Київський національний торговельно-економічний університет

Кафедра програмної інженерії та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка інформаційно-аналітичної моделі діяльності деканату (модуль моніторингу активності користувачів)»

Студента 2м курсу, 5 групи, спеціальності 121«Інженерія програмного забезпечення», спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Слобожаніна Іллі Івановича

Науковий керівник кандидат технічних наук, доцент _____ підпис керівника — Харченко Олександр Анатолійович

Гарант освітньої програми доктор технічних наук, професор _____ підпис гаранта — Криворучко Олена Володимирівна

КИЇВ – 2019

ЗМІСТ

ВСТУП

Розробка модуля моніторингу активності користувачів інформаційно-аналітичної моделі діяльності деканату.-Слобожанін І. 2 к 5м гр.

Актуальність: на сьогоднішній день одними з найбільш широко використовуваних компонентів стека технологій, що використовуються при розробці веб-додатків, є Spring Framework та СУБД PostgreSQL. Знання даних технологій, і розуміння принципів і патернів, що лежать в їх основі широко затребувані на ринку праці. Хоча у відкритому доступі є велика кількість матеріалів за даними технологіями, досить складних опенсорсний проектів з їх використанням вкрай мало, в зв'язку з чим їх практичне вивчення ускладнене.

Об'єкт дослідження: освітня діяльність деканату університету.

Предмет дослідження: розробка пропозицій щодо розробки модуля моніторингу активності користувачів інформаційно-аналітичної моделі діяльності деканату.

Мета роботи: розробити модуль моніторингу активності користувачів інформаційно-аналітичної моделі діяльності деканату.

Задачі дослідження: розробити модуль моніторингу активності користувачів, які дають змогу користувачеві використовувати функціонал програмного забезпечення.

Методи і технології розробки: аналіз технологій для створення веб-додатків, вибір підходящої мови програмування, аналіз існуючих підходів до розробки веб-додатків та ентерпрайз систем, вибір технологій та кращих алгоритмів для доступу до баз даних, технологій для відображення даних у графічному вигляді з використанням браузерів.

Результат роботи: Створення модуля моніторингу активності користувачів інформаційно-аналітичної моделі діяльності деканату.

РОЗДІЛ 1. ОГЛЯД ОРГАНІЗАЦІЇ ОСВІТНЬОГО ПРОЦЕСУ В КНТЕУ ТА СИСТЕМИ УПРАВЛІННЯ У ДІЯЛЬНОСТІ ДЕКАНАТУ

1.1. Принципи побудови систем управління у діяльності деканату

Для забезпечення ефективного функціонування деканату важливе значення має побудова організаційно-функціональної структури управління.

Організаційно-функціональна структура управління - це впорядкована система управлінських ланок, розташованих у строгому підпорядкуванні, що забезпечують взаємозв'язок між керуючою і керованою підсистемами, розвиток системи як єдиного цілого. Ланки управління утворюють структуру управління з конкретним розташуванням, співвідношенням і взаємозв'язком між собою. Створення організаційної структури управління зумовлене необхідністю розподілу прав і обов'язків між окремими підрозділами організації.

Організаційна структура управління - одне з ключових понять у менеджменті необхідне для визначення чіткого взаємозв'язку і взаємодії структурних ланок в управлінському процесі і коригування їхнім функціональним процесом у досягненні цілей організації. Організаційна структура деканату формується для забезпечення його перспективної конкурентноздатності, економічної ефективності, раціональної кооперації.

Представлена організаційно-функціональна структура управління у вигляді схеми, що відображає взаємозв'язки структурних підрозділів деканату, згідно виконання функцій в управлінні університету. Ця схема відображає розташування кожної служби і посади у загальній організації деканату, ілюструє розподіл повноважень і обов'язків персоналу, необхідна для ефективного виконання ключових функцій працівниками, визначення їх підзвітності.

Управління в деканаті пов'язується з оптимальним розподілом цілей і завдань між структурними ланками (службами, змінами, бригадами та ін.) і кожним

окремим працівником організації. Структура управління, таким чином, визначається розподілом органів управління деканату, характером їхньої спеціалізації - завданнями управління і формами координації їхньої діяльності.

Візуально організаційна структура управління деканату відображається графічним способом у вигляді двовимірної схеми. Схема фіксує у компактній формі інформацію про ієрархічність, повноваження і підпорядкування рівнів управління.

Організаційна схема управління деканату створюється керівниками університету на початковій стадії його виникнення і визначається спеціалізацією закладу, його категорією, обсягом студентів, розташуванням навчальних аудиторій та корпусів, кількістю бюджетних та контрактних місць, кількістю навчальних дисциплін та іншими факторами. У створенні організаційної схеми вагому роль відіграє аналіз створення підрозділів управління, чіткого визначення їхніх функціональних обов'язків, зв'язків у самому підрозділі (службі, відділі) та між підрозділами у навчальному процесі. Затверджується структура управління Статутом університету і документально оформляється спеціальним положенням про про внутрішній розпорядок, посадовими інструкціями всіх рівнів управління.

Організаційна структура управління деканатом є оптимальною тоді, коли всі структурні підрозділи доповнюють навчальний процес та забезпечують його функціонування, водночас вони забезпечують максимальну ефективність функціонування у досягненні кінцевого результату, яким вважається надання конкурентоздатних послуг.

Серед них виділяються:

- велика кількість робочих місць, на яких періодично змінюється комп'ютерна техніка та перевстановлюються операційні системи;
- «розкиданість» корпусів навчального закладу (а, отже, і користувачів системи);
- інформація створюється у системі невеликою кількістю «активних» операторів. Користуватись же цією інформацією має широке коло її

споживачів (керівний склад, працівники деканатів та кафедр, викладачі та студенти – кожний в межах своїх прав).

Головною метою виконання теми було:

- аналіз відомих на сьогодні інформаційних систем управління навчальним процесом, критеріїв і принципів їх розробки, впровадження, експлуатації та концепції розвитку;
- розробка концептуальної моделі автоматизованої системи управління навчальним процесом, визначення видів та обсягів вхідних та вихідних даних;
- систематизувати і класифікувати існуючий перелік планувальної і обліково-контролюючої документації на основі форм, рекомендованих Міністерством освіти і науки України та з урахуванням вимог і стандартів ECTS;
- розробити та узгодити форми документів для обліку та звітності внутрішньокафедральних та внутрішньофакультетських видів діяльності (навчально-методична, наукова, організаційно-виховна, профорієнтаційна та ін.);
- здійснити практичну розробку програмного продукту для автоматизації управління навчальним процесом сумісного з існуючим апаратним і програмним забезпеченням з можливістю використання програми, як на окремому взятому комп'ютері, так і в локальній загальноуніверситетській мережі. Передбачити заходи захисту інформації від несанкціонованого доступу та від непередбачених втрат інформації: пошкоджень програми та інформаційної бази вірусами й іншими негативними факторами;
- провести дослідну перевірку програмної розробки в умовах наповнення бази реальними даними.

- Здійснити аналіз вихідних даних на основі отриманих облікових, обліково-контролюючих і звітних форм документів.
- розробити пропозиції і рекомендації щодо впровадження системи автоматизації навчального процесу на всіх факультетах КНТЕУ.

Організаційна схему управління повинна постійно змінюватись, зокрема при динамічному зовнішньому середовищі деканату і структурі управління. Відповідно до змін у структурі управління, які зумовлюють зміни у чисельності персоналу, важливо, щоб ці зміни суттєво не позначались на якості навчання. Персонал у деканаті завжди повинен взаємодоповнюватись, бути взаємозамінним у певному структурному підрозділі.

1.2. Особливості роботи деканату

Мається на увазі, що інформація буде зміняться і поповняться протягом терміну навчання. Ведення архіву студентів. Після закінчення терміну навчання інформація про студентів може ще деякий час бути необхідною (в паперових архівах університету вона зберігається протягом 75 років), тому необхідно щоб студенти закінчили навчання заносилися в архів і зберігалися в ньому до прийняття рішення про їх відрахування. Поповнення списку новими абітурієнтами. На початку кожного навчального року в базу даних повинні заноситися студенти з числа абітурієнтів надійшли в поточному році. Переведення факультету на наступний курс. Після закінчення навчального року інформація про поточний курс кожного студента повинна бути змінена тобто студент повинен бути переведений на наступний курс в разі успішного завершення сесії. Ет і зміни повинні торкатися лише особистостей з дійсного числа студентів і не торкатися знаходяться в архіві. Крім того, студенти вважаються закінчили навчання повинні бути автоматично занесені в архів. Розподіл груп за спеціальностями.

З цим завданням стикаються після здачі держіспиту на 4 курсі, коли вже сформовані групи знову перерозподіляються в залежності від обраної студентами спеціальності. Розподіл мови по групах. На першому курсі розподіл студентів по

групах ведеться в залежності від досліджуваної іноземної мови. Протягом навчання студенту може бути надано академічну відпустку на один рік після його закінчення студент продовжує навчання. Протягом цього часу інформація про студента повинна зберігатися в архіві, щоб бути затребуваною при відновленні. Крім того іноді потрібна інформація про дату виходу студента в академічну відпустку і номері наказу за яким академічну відпустку було надано. Відрахувння і відновлення. На будь-якому курсі студент може бути відрахований за низкою причин. Однак факт відрахування не носить фатальний характер і в ряді випадків у нього є можливість відновитися. Тому аж до факту відновлення інформація повинна зберігатися в архіві поки не буде затребуваною або не буде ухвалено рішення про недоцільність її зберігання. У разі відрахування також потрібна інформація про дату відрахування студента та номер наказу за яким відрахування відбулося. Перевод в іншу групу / підгрупу. Це завдання часто має місце на першому курсі коли студенти виявляють бажання перейти в іншу групу або підгрупу коли остаточний склад групи ще повністю не сформувався. Надання інформації про навчальний план.

Дисципліни за якими проводиться навчання в групах в загальноосвітній період і після вибору спеціалізації повинні бути точно представлені. За ним ведуться відомості про підсумки сесії для кожного студента. Ведення інформації про підсумки сесії і проводяться атестацій. В період навчання кожен студент вивчає дисципліни зазначені в навчальному плані і, отже повинен проходити контроль знань по ним в кінці кожного семестру. Крім того, в середині семестру проводиться додатковий контроль знань по системі відрізняється від екзаменаціонного. Отримання статистичної інформації про факультет. Для аналізу, планування і прийняття рішень щодо подальшого розвитку факультету корисно мати інформацію представлену в узагальненому вигляді (підсумкових сум, таблиць, графіків), яку можна отримати з інформації про підсумки проведених сесій, атестацій і т.д. (кількісна та якісна успішність, кількість боржників з предметів, кількість склали іспити і т.д.

1.3. Автоматизація діяльності деканату

Однією з найважливіших задач, які стоять перед ЗВО на сучасному етапі розвитку, постає ефективна організація процесів управління. Широкі можливості для вирішення цієї задачі надають сучасні інформаційні технології. В більшості ЗВО використовуються різноманітні інформаційні системи для автоматизації конкретних сфер діяльності. Але дуже часто вони не согласовані між собою, мають різних виробників, різний формат даних і зрозумілий апарат. Виникає питання інтеграції цих систем в єдину інформаційну інфраструктуру, що дозволить ефективно керувати діяльністю, використовуючи сучасні методи планування і аналізу стану навчального процесу, забезпечить оперативність і обоснованість прийняття рішень, потребу в інформаційній взаємодії між підрозділами ЗВО.

Однією з найважливіших умов успішного функціонування будь-якого ЗВО є ефективний обмін інформації між підрозділами. Зараз, не дивлячись на достатньо хорошу інформатизацію більшості деканатів, процеси інформаційного обміну в більшості ЗВО неоптимальні.

