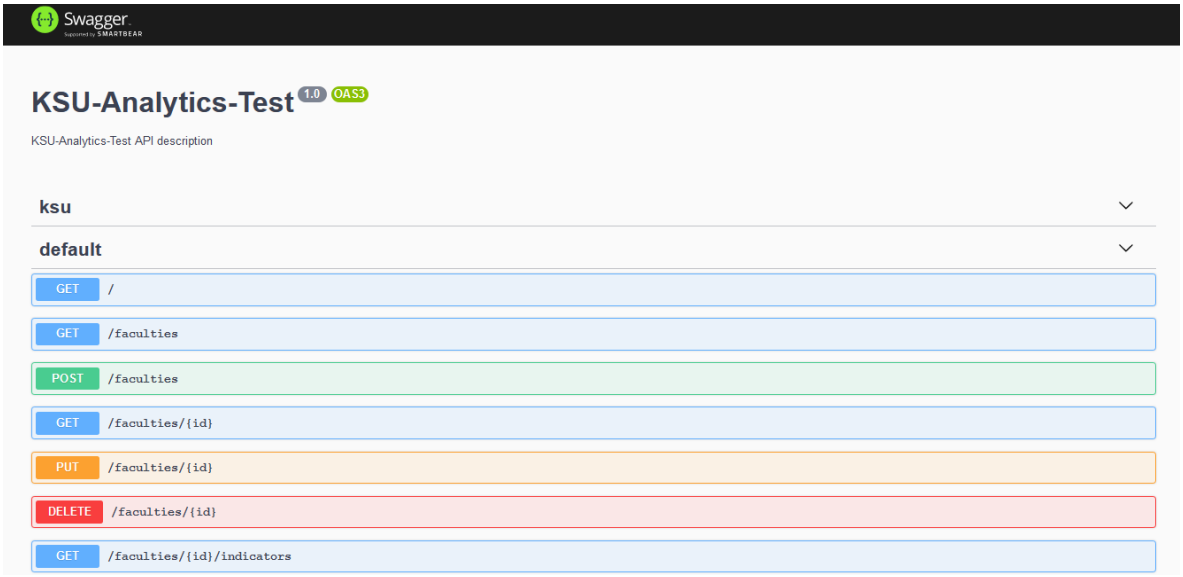
Об’єкт бази даних створений по таблиці Amount_of_sudents_by_area:

```
import {
  Column,
  Entity,
  Index,
  JoinColumn,
  ManyToOne,
  PrimaryGeneratedColumn,
} from "typeorm";
import { DateEntity } from "./Date";
import { Specialties } from "./Specialties";
import { PaymentForm } from "./PaymentForm";
import { FormOfStudying } from "./FormOfStudying";
import { LevelOfEducation } from "./LevelOfEducation";

@Index("PK__Amount_o__3213E83F7D92F72C", ["id"], { unique: true })
@Entity("Amount_of_students_by_area$", { schema: "dbo" })
export class AmountOfStudentsByArea {
  @Column("int", { name: "Amount", nullable: true })
  amount: number | null;

  @PrimaryGeneratedColumn({ type: "int", name: "id" })
  id: number;

  @ManyToOne(() => DateEntity, (date) => date.amountOfStudentsByAreas)
  @JoinColumn([{ name: "Date", referencedColumnName: "id" }])
  date: DateEntity;
```

```
28    @ManyToOne(
29      () => Specialties,
30      (specialties) => specialties.amountOfStudentsByAreas
31    )
32    @JoinColumn([{ name: "Specialty", referencedColumnName: "id" }])
33    specialty: Specialties;
34
35    @ManyToOne(
36      () => PaymentForm,
37      (paymentForm) => paymentForm.amountOfStudentsByAreas
38    )
39    @JoinColumn([{ name: "Payment_form", referencedColumnName: "id" }])
40    paymentForm: PaymentForm;
41
42    @ManyToOne(
43      () => FormOfStudying,
44      (formOfStudying) => formOfStudying.amountOfStudentsByAreas
45    )
46    @JoinColumn([{ name: "Form_of_studying", referencedColumnName: "id" }])
47    formOfStudying: FormOfStudying;
48
49    @ManyToOne(
50      () => LevelOfEducation,
51      (levelOfEducation) => levelOfEducation.amountOfStudentsByAreas
52    )
53    @JoinColumn([{ name: "Level_of_education", referencedColumnName: "id" }])
54    levelOfEducation: LevelOfEducation;
55  }
```