Робітникам деканатів необхідно виконувати великий об’єм рутинної роботи по обліку контингенту студентів, забезпеченню навчального процесу, наданню інформації в різноманітні підрозділи ЗВО. При чому всю інформацію необхідно надавати в різних форматах. Це не зручно, тому що необхідно вносити зміни відразу в дві або більше систем, при цьому виникає вірогідність допущення помилок, що призводить до некоректної діяльності не лише деканата, а й відділів, які з ним взаємодіють.

1.4. Висновки та пропозиції до розділу 1

В першому розділі було розглянуто організацію освітнього процесу в КНТЕУ, визначено за якими формами здійснюється навчання (очна, заочна). Освітній процес у КНТЕУ здійснюється за такими формами: навчальні заняття, самостійна робота, практична підготовка, контрольні заходи, що передбачає різні види його контролю.

Досліджено організаційну структуру університету та визначено, що в КНТЕУ навчається більше 40 тис. студентів. Функціонують 6 факультетів: міжнародної торгівлі та права; економіки, менеджменту та психології; фінансів та банківської справи; обліку, аудиту та інформаційних систем; ресторанно-готельного та туристичного бізнесу; торгівлі та маркетингу.

Проведено аналіз наявних програмних засобів управління в діяльності деканату та визначено, що хоч і існує програмний продукт, який автоматизує управління навчальним процесом, але він має ряд недоліків. Тому було прийнято рішення розробити програмне забезпечення, яке б не мало існуючих недоліків та дозволяло оптимізувати процес управління навчальним процесом.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ НА ОСНОВІ ФРЕЙМВОРКУ SPRING ТА СУБД PostgreSQL

2.1. Постановка задачі про проектуванню системи

Мета – створення бази даних з веб-додатком для системи управління деканатом та системи моніторингу активності користувачів

Вхідні дані:

- прототип, з описом технології, організації розробки
- методичні матеріали
- програмне забезпечення

Вихідні дані:

- фізична та логічна модель даних
- схема даних
- база даних
- веб-додаток

2.2. Проектування модуля моніторингу активності користувачів

Для забезпечення ефективного функціонування аналітичної системи діяльності деканату та модулю моніторингу активності користувачів важливе значення має побудова організаційно-функціональної структури управління.

Організаційно-функціональна структура управління - це впорядкована система управлінських ланок, розташованих у строгому підпорядкуванні, що забезпечують взаємозв'язок між керуючою і керованою підсистемами, розвиток системи як єдиного цілого. Ланки управління утворюють структуру управління з конкретним розташуванням, співвідношенням і взаємозв'язком між собою. Створення організаційної структури управління зумовлене необхідністю розподілу прав і обов'язків між окремими підрозділами організації.

Організаційна структура управління - одне з ключових понять у менеджменті необхідне для визначення чіткого взаємозв'язку і взаємодії структурних ланок в управлінському процесі і коригування їхнім функціональним процесом у досягненні цілей організації. Організаційна структура деканату формується для забезпечення його перспективної конкурентноздатності, економічної ефективності, раціональної кооперації.

Представлена організаційно-функціональна структура управління у вигляді схеми, що відображає взаємозв'язки структурних підрозділів деканату, згідно виконання функцій в управлінні університету. Ця схема відображає розташування кожної служби і посади у загальній організації деканату, ілюструє розподіл повноважень і обов'язків персоналу, необхідна для ефективного виконання ключових функцій працівниками, визначення їх підзвітності.

Управління в аналітичній системі пов'язується з оптимальним розподілом цілей і завдань між структурними ланками (службами, змінами, бригадами та ін.) і кожним окремим працівником організації. Структура управління, таким чином, визначається розподілом органів управління деканату, характером їхньої спеціалізації - завданнями управління і формами координації їхньої діяльності.

Візуально організаційна структура управління деканату відображається графічним способом у вигляді двовимірної схеми. Схема фіксує у компактній формі інформацію про ієрархічність, повноваження і підпорядкування рівнів управління.

Організаційна схема управління деканату створюється керівниками університету на початковій стадії його виникнення і визначається спеціалізацією закладу, його категорією, обсягом студентів, розташуванням навчальних аудиторій та корпусів, кількістю бюджетних та контрактних місць, кількістю навчальних дисциплін та іншими факторами. У створенні організаційної схеми вагому роль

відіграє аналіз створення підрозділів управління, чіткого визначення їхніх функціональних обов'язків, зв'язків у самому підрозділі (службі, відділі) та між підрозділами у навчальному процесі.

Затверджується структура управління Статутом університету і документально оформляється спеціальним положенням про про внутрішній розпорядок, посадовими інструкціями всіх рівнів управління.

Організаційна структура управління деканатом є оптимальною тоді, коли всі структурні підрозділи доповнюють навчальний процес та забезпечують його функціонування, водночас вони забезпечують максимальну ефективність функціонування у досягненні кінцевого результату, яким вважається надання конкурентоздатних послуг.

Серед них виділяються:

- велика кількість робочих місць, на яких періодично змінюється комп'ютерна техніка та перевстановлюються операційні системи;
- «розкиданість» корпусів навчального закладу (а, отже, і користувачів системи);
- інформація створюється у системі невеликою кількістю «активних» операторів. Користуватись же цією інформацією має широке коло її споживачів (керівний склад, працівники деканатів та кафедр, викладачі та студенти – кожний в межах своїх прав).
- Головною метою виконання теми було:
- аналіз відомих на сьогодні інформаційних систем управління навчальним процесом, критеріїв і принципів їх розробки, впровадження, експлуатації та концепції розвитку;
- розробка концептуальної моделі автоматизованої системи управління навчальним процесом, визначення видів та обсягів вхідних та вихідних даних;
- систематизувати і класифікувати існуючий перелік планувальної і обліково-контролюючої документації на основі форм, рекомендованих Міністерством освіти і науки України та з урахуванням вимог і стандартів ECTS;

- розробити та узгодити форми документів для обліку та звітності внутрішньокафедральних та внутрішньофакультетських видів діяльності (навчально-методична, наукова, організаційно-виховна, профорієнтаційна та ін.);
- здійснити практичну розробку програмного продукту для автоматизації управління навчальним процесом сумісного з існуючим апаратним і програмним забезпеченням з можливістю використання програми, як на окремому взятому комп'ютері, так і в локальній загальноуніверситетській мережі. Передбачити заходи захисту інформації від несанкціонованого доступу та від непередбачених втрат інформації: пошкоджень програми та інформаційної бази вірусами й іншими негативними факторами;
- провести дослідну перевірку програмної розробки в умовах наповнення бази реальними даними. Здійснити аналіз вихідних даних на основі отриманих облікових, обліково-контролюючих і звітних форм документів.
- розробити пропозиції і рекомендації щодо впровадження системи автоматизації навчального процесу на всіх факультетах КНТЕУ.

Організаційна схему управління повинна постійно змінюватись, зокрема при динамічному зовнішньому середовищі деканату і структурі управління. Відповідно до змін у структурі управління, які зумовлюють зміни у чисельності персоналу, важливо, щоб ці зміни суттєво не позначались на якості навчання. Персонал у деканаті завжди повинен взаємодоповнюватись, бути взаємозамінним у певному структурному підрозділі.

2.3. Система керування базами даних postgresql

PostgreSQL - це одна з найпопулярніших і найпоширеніших систем керування базами даних (СКБД) у світі. Її використовують у проектах різних розмірів та різної складності. Дану СКБД використовують багато компаній, починаючи з Apple, Google, NASA та закінчуючи багатьма державними структурами багатьох країн.

PostgreSQL швидко отримала свою популярність так як відрізнялася хорошою швидкістю роботи, надійністю, зручністю і робота з нею, як правило, не викликає великих труднощів у початківців. Ще одні великим плюсом є те, що поширюється вона безкоштовною та має гарну документацію.

Раніше вся інформація, яку потрібно було зберігати тривалий час, заносили в файли і при потребі звідти діставали. Працювати таким способом було досить складно тому для роботи з файлами потрібно написання досить великого обсягу коду, і ймовірність допустити в ньому помилку збільшувалася. Так само викликало труднощі сортування даних. Всі ці проблеми вирішує використання СКБД.

В наш час майже всі реляційні СКБД використовують SQL в якості мови запитів.

SQL - формальна непроцедурна мова програмування, застосовується для створення, модифікації та управління даними в довільній реляційній базі даних, керована відповідною системою керування базами даних (СКБД) та грунтується на обчисленні кортежів.

Зараз SQL поділяється на 2 великі підгрупи:

- DDL (Data Definition language) – мова опису даних. До цієї частини SQL належать всі команди, що дозволяють створювати БД, задавати структуру таблиць і тд
- DML (Data Modifying language) – мова редагування даних. Сюди належать команди, за допомогою яких можна робити вибірку, додавати, редагувати, видаляти певні дані з БД.

Для виконання будь-яких команд потрібно запустити консоль або графічний інтерфейс PostgreSQL. При вході ви повинні вказати ім'я користувача, його пароль, назву БД, до якої підключитися. Якщо БД знаходиться не локально, а на сервері , то потрібно ввести адресу, де знаходиться сервер. Це все можна зробити однією командою

psql -h <host> -p <port> -U <username> -W <password> <database>

psql – виконуючий файл, що встановлює фізичне з’єднання з БД.

При імпорті даних з Erwin в якості SQL-скрипта (з назвою script.sql), можна скористатися командою, де СКБД виконає даний скрипт.

psql -d <database> -a -f script.sql

Як було зазначено вище, основні команди, що допомагають редагувати зміст БД – DML. При роботі з БД в контексті мов програмування зустрічається поняття CRUD-операції (Create, Update, Read, Delete) – основні операції зі змістом конкретної БД, такі як створення нових записів, редагування та видалення існуючих та зчитування інформації.

INSERT INTO table_name [AS alias] [(column_name [, ...])] VALUES ({ expression | DEFAULT } [, ...])

UPDATE [ONLY] table [*] [[AS] alias]
SET { column = { expression | DEFAULT } |
(column [, ...]) = ({ expression | DEFAULT } [, ...]) } [, ...]
[FROM from_list]
[WHERE condition]

DELETE FROM [ONLY] table_name [*] [[AS] alias] [WHERE condition]

При запису в таблицю ми вказуємо які дані потрібно вставити атрибутам, при оновленні ми вказуємо які дані потрібно замінити та умову в яких записах це робити. При видаленні ми, лише, вказуємо умову за якою потрібно видалити записи в таблиці.

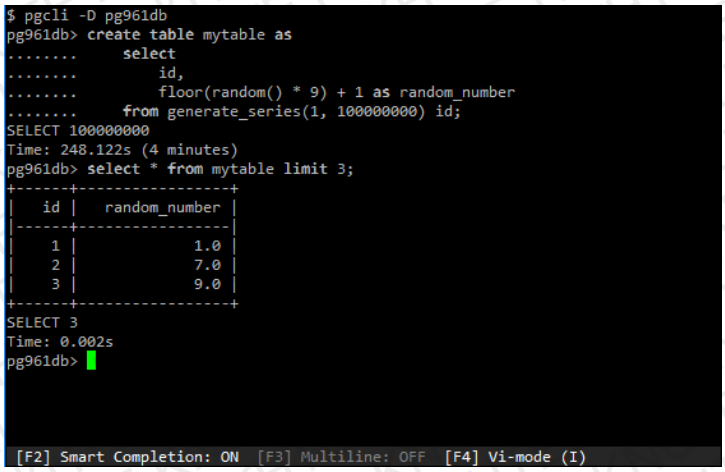

Рис. 2.1. Приклад консольного клієнту postgresql

Приклад консольного клієнту postgresql можна побачити на рисунку 2.1.

2.4. Опис фреймворку для створення веб-додатків spring

Spring являє собою легкий, багатофункціональний java фреймворк для побудови складних додатків з багаторівневою архітектурою, який найчастіше використовують при розробці веб-додатків. Фундаментом даного фреймворка є його реалізація IoC контейнера, про який мова піде в наступних підпунктах.

Типовий додаток на java складається з безлічі об'єктів, що взаємодіють між собою, і тим самим формуючи залежності між собою. Платформа java надає величезний функціонал при розробці ПЗ, проте в ній відсутні способи організації компонентів додатка в єдине ціле. І хоча можна використовувати дизайн-патерни, такі як Builder, Factory, і Service Locator для композиції об'єктів, які формують додаток, ці патерни є не більше ніж описаної практикою в програмуванні, і їх ще необхідно реалізувати. IoC контейнер Spring вирішує дану проблему, надає формалізовані способи складання різних компонентів в працюючий додаток, готовий до використання.

Розглянутий фреймворк дуже великий і має модульну структуру, найбільш часто використовувані елементи якої представлені в таблиці 2.1.

Таблиця 2.1.

Основні елементи фреймворка spring

Група	Модуль	Опис
Основний модуль	spring-core, spring-beans	Фундамент фреймворка, що надає функціонал інверсії контролю та впровадження залежностей.
	spring-context	Надбудова над попередніми модулями, що формалізує доступ до компонентів додатку
	spring-context-support	Підтримка інтеграції сторонніх бібліотек до Spring.

	spring-expression	Реалізація мови виразів spring expressiong language (SPel)
	spring-tx	Підтримка декларативних транзакцій.
Доступ до даних	spring-orm	Модуль для інтеграції для об’єктно-реляційного відображення
Веб	spring-webmvc	Реалізація парадигми MVC для веб-додатків, а також REST веб-сервісів.

Spring реалізує принцип інверсії управління (IoC), також відомий як впровадження залежностей (Dependency Injection, DI). Згідно з цим принципом, компоненти програми визначають свої залежності, тобто інші об'єкти, з якими вони працюють тільки через аргументи конструктора, аргументи до factory-методам, або в своїх властивостях, які встановлюються після створення об'єкта або повертаються з factory метода. Контейнер потім впроваджує ці залежності в момент створення компонента. Метадані, необхідні контейнеру для створення екземплярів компонентів можуть поставлятися у вигляді конфігугаціонних xml-файлів, у вигляді java-анотацій, або у вигляді java-класу конфігурації.

Spring WebMVC реалізує парадигму MVC для веб додатків. Для роботи з даним модулем необхідно оголосити сервлет DispatcherServlet, який грає роль передового контролера (front controller), отримуючи всі запити до сервлет-контейнеру. Після отримання запиту, як правило, відбувається наступне:

1. Сервлет передає отриманий запит компоненту HandlerMapping, який в залежності від запитуваної шляху, HTTP метода, і ряду інших параметрів віддає назад екземпляр компонента Controller, який відповідає цьому запиту.
2. Сервлет передає запит отриманому раніше контролера, і отримує назад модель і ім'я виду, який буде використовуватися для відображення даних моделі. Ці дані вміщені в об'єкті ModelAndView

3. Сервлет передає ім'я виду компоненту ViewResolver, який по імені виду надає екземпляр компонента View, який буде Відповідальний за відображення даних моделі.
4. Сервлет передає модель виду, отриманого на попередньому кроці, отримуючи назад сформований відповідь сервера.

Примірники компонентів HandlerMapping і Controller створюються і конфігуруються контейнером відповідно до анотацій класів, проанотованих @Controller. Мова про них піде далі. Примірники компонентів ViewResolver створюються і конфігурується контейнером відповідно до їх оголошеннями в конфігурації програми. Приклад такого оголошення представлений далі. Взаємодія

згаданих компонентів проілюстровано на рисунку 2.1.

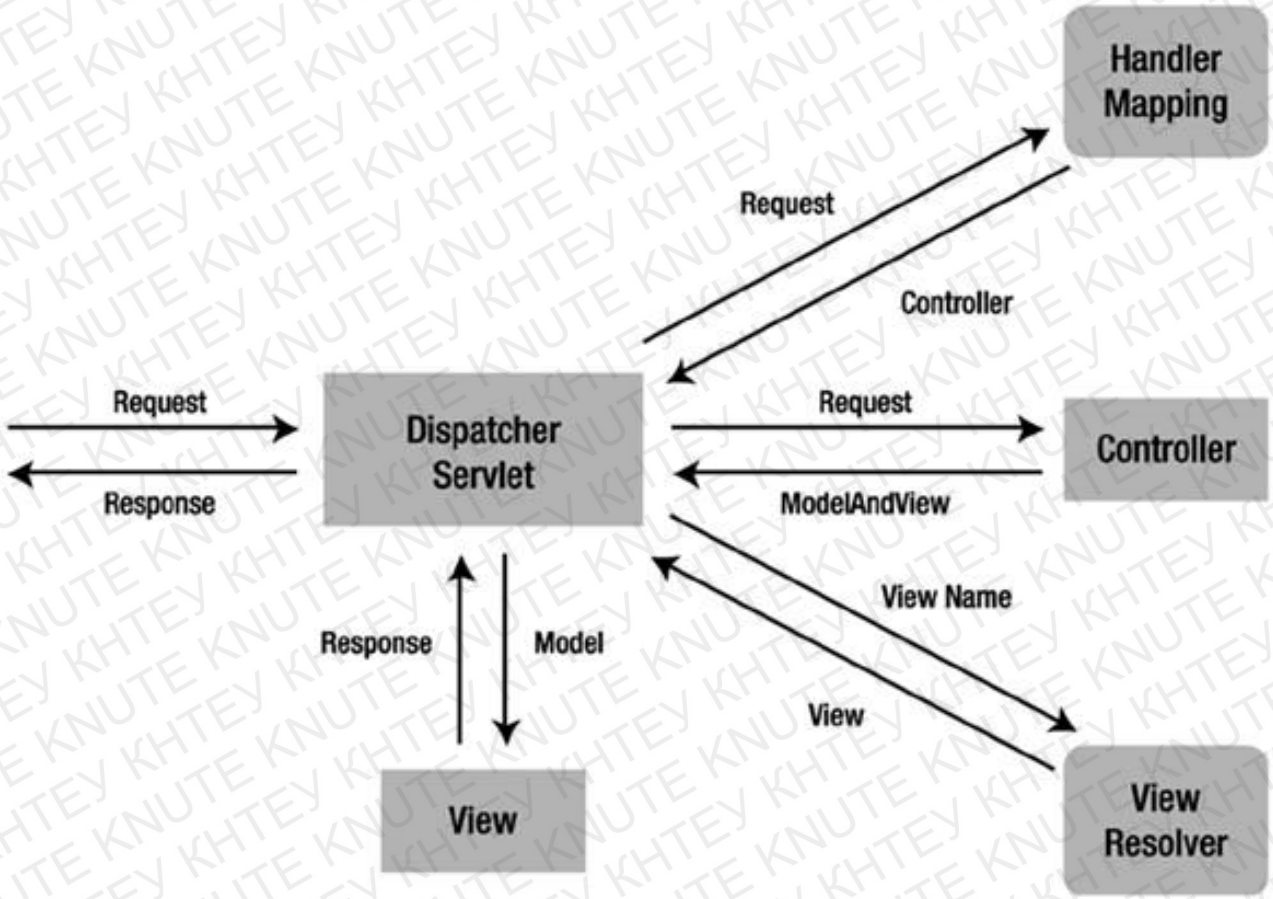

Рис. 2.1. Взаємодія компонентів Spring MVC

Ключовою абстракцією даного модуля є контролер. Саме контролери при використанні SpringMVC грають роль точки входу для усієї логіки додатка. Для реєстрації класу-контролера, його необхідно проанотувати як @Controller.

За допомогою анотацій java можна описати безліч аспектів поведінки контролера. Найбільш часто використовувані анотації, використовувані при описі контролерів представлені в таблиці 2.2.

Таблиця 2.2.

Основні анотації spring mvc

Приклад анотації	Опис
1	2
@Controller	Перед ім'ям класу сигналізує про те, що цей клас - контролер, і повинен бути відповідним чином оброблений контейнером.
1	2
@RestController	Перед ім'ям класу сигналізує про те, що цей клас - контролер, і крім цього – що повернення методів даного контролера повинен інтерпретуватися як тіла відповіді, а не ім'я моделі.
@RequestMapping(“/foo”)	Перед методом контролера сигналізує про тому, що це метод-обробник, який необхідно зареєструвати по шляху /foo
@RequestParam(“foo”)	Перед аргументом методу контролера сигналізує про те, що в цим аргументом треба передати значення параметра запиту з ім'ям foo
@PathVariable(“foo”)	Перед аргументом методу сигналізує про те, що в цей аргументом треба передати

	значення змінної шляху запиту з ім'ям foo
@ModelAttibute(“foo”)	Перед аргументом методу сигналізує про те, що в цей аргументом треба десереалізувати атрибут моделі з ім'ям foo
@RequestBody	Перед аргументом методу сигналізує про те, що в цей аргументом треба десереалізувати тіло запиту
@ResponseBody	Перед методом сигналізує про те, що повернення необхідно трактувати як тіло відповіді, а не назву моделі.
@ResponseStatus	Перед методом вказує, який HTTP статус використовувати, в разі успішної обробки запиту.

2.5. Порівняння Spring та Java EE

Java EE - набір специфікацій і відповідної документації для мови Java, яка описує архітектуру серверної платформи для задач середніх і великих підприємств.

Spring Dependency Injection та Java EE Context Dependency Injection - Навіть з назви зрозуміло, що основа в обох стеках, майже, однакові зв характеристиками. Фактично в Java EE спочатку і використовувався спрінговий DI. Але після того як його включили прямо в основний проект Spring, використовувати його окремо стало неможливо. Довелося робити свій DI.

Spring Beans та Enterprise Java Beans - Spring beans - це просто звичайні джава біни, які використовуються для впровадження та нічого більше. EJB - досить потужний механізм, в який вбудована підтримка розподіленого виконання, включаючи розподілений збирач сміття, аутентифікацію, підтримку транзакцій та багато чого іншого.

Spring Service Locator та Java Naming and Directory Interface - Служба, яка допомагає знайти в тому ж JVM сервіс, та служба, яка може працювати на

розподіленій системі і знаходити відповідність між чим завгодно, включаючи прив'язку до LDAP компанії відповідно.

Асинхронність в обох фреймворках - Асинхронність також реалізована в обох стеках на різному рівні. У стеці Java EE це, зокрема, розподілена служба, що підтримує передачу повідомлень не тільки точка-точка, а й багатьом одержувачам. Загалом, ситуація аналогічна до попередніх пунктів. Просте і швидке проти розподіленої технології.

Spring MVC та Java Server Faces - Проблема цього порівняння в тому, що для Spring MVC в якості шаблонізатору, зазвичай, використовується Thymeleaf або в більш сучасному варіанті - будь-який фреймворк для інтерфейсу користувача React, або Angular. У той час, як JSF - повноцінне GUI рішення з підтримкою технології AJAX з коробки та SPA додатків за допомогою додаткових бібліотек.

Spring Data та Java Persistence API - У стеку Java EE специфікація JPA є досить об'ємною та важкою у налаштуванні та підтримці, Spring Data - налаштування над JPA, що дозволяє програмісту, майже, без коду та конфігурацій робити досить швидкий доступ до рівню даних.

Хоча технологія Java EE і здається досить потужною, але в наш час вона стає все старішою, що має свої наслідки, такі як: все менше людей користується даною технологією, менше інформації по новим версіям можна знайти, рідше випускаються оновлення, повільніше йде процес полагодження багів. Spring дає можливість програмісту не так багато часу витрачати на конфігурування системи (фреймворку), та автоматизує за програміста багато рутинної роботи.

2.6. Використання зв’язки фреймворку spring та СКБД postgresql

Для зв’язування СКБД postgresql та фреймворку spring використовуються різні конектори та зв'язки бібліотек. Але, оскільки, ручне установлення може займати багато часу при конфігуруванні, то розробники spring створили свій невеликий підпроект – spring boot.