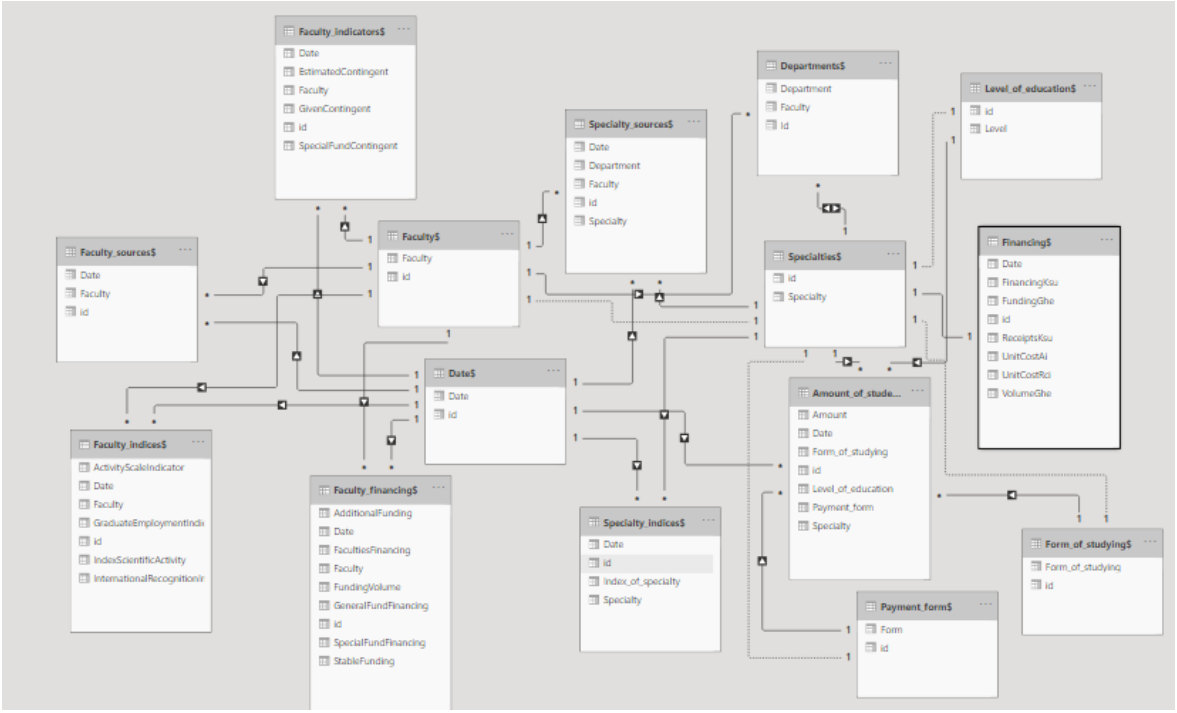
Додаток Б

Swagger створений у рамках роботи:



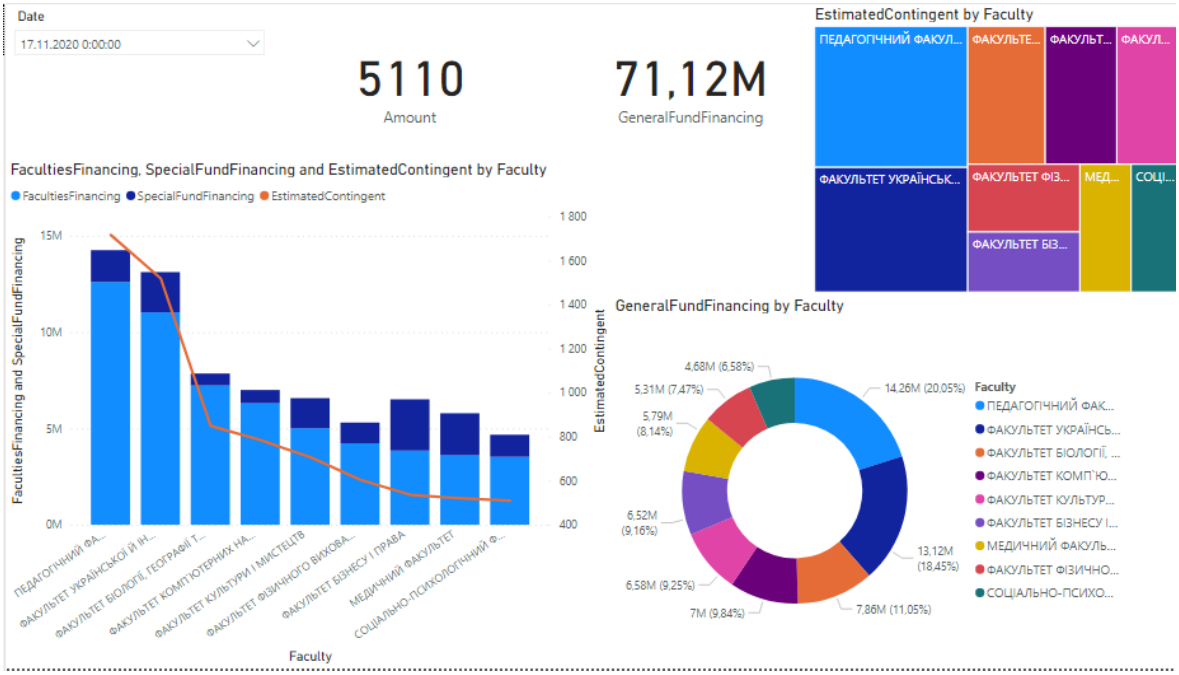
Додаток В

Модель бази даних економічної діяльності університету.



Додаток Г

Звіт зроблений у Power BI Desktop під час виконання роботи:



Додаток Ґ

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Цуркан Антон Федорович, учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

- не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;
- не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

_____	_____	_____
(дата)	(підпис)	(ім'я, прізвище)