Spring boot - це набір зручних дескрипторів залежностей між модулями, які значно спрощують конфігурацію системи збору проекту Maven.

Дані дескриптори залежностей називаються стартерами. Основний стартер spring-boot-starter має деякі загальні конфігурації для фреймворку spring.

- spring-boot-starter-web - це стартер для створення веб-сторінок, що включає в себе багато конфігурацій для проекту включаючи Spring MVC. Він використовує Tomcat як вбудований контейнер сервлетів за замовчуванням.
- spring-boot-starter-thymeleaf - це стартер для створення веб-додатків Spring MVC з використанням шаблонізатору Thymeleaf.
- spring-boot-starter-data-jpa - це стартер для використання Spring Data JPA з Hibernate.

За допомогою підключення драйверу системи керування базами даних та стартеру spring-boot-starter-data-jpa, фреймворк сам розуміє з опису підключеного драйверу, що потрібно використовувати, саме, СКБД postgresql

2.7. Висновки та пропозиції до розділу 2

В даному розділі було описано основні компоненти бібліотек spring та їх взаємодію, велику увагу приділено бібліотеці spring mvc (Model-view-controller), що надлаштовується на бібліотеку spring core, та відповідає за передачу даних з мови програмування Java на зовнішній вигляд, що буде відображатися користувачу у браузері у вигляді html сторінок, або у текстовому вигляді у різних форматах, як, наприклад, json, hateoas, xml та інші. Було порівняно більш нову технологію Spring та застарілу технологію Java EE. Також будо описано систему керування базами даних postgresql, основні принципи роботи з нею, приведено приклади базових команд в консольному клієнті СКБД postgresql. Та було описано тенологію контейнеризації Docker, її переваги по зрівнянню з віртуалізаціє

РОЗДІЛ 3. РОЗРОБКА МОДУЛЯ МОНІТОРИНГА АКТИВНОСТІ КОРИСТУВАЧІВ

3.1. Визначення основних термінів spring framework

Spring являє собою легкий, багатофункціональний java фреймворк для побудови складних додатків з багаторівневою архітектурою, який найчастіше використовують при розробці веб-додатків. Фундаментом даного фреймворка є його реалізація IoC контейнера, про який мова піде в наступних підпунктах.

Типовий додаток на java складається з безлічі об'єктів, що взаємодіють між собою, і тим самим формуючи залежності між собою. Платформа java надає величезний функціонал при розробці ПЗ, проте в ній відсутні способи організації компонентів додатка в єдине ціле. І хоча можна використовувати дизайн-патерни, такі як Builder, Factory, і Service Locator для композиції об'єктів, які формують додаток, ці патерни є не більше ніж описаної практикою в програмуванні, і їх ще необхідно реалізувати. IoC контейнер Spring вирішує дану проблему, надає формалізовані способи складання різних компонентів в працюючий додаток, готовий до використання.

Spring реалізує принцип інверсії управління (IoC), також відомий як впровадження залежностей (Dependency Injection, DI). Згідно з цим принципом, компоненти програми визначають свої залежності, тобто інші об'єкти, з якими вони працюють тільки через аргументи конструктора, аргументи до factory-методам, або в своїх властивостях, які встановлюються після створення об'єкта або повертаються з factoryметода. Контейнер потім впроваджує ці залежності в момент створення компонента. Метадані, необхідні контейнеру для створення екземплярів компонентів можуть поставлятися у вигляді конфігугаціонних xml-файлів, у вигляді java-анотацій, або у вигляді java-класу конфігурації. Нижче представлені три підходи до конфігурації контейнера.

xml-конфігурація:

```
<? Xml version = "1.0" encoding = "UTF-8"?>
<Beans xmlns = "http://www.springframework.org/schema/beans"
 xmlns: xsi = "http://www.w3.org/2001/XMLSchema-instance"
xmlns: context = "http://www.springframework.org/schema/context"
xsi: schemaLocation = "http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/springbeans.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/springcontext.xsd ">
 <! - Опис компонента sampleRepository! ->
 <Bean class = "example.SampleRepository">
 <! - Блок залежностей компонента! ->
 </ Bean>
 <! - Опис компонента sampleService! ->
 <Bean class = "example.SampleService">
 <! - Блок залежностей компонента! ->
 </ Bean>
</ Beans>
```

annotation-конфігурація:

```
// Клас конфігурації, що включає сканування пакету "example-
// package ". Для класів в цьому пакеті, проаннотірованних
// @ Component будуть згенеровані оголошення компонентів
@Configuration
@ComponentScan ("example-package")
public class BeanConfiguration {
 // Додаткові оголошення компонентів
}
```

java-конфігурація:

```
// Клас конфігурації, що містить factory-методи для створення
```

```
// примірників компонентів
@Configuration
public class BeanConfiguration {
 // factory-метод для компонента sampleRepository
 @Bean
 public SampleRepository sampleRepository () {...}
 // factory-метод для компонента sampleService
 @Bean
 public SampleService sampleService () {...}
}
```

Всі три конфігурації приведуть до одного результату – оголошенню компонентів програми з ідентифікаторами sampleRepository і sampleService. Однак слід зауважити, що java-конфігурація найбільш гнучка, тому що вона дозволяє описати будь-який процес створення компонента. Слід також зазначити, що annotation-конфігурація буде ефективною, тільки якщо класи компонентів будуть проаннотіровани як @Component. Після надання необхідних метаданих контейнер можна використовувати явно, за допомогою наданих реалізацій інтерфейсів ApplicationContext і BeanFactory.

```
// створюємо контекст
ApplicationContext context =
 new ClassPathXmlApplicationContext (new String []
{ "Repositories.xml", "services.xml"});
// отримуємо екземпляр компонента
SampleRepository repository =
context.getBean ("sampleRepository", SampleRepository.class);
// використовуємо конфігурований примірник
Data someData = service.getSomeData ();
```

3.2. Можливості spring framework

Spring - це інтеграційна основа для легкого розвитку корпоративних додатків . Spring Framework містить такі особливості:

а. Легкий. Spring описується як легкий каркас, що стосується розміру та прозорості. Легкий каркас допомагає зменшити складність коду програми. Це також допомагає уникнути зайвих складностей у власному функціонуванні. Легкий каркас не матиме високого часу запуску і працюватиме в будь-яких умовах. Легкий каркас також не передбачає величезних бінарних залежностей;

б. Не нав'язливий. Це означає, що логічний код домену не залежить від самого фреймворку. Spring Framework розроблений таким, що не є нав'язливим. Об'єкт у програмі з підтримкою Spring зазвичай не має залежності від будь-якого заздалегідь визначеного інтерфейсу чи класу, заданого Spring API. Таким чином, Spring може конфігурувати об’єкти програми, які не імпортують API;

в. Інверсія управління (IoC). Контейнер Spring - це легкий контейнер, який містить Spring боби і керує їх життєвим циклом. Основний контейнер Spring Framework забезпечує реалізацію для ін'єкцій, що підтримують IoC. IoC - це архітектурна схема, яка описує введення залежностей, яку необхідно робити зовнішньою сутністю, а не створювати залежності самим компонентом. Об'єкти пасивно задаються залежностями, а не створюють для себе залежні об'єкти;

г. Аспект-орієнтоване програмування (AOP). Це стосується парадигми програмування, яка ізолює допоміжні функції від бізнес-логіки основної програми. Це дозволяє розробнику будувати основну функціональність системи, не усвідомлюючи додаткових вимог. AOP використовується у Spring Framework для надання декларативних аспектів, таких як транзакції та безпека. Тут об’єкти додатків виконують ділову логіку і не відповідають за інші проблеми системи, такі як реєстрація, безпека, аудит, блокування та

обробка подій. AOP - це метод застосування проміжних програм, таких як служба безпеки та служба управління транзакціями у додатку Spring;

д. Обробка винятків JDBC. Шар абстракції JDBC Spring Framework забезпечує ієрархію винятків. Це скорочує стратегію обробки помилок в JDBC. Це один із напрямків, коли Spring дійсно допомагає зменшити кількість коду, який потрібно записати в обробці винятків;

е. Spring MVC Framework. Це допомагає у створенні надійних та бездоганних веб-додатків. Він використовує IoC, що забезпечує розділення логіки контролера. Spring MVC Framework, що є частиною Spring Framework, що ліцензується під ліцензією Apache, є фреймворком веб-додатків з відкритим кодом. Spring MVC Framework пропонує утилітні класи для обробки деяких найпоширеніших завдань у розробці веб-додатків;

ж. Spring безпека. Це забезпечує декларативний механізм безпеки для програм на базі Spring, що є критичним аспектом багатьох програм;

з. Інші особливості Spring:

1. Веб-сервіси Spring: Це надає першу за контрактом модель веб-сервісів, згідно з якою реалізація послуг пишеться для задоволення договору про надання послуг;
2. Spring Batch : Це корисно, коли необхідно виконувати об'ємні операції над даними;
3. Spring Social: Соціальні мережі сьогодні є тенденцією до зростання в Інтернеті, і все більше додатків, таких як Facebook та Twitter, оснащуються інтеграцією в сайти соціальних мереж;
4. Spring Mobile: Мобільні додатки - ще одна важлива область розробки програмного забезпечення. Spring Mobile підтримує розробку мобільних веб-додатків.

3.2. Використання технології spring для розробки системи

Для зв'язки роботи з базою даних потрібно використовувати модуль ORM.

Цей модуль відповідає за відображення таблиці в базі даних в об’єкти мови програмування java.

Перший рівень, який потрібно описати – рівень сутностей. На рисунку 3.1. показано як в програмному коді, за допомогою spring orm робиться робиться це відображення

```java
@Entity
@Table(name = "student", schema = "public", catalog = "decanat")
public class Student {
    private int studentId;
    @NotNull
    @Size(min = 2, max = 30)
    private String firstName;

    @NotNull
    @Size(min = 2, max = 30)
    private String lastName;

    @NotNull
    @Size(min = 2, max = 30)
    private String secondName;

    @Past
    @NotNull
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date birth;

    @NotNull(message = "Не відповідає шаблону: +380633030303")
    @Pattern(regexp = "^\\+(?:[0-9] ?){6,14}[0-9]$", message = "Не відповідає шаблону: +380633030303")
    private String phone;
```

Рис. 3.1. Об’єктно-реляційне відображення об’єктів

Основні аспекти об'єктно-реляційного відображення – це анотації

- Анотація @Entity дає фреймворку зрозуміти, шо цей клас буде використовуватися для обробки об’єктно-реляційним модулем.
- Анотація @Table відповідає за зв’язку об’єкту мови програмування, до назви таблиці в базі даних. Параметри, що мають встановлюватися – назва таблиці, схема в базі даних та каталог.
- Анотація @NotNull зобов’язує фреймворк валідувати значення параметра, щоб воно не було пустим
- Анотація @Size використовується для чисельних значень, щоб валідувати діапазрон значень, що може приймати даний параметер
- Анотація @DateTimeFormat вказує який формат дати та часу потрібно використовувати.

- Анотація @Pattern показує яким патерном потрібно валідувати строку, та показувати помилку, якщо строка не підходить під патерн.

Настуний рівень архітектури додатку – репозиторій. Репозиторій – частина системи, що відповідає за звернення до бази данизх, отримання відповідей та обробку результатів, код можна побачити на рисунку 3.2.

```
public interface StudentRepository extends JpaRepository<Student, Integer> {
    List<Student> findAllByGroupIs(DecGroup groupId);

    @Query("select s from Student s " +
            "inner join s.group g " +
            "where g.decGroupId = (select g.decGroupId from Student s inner join s.group g " +
            "inner join s.user u " +
            "where u.login = :username) " + "" +
            "order by s.lastName")
    List<Student> getStudentsByGroupByStudentUsername(@Param("username") String username);
}
```

Рис. 3.2. Репозиторій доступу до бази даних

В репозиторії потрібно вказувати методи, щоб дістати дані з бази даних, наприклад, дістати всіх, дістати по номеру, та інші. За допомогою анотації @Query можна вказувати запити на мові програмування SQL.

```
@Service
public class StudentServiceImpl implements StudentService {
  private static final Logger LOG = LoggerFactory.getLogger(StudentServiceImpl.class);

  private final StudentRepository studentRepository;
  private final RoleRepository roleRepository;

  private final PasswordEncoder encoder;

  @Autowired
  public StudentServiceImpl(StudentRepository studentRepository,
                            RoleRepository roleRepository,
                            PasswordEncoder encoder) {
    this.studentRepository = studentRepository;
    this.roleRepository = roleRepository;
    this.encoder = encoder;
  }

  @Override
  public List<Student> getAll() {
    return studentRepository.findAll();
  }

  @Override
  @Transactional
  public Student save(Student entity) {
```

Рис. 3.3. Рівень сервісів

Ще один рівень архітектури – рівень сервісів. Сервіс може об’єднувати у собі декілька репозиторіїв та агрегувати дані з них, як це показано ма рисунку 3.3.

Як ми можемо побачити, що сервіс, що відповідає за обробку даних студентів має декілька репозиторіїв та може обробляти дані з обох. Анотація @Transactional показує, що в даному методі буде викликаний метод з 2-х репозиторіїв (таблиць), та потрібно обернути ці запити в транзакцію на стороні системи керування базами даних.

Останній рівень – рівень контроллерів. Від відповідає за обробку результатів з сервісів та пересилання даних для відображення інтерфейсу користувача.

На рисунку 3.4. можна побачити код, що і є контроллером.

```
@Controller
@RequestMapping("/student")
public class StudentController {
  private final StudentService studentService;
  private final FacultyService facultyService;

  @Autowired
  public StudentController(StudentService studentService, FacultyService facultyService) {
    this.studentService = studentService;
    this.facultyService = facultyService;
  }

  @GetMapping
  public String findAll(Model model) {

    model.addAttribute( s: "students", studentService.getAll());

    return "student.html";
  }

  @GetMapping("/add")
  public String addStudent(Model model) {
    model.addAttribute( s: "student", new Student());

    model.addAttribute( s: "faculties", facultyService.getAll());
    return "addStudent.html";
  }

  @PostMapping("/add")
  public String addStudent(/*@Valid*/ Student student, BindingResult bindingResult, Model model) {

    if (bindingResult.hasErrors()) {
      model.addAttribute( s: "faculties", facultyService.getAll());
      return "addStudent.html";
    }

    studentService.save(student);

    return "redirect:/student";
  }
}
```

Рис. 3.4. Контроллер spring

Основні елементи контроллеру –це:

- Анотація @Controller показує фреймворку, що це є контроллер.
- Анотація @ModelAttribute показує, який саме об’єкт потрібно передати на користувацький інтерфейс.

- Анотація @GetMapping показує що метод – get.
- Анотація @PostMapping показує, що метод – post.

3.3 Робота з додатком системи керування базами даних PgAdmin

Для початку роботи з PgAdmin необхідно на локальній машині відкрити веб-браузер і перейти за IP-адресою сервера, вказаного під час процесу встановлення.

Потрапивши туди, необхідно ввести дані для входу (рис. 3.5). Після введення даних на екрані буде відображена головна сторінка PgAdmin.

Рис. 3.5. Вікно входу в PgAdmin

Тепер, коли ви підтвердили, що можете отримати доступ до інтерфейсу pgAdmin, все, що вам потрібно зробити, це підключити pgAdmin до вашої бази даних PostgreSQL. Для підключення до бази даних, необхідно натиснути на поле Сервер правою клавішею миші, далі З'єднання. На екрані буде відображено вікно підключення бази даних (рис. 3.6).

У полі Ім'я / адреса хоста необхідно ввести localhost. Порт повинен бути встановлений 5432 за замовчуванням, який буде працювати для цієї установки, так як це порт за замовчуванням використовується PostgreSQL. У полі бази даних Технічне обслуговування необхідно ввести ім'я бази даних, до якої необхідно підключитися. Ця база даних вже повинна бути створена на сервері. Потім

необхідно ввести ім'я користувача та пароль PostgreSQL, які було налаштовані раніше, у полях Ім'я користувача та Пароль відповідно.

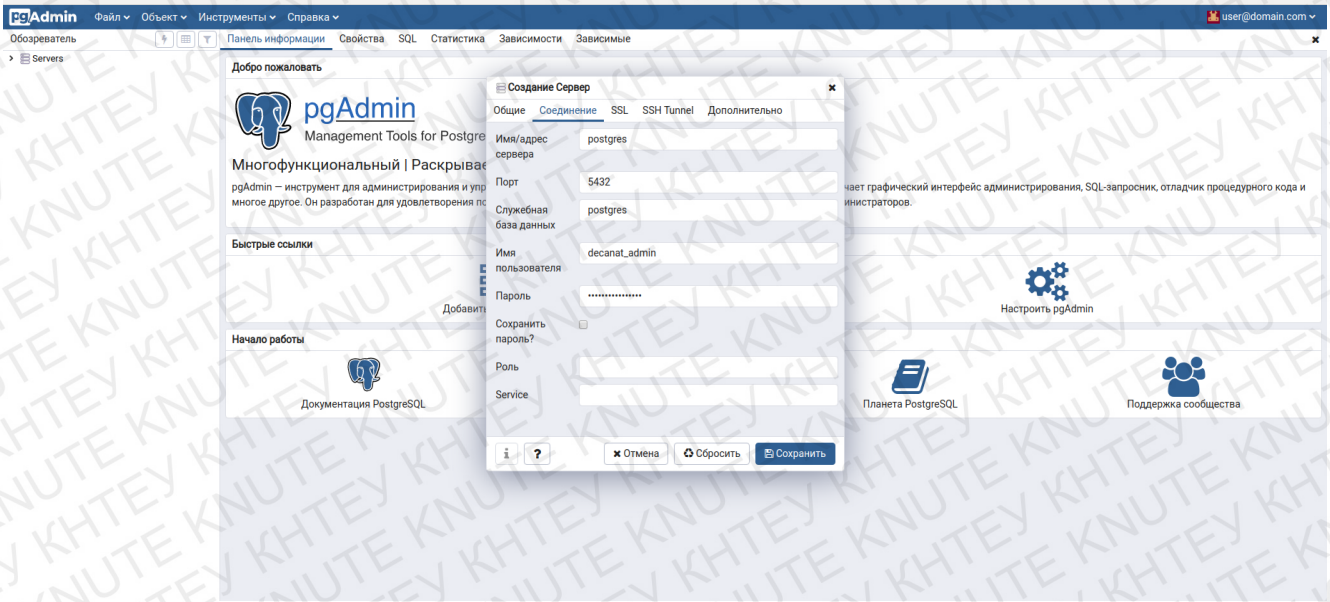

Рис. 3.6. Вікно підключення нової бази даних

Після введеня інформації необхідно натиснути кнопку Зберегти для підключення PgAdmin до бази даних. Коли PgAdmin підключено до бази даних на екрані відображається структура бази даних у вигляді дерева та основні показники роботи з базою даних у вигляді графіків в режимі реального часу (Рис 3.7).

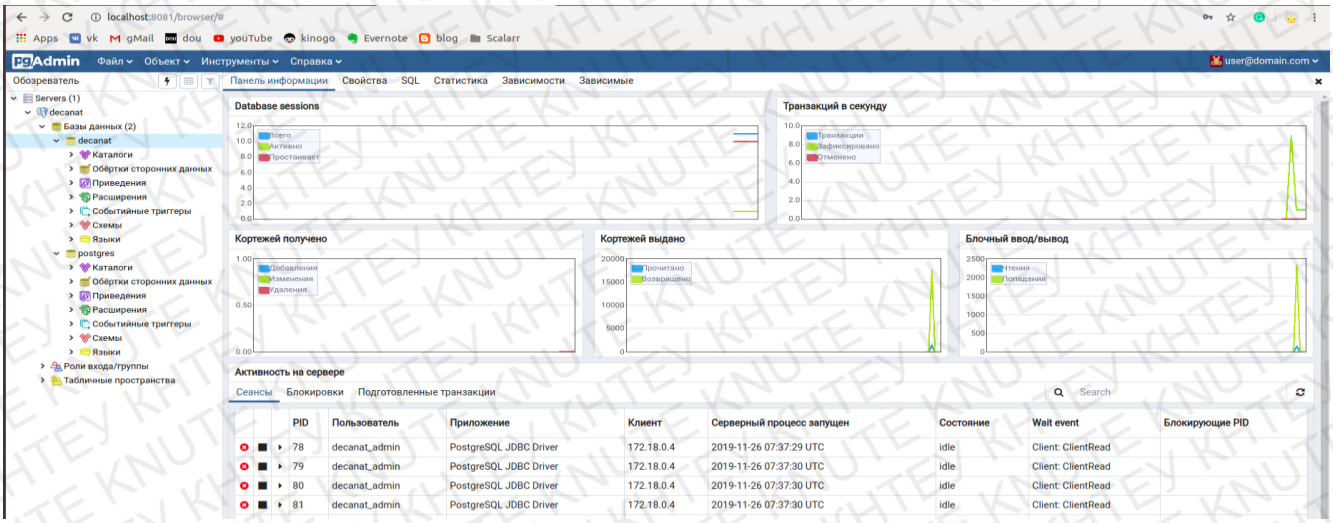

Рис. 3.7. Головний екран роботи з базою даних

Дерево і правій частині екрану дозволяє переглядати таблиці та їх властивості (Рис. 3.8).

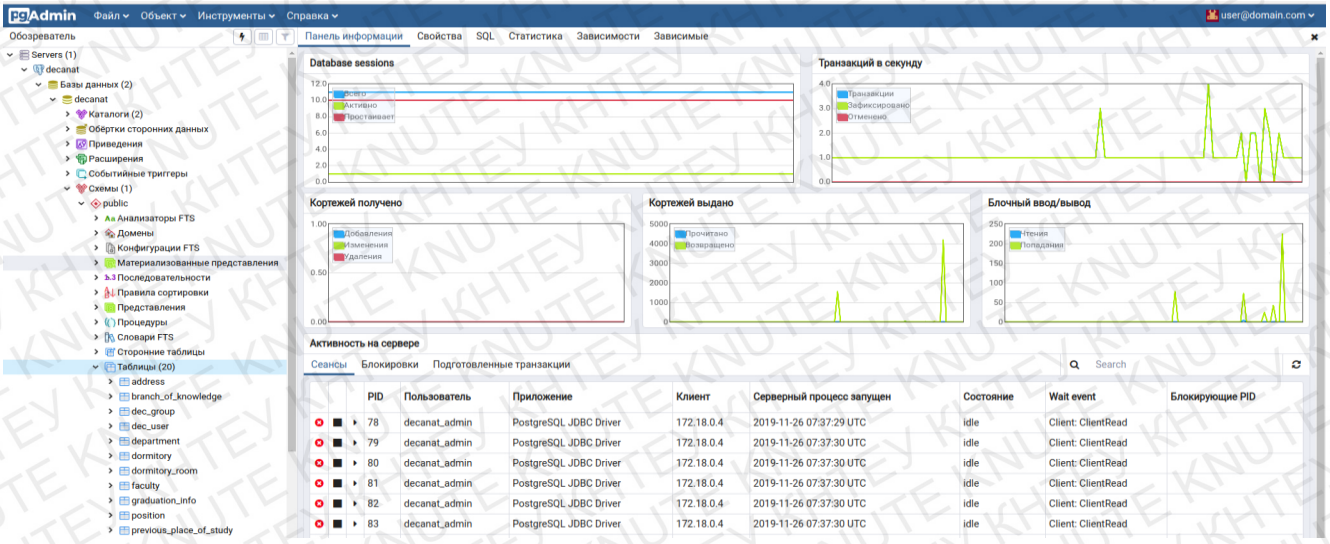

Рис. 3.8. Структура таблиць в базі даних

Для перегляду та редагування властивостей кожної із таблиць необхідно розвернути її ієрархію (рис 3.9).

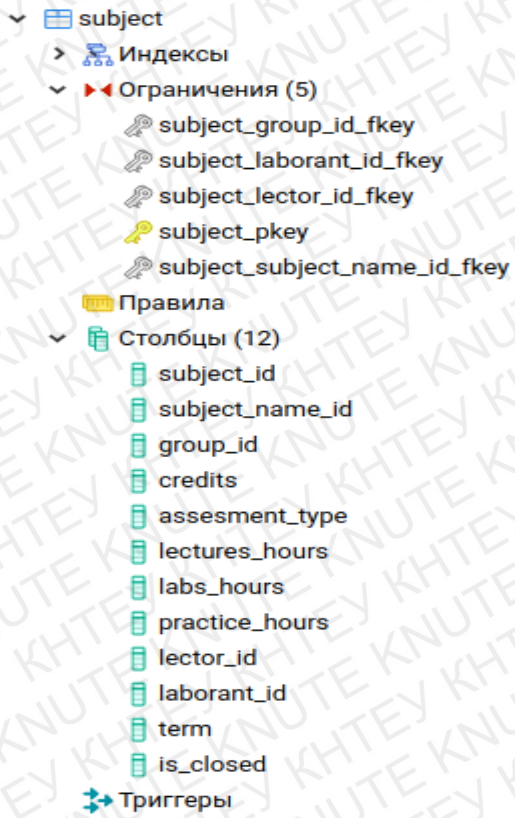

Рисунок 3.9. Властивості таблиці бази даних

Також PgAdmin дозволяє створювати запити до бази даних за допомогою вбудованої консолі. Приклад такого запиту зображено на рисунку 3.10.

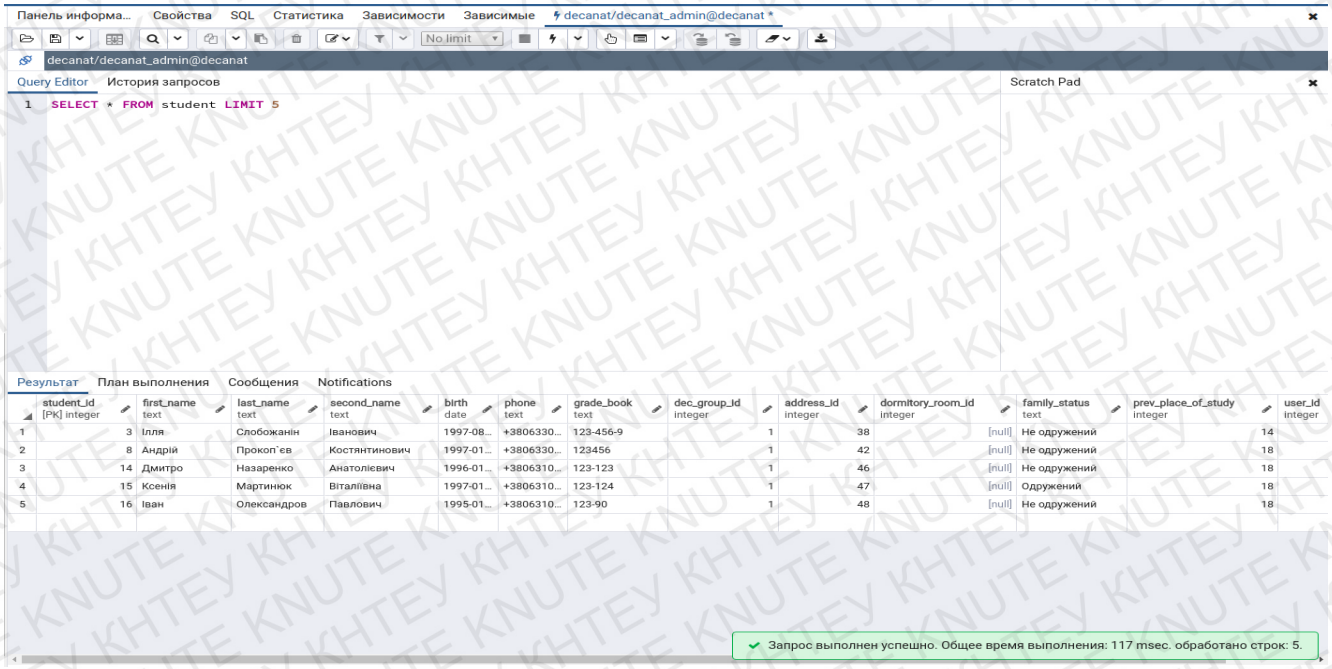

Рис. 3.10. Приклад запиту до бази даних

3.4. Висновки та пропозиції до розділу 3

В даному розділі було продемонстровано зв'язку системи в цілому, розписано про взаємодію компонентів та технологій, таких, як spring, postgresql, pgadmin, та інших. Велику увагу було приділено опису можливостей spring та показано його використання в реальних проектах. Також було показано графічний інтерфейс до системи керування базами даних postgresql, який називається PgAdmin, та описані та проілюстровані основні його його можливості.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

В даній роботі описано концепцію інверсії контролю (IoC, Inversion of control), його переваги у порівнянні з ручним управлінням життєвого циклу об'єктів, описано spring framework, та яке місце в ньому займає контейнер інверсії контролю, а саме його реалізацію у вигляді впровадження залежностей (dependency injection) та як він працює. Також було описано основні компоненти бібліотек spring та їх взаємодію, велику увагу приділено бібліотеці spring mvc (Model-view-controller), що надлаштовується на бібліотеку spring core, та відповідає за передачу даних з мови програмування Java на зовнішній вигляд, що буде відображатися користувачу у браузері у вигляді html сторінок, або у текстовому вигляді у різних форматах, як, наприклад, json, hateoas, xml та інші. Описано інструментальний засіб для розробки баз даних PostgreSQL. Наведено основні переваги СКБД PostgreSQL. Визначено вбудовані типи даних в PostgreSQL.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лазарев А.А. Решение NP-трудной задачи теории расписаний минимизации суммарного запаздывания // Журнал Вычислительной математики и математической физики – 2007. том 47, N.6. – С. 1087– 1098
2. Лазарев А.А., Гафаров Е.Р. Теория расписаний. Минимизация суммарного запаздывания для одного прибора. // Научное издание, М.: Вычислительный центр им. А.А. Дороницына РАН, 2006. - 134 с.
3. Лазарев А.А., Гафаров Е.Р. Теория расписаний. Исследование задач с отношениями предшествования и ресурсными ограничениями. // Научное издание, М.: Вычислительный центр им. А.А. Дороницына РАН, 2007. – 80 с.
4. Лазарев А.А., Гафаров Е.Р. Теория расписаний задачи и алгоритмы. М.: Наука, 2009.
5. Танаев В.С., Шкурба В.В. Введение в теорию расписаний. М.: Наука, 1975.
6. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы. М.: Наука. Гл. ред. физ.-мат. лит., 1984. 384 с.
7. Танаев В.С., Сотсков Ю.Н., Струсевич В.А. Теория расписаний. Многостадийные системы. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 328 с.
8. Танаев В.С., Ковалев М.Я., Шафранский Я.М. Теория расписаний. Групповыетехнологии. Минск: Институт технической кибернетики НАН Беларуси, 1998. 290 с.
9. Alon N., Woeginger G.J., Yadid T. Approximation schemes for scheduling on parallel machines // J. of Scheduling.– 1998.– V. 1.– P. 55 – 66.
10. Van de Akker J.M., Hoogeveen J.A., van de Velde S.L. Parallel machine scheduling by column generation // Oper. Res.– 1999.– V. 47, N 6.– P. 862 – 872.

Інтернет ресурси

11. Oracle – Hardware and Software docs. – Режим доступу: www.oracle.com/. - Дата доступу: 19.05.2016.

12. Wikipedia. – Режим доступу: https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm) . - Дата доступу: 08.05.2016.
13. Noureddin Sadawi. – Режим доступу: https://www.youtube.com/channel/UCNYv4HA3WjV3gZGLfBehRWQ. - Дата доступу: 27.05.2016.
14. SiteMarker.Ru Интернет технологии. – Режим доступу: http://sitemaker.ru/technologies/database/mysqlvspostgresql/. - Дата доступу: 19.05.2016.
15. Spring Framework Overview. – Режим доступу: http://www.tutorialspoint.com/spring/spring_overview.htm. - Дата доступу: 02.06.2016.
16. Принцип MVC в веб программировании. – Режим доступу: http://folkprog.net/printsip-mvc-u-web-programmirovanii/.
17. Spring security. – Режим доступу: http://projects.spring.io/spring-security/.
18. Hibernate (framework). – Режим доступу: https://en.wikipedia.org/wiki/Hibernate_(framework).

ДОДАТКИ

Додаток А

Програма та методики тестування

При тестуванні Java додатків існує основні два підходи

Перший підхід говорить, що будь-яка існуюча логіка повинна тестуватися в повній ізоляції від будь-якої іншої логіки. Іншими словами, потрібно створювати повністю ізольоване оточення для кожної частини бізнес-логіки. При тестуванні сервісів будь-які інші класи, використані , щоб викликати ще якісь частини логіки, трансформації і т.п., повинні бути ізольовані. Тобто цей підхід полягає у тому, що всюди слід використовувати моки і тестувати в повній ізоляції. У цього підходу є чудова перевага: в тесті ви можете емулювати будь-які умови, які тільки можна придумати. Треба вам, протестувати, кидання помилки будь-якого роду (навіть таку, яку дуже важко симулювати в реальному житті). Це дає певну свободу. І плюси підходу явно видно: тести виконуються дуже швидко, з ними просто. І тут існує величезна кількість бібліотек.

Другий підхід говорить зворотнє. Якщо ми тестуємо логіку, то ми повинні тестувати її цілком з усім оточенням, за винятком досить складних структур, наприклад, зовнішніх систем - їх потрібно якимось чином ізолювати. У цього підходу також є певні плюси. Один з них - в тому, що ви одночасно тестуєте не тільки саму логіку, але ще і інтеграцію з оточенням. Це означає, що тести будуть більш показовими, дадуть більше інформації про ситуацію, якщо щось пішло не так. Адже часто буває, що всі юніт-тести проходять, але додаток навіть не запускається. А причина дуже проста: кожен елемент окремо мав якийсь свій протокол, але коли вони зібралися разом, виявилося, що для емуляції поведінки ви використовували старий протокол, та нічого не працює. Даний підхід захищає від такого розвитку подій.

Але що відбувається, якщо бізнес-логіка володіє якоюсь витіюватістю (є логічні умови, цикли та інше), причому у вкладених шматочках логіки, яка делегована комусь, теж є цикли і вкладеність. В цьому випадку у вас з'являється дуже багато тестів, тому що треба пройти всі шляхи. Можна прийти до того, що для тестування одного сервісного методу доведеться написати близько п'ятдесяти тестів. Звісно, ніхто не буде писати всі 50 тестів. Бажано написати 10 або 15 і на цьому зупинися, віддалившись від початкової мети - забезпечити нормальне покриття коду тестами, які дали б вам точне розуміння, що все йде добре.

Такі тести запускаються повільніше. Виходить, що це важко, але зате не виникає поширеної проблеми з моками.

Є багато бібліотек для моків: EasyMock, Mockito, JMock, Powermock, Spock і інші. У кожної є свої плюси і мінуси. Наприклад, один з мінусів Mockito - якщо ви його використовуєте, необхідно багато часу, по-зрівнянню з іншими фреймворками, щоб у ньому розібратися. Mockito працює за принципом spy, тобто записує всі дії, а ви можете частину логіки перевіряти, а частину не перевіряти.

EasyMock за замовчуванням перевіряє все. Об'єкти в даному фреймворку, коли ви його чогось не навчили (викликали якийсь метод, про який не попереджали, або раптом два рази викликали те, що обіцяли викликати один раз), він не працює, або видає помилку.

Найбільша проблема з моками виникає, коли у вас є якийсь сервіс, який насправді є інтегратором і сам ніякої логіки не містить, але всередині викликає багато коду інших.

Додаток Б

Керівництво користувача

При вході на сайт, при відкритті головного вікна буде видно стартову сторінку з гербом КНТЕУ та формою входу, де потрібно вводити свій логін та пароль. (Рис. Б.1.)

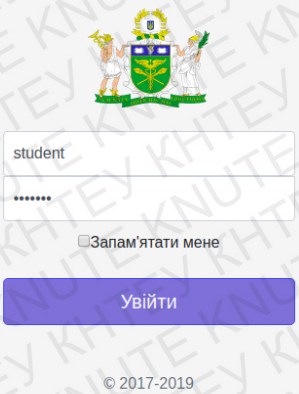

Рис. Б.1. Сторінка автентифікації

В залежності від ролі, що є у користувача (методист, студент, вчитель), буде відображено різне контекстне меню с лівого боку. Наприклад, для методистів там

відображено найбільше інформації, такої, як факультети, кафедри, студенти, групи з можливістю редагувати та переглядати дані сторінки. (Рис. Б.2.)

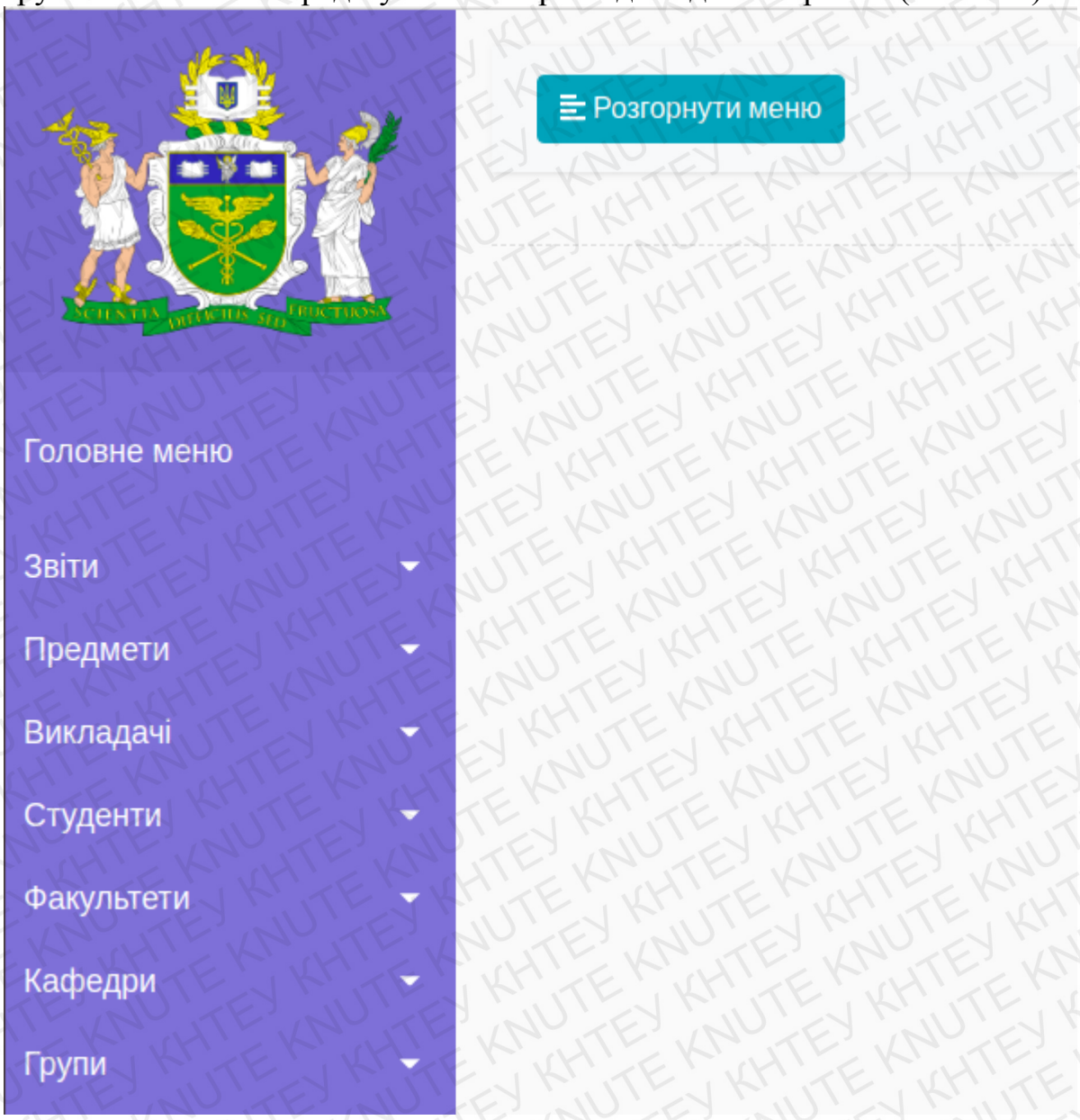

Рис. Б.2. Основна навігація.

Методисти можуть додавати дані про викладачів, студентів, факультети, кафедри та групи, редагувати існуючі дані, та витягати звіти по результатам сесії. При відкритті певної сторінки, наприклад сторінки студентів - випадає підменю з опціями, де при кліку межна перейти на сторінку списку (перегляд даних про студенті), додання даних про студента та редагування даних про студента. (Рис. Б.3.)

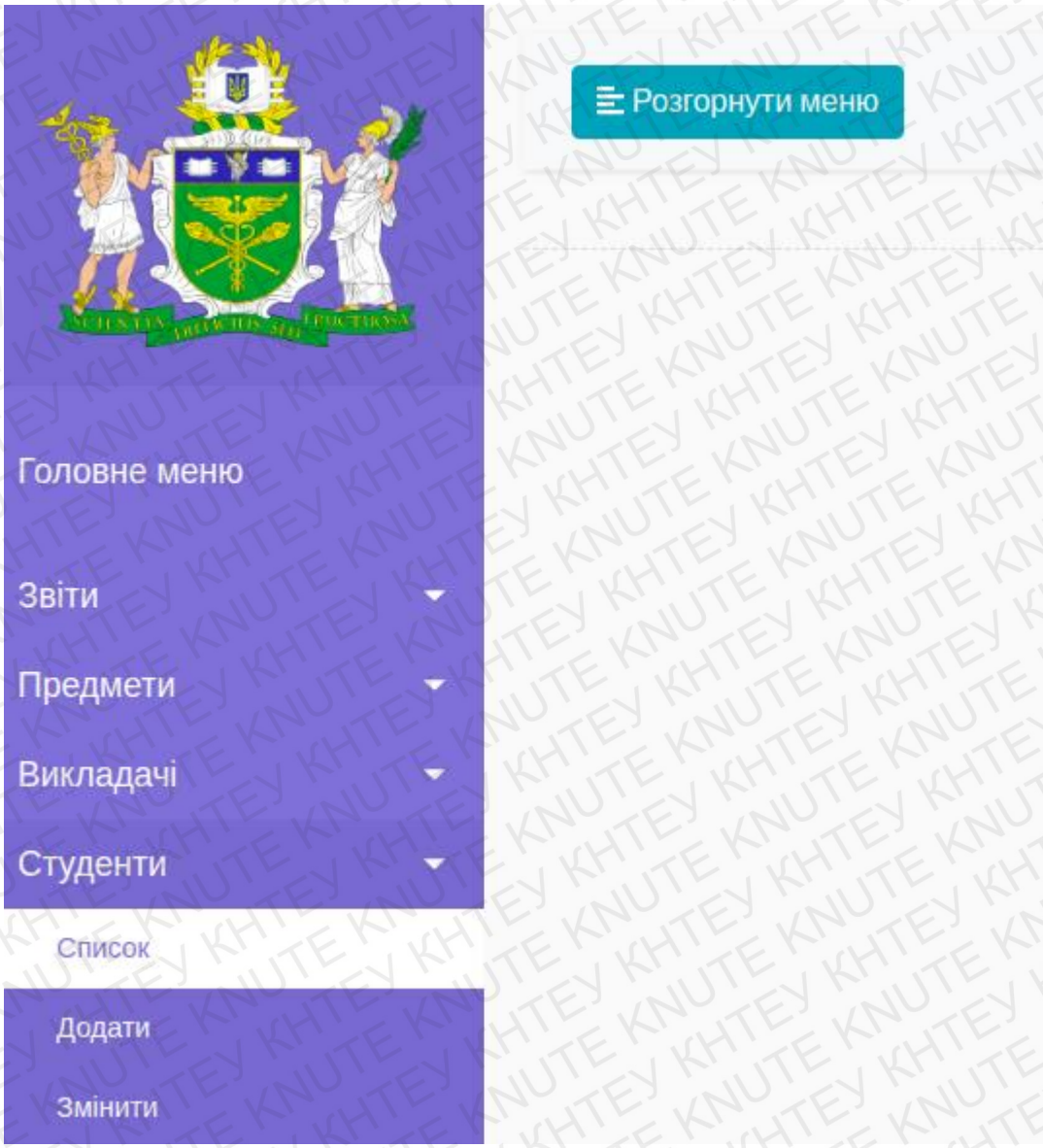

Рис. Б.3. Підменю студентів.

При переході на сторінку списку, можна побачити список всіх студентів, де буде показана основна інформація про студентів, як, наприклад: прізвище, ім’я, по-батькові, дата народження, контактна інформація та інше. Справа від списку студентів (таблиці) є поле для вводу пошук, де можна динамічно по всім полям шукати будь-які дані. (Рис. Б.4.)

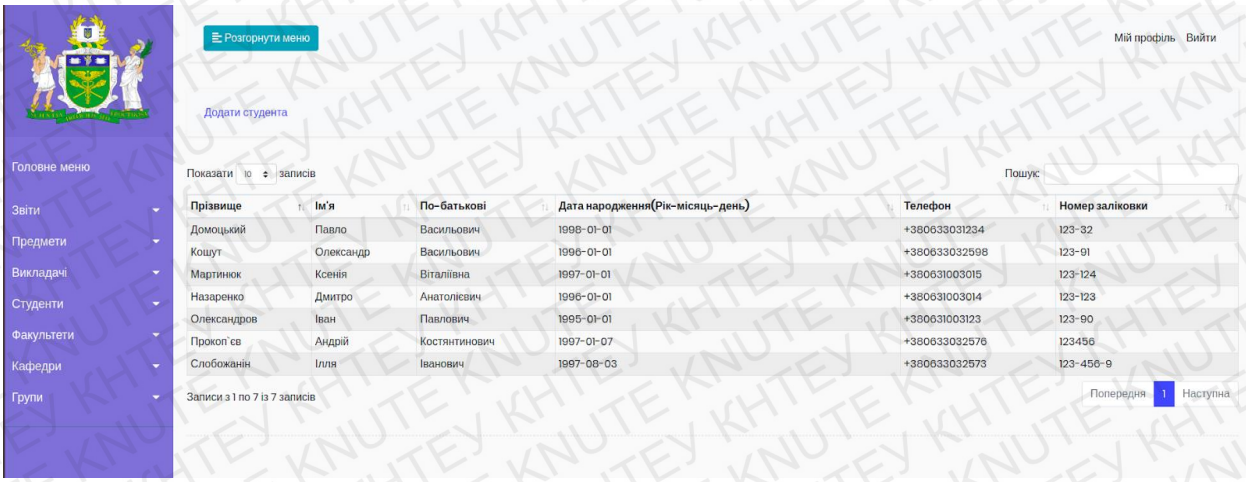

Рис. Б.4. Сторінка списку студентів.

При переході на сторінку додання даних про студента, ми побачимо сторінку з полями для вводу основних даних (Рис. Б.5.)

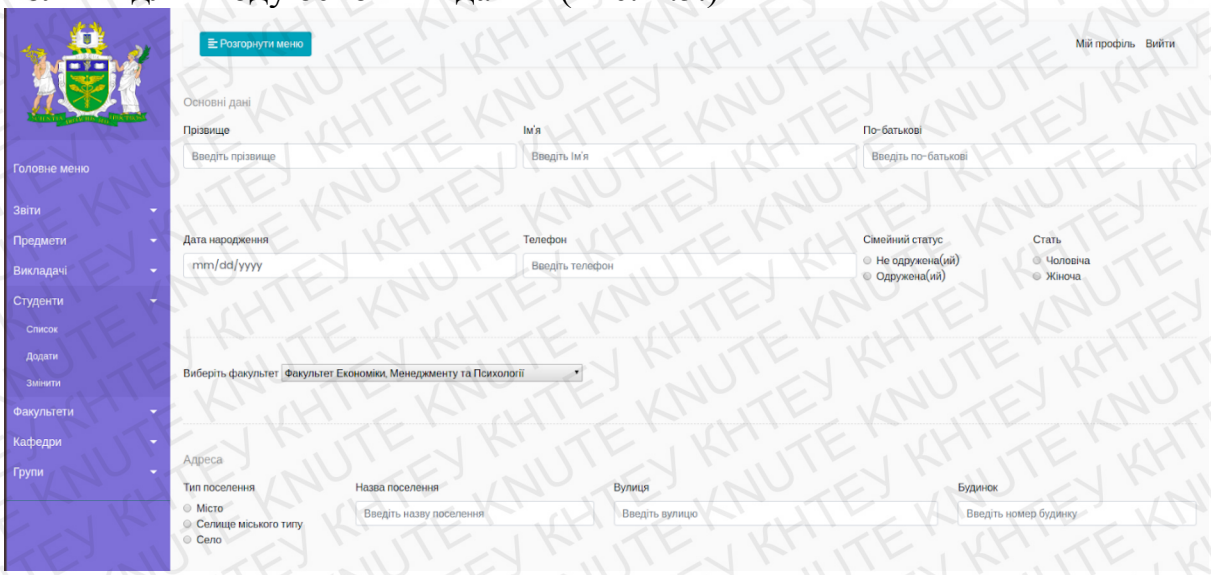

Рис. Б.5. Сторінка додання студентів.

Дані сторінки додання, редагування та таблиці є, майже, для всіх об’єктів, якими може оперувати методисти.

При автентифікації з роллю викладача - меню буде іншим, лише с даними про предмети та сторінками виставлення оцінок. При переході на сторінку предметів - можна побачити список всіх предметів викладача. Якщо відомість не закрита, можна змінити оцінки. (Рис. Б.6.)

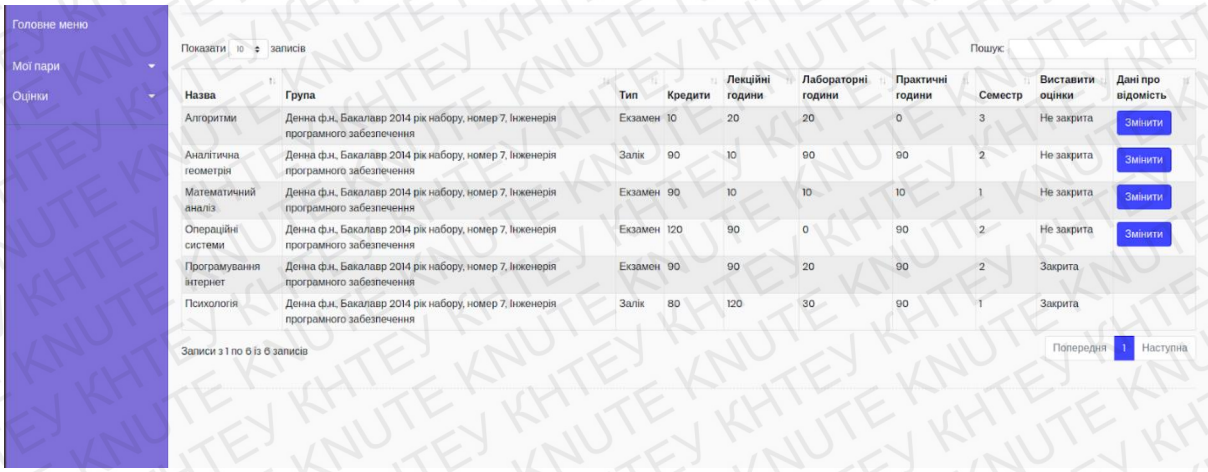

Рис. Б.6. Сторінка предметів.

Коли викладач вибирає, за який предмет виставити оцінку від побачить сторінку редагування оцінок. (Рис. Б.7.)

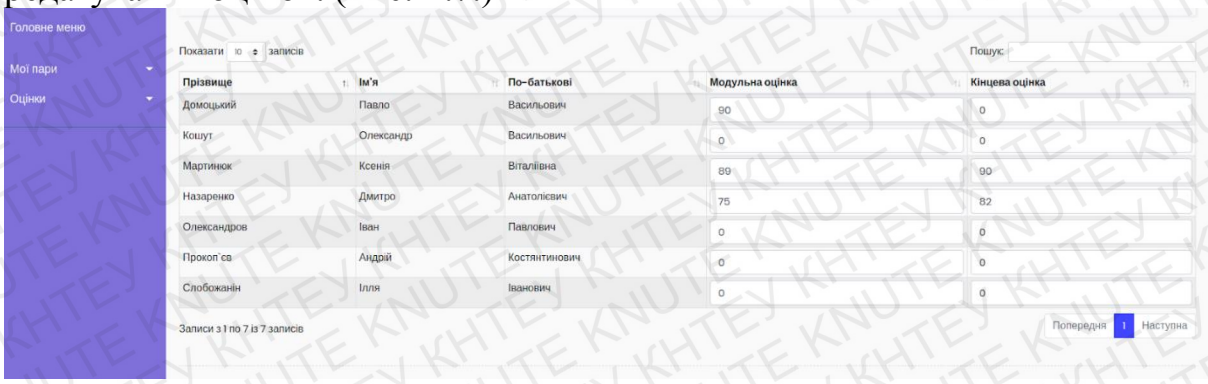

Рис. Б.7. Сторінка виставлення оцінок.

На сторінці виставлення оцінок - можна буде побачити список студентів та оцінок за модуль та фінальну (екзаменаційну) оцінку. При виставленні оцінку в таблицю - оцінка автоматично обновляється в базі даних. Та логується інформація, що викладач змінив оцінку такому студенту з такого предмета.

Додаток В

Програмний код додатку

Докер файл системи деканат

```
version: '2'
services:
  postgres:
```

```
  image: postgres
  container_name: postgres
  ports:
   - "5433:5432"
  environment:
   - POSTGRES_USER=postgres
   - POSTGRES_PASSWORD=postgres
  volumes:
   - ./scripts/create.sql:/docker-entrypoint-initdb.d/1-create.sql
   - ./scripts/fill.sql:/docker-entrypoint-initdb.d/2-fill.sql

 pgadmin:
  image: dpage/pgadmin4
  container_name: pgadmin
  ports:
   - "8081:80"
  environment:
   - "PGADMIN_DEFAULT_EMAIL=user@domain.com"
   - "PGADMIN_DEFAULT_PASSWORD=postgres"

  depends_on:
   - postgres
```

SQL скрипт створення бази даних

```
CREATE TABLE public.address (
  address_id integer NOT NULL,
  type text NOT NULL,
  city_name text NOT NULL,
  street text NOT NULL,
  house_num text NOT NULL
);

ALTER TABLE public.address OWNER TO postgres;

--
-- Name: address_address_id_seq; Type: SEQUENCE; Schema: public; Owner:
postgres
--
```

```
CREATE SEQUENCE public.address_address_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE public.address_address_id_seq OWNER TO postgres;

--
-- Name: address_address_id_seq; Type: SEQUENCE OWNED BY; Schema: public;
Owner: postgres
--

ALTER SEQUENCE public.address_address_id_seq OWNED BY
public.address.address_id;


--
-- Name: branch_of_knowledge; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.branch_of_knowledge (
    bok_id integer NOT NULL,
    bok_cipher integer NOT NULL,
    bok_name text NOT NULL
);

ALTER TABLE public.branch_of_knowledge OWNER TO postgres;

--
-- Name: branch_of_knowledge_bok_id_seq; Type: SEQUENCE; Schema: public;
Owner: postgres
--

CREATE SEQUENCE public.branch_of_knowledge_bok_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE public.branch_of_knowledge_bok_id_seq OWNER TO postgres;
```

```
--
-- Name: branch_of_knowledge_bok_id_seq; Type: SEQUENCE OWNED BY;
Schema: public; Owner: postgres
--

ALTER SEQUENCE public.branch_of_knowledge_bok_id_seq OWNED BY
public.branch_of_knowledge.bok_id;


--
-- Name: dec_group; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.dec_group (
    dec_group_id integer NOT NULL,
    year integer NOT NULL,
    group_num integer NOT NULL,
    specialty_id integer,
    department_id integer,
    study_form text NOT NULL,
    qualification text NOT NULL
);
```

Код додатку Java
Контроллер системи деканат

```
@Controller
@RequestMapping("/student")
public class StudentController {
  private final StudentService studentService;
  private final FacultyService facultyService;

  @Autowired
  public StudentController(StudentService studentService, FacultyService
facultyService) {
    this.studentService = studentService;
    this.facultyService = facultyService;
  }

  @GetMapping
  public String findAll(Model model) {

    model.addAttribute("students", studentService.getAll());
```

```
    return "student.html";
  }

  @GetMapping("/add")
  public String addStudent(Model model) {
    model.addAttribute("student", new Student());

    model.addAttribute("faculties", facultyService.getAll());
    return "addStudent.html";
  }

  @PostMapping("/add")
  public String addStudent(/*@Valid*/ Student student, BindingResult bindingResult,
Model model) {

    if (bindingResult.hasErrors()) {
      model.addAttribute("faculties", facultyService.getAll());
      return "addStudent.html";
    }

    studentService.save(student);

    return "redirect:/student";
  }
}
```

Сервіс фреймворку Spring

```
@Service
public class StudentServiceImpl implements StudentService {
  private final StudentRepository studentRepository;
  private final RoleRepository roleRepository;

  private final PasswordEncoder encoder;

  @Autowired
  public StudentServiceImpl(StudentRepository studentRepository,
                            RoleRepository roleRepository,
                            PasswordEncoder encoder) {
    this.studentRepository = studentRepository;
    this.roleRepository = roleRepository;
    this.encoder = encoder;
  }

  @Override
```

```
  public List<Student> getAll() {
    return studentRepository.findAll();
  }

  @Override
  @Transactional
  public Student save(Student entity) {
    DecUser user = entity.getUser();

    if (user == null) {
      user = new DecUser();

      String password = PasswordUtil.generatePassword();

      System.out.println(password);

      user.setPassword(encoder.encode(password));

      System.out.println(user.getPassword());

      user.setRole(roleRepository.findByRoleNameEquals("STUDENT"));
      entity.setUser(user);
    }

    return studentRepository.save(entity);
  }

  @Override
  public Student getById(Integer id) {
    return studentRepository.getOne(id);
  }

  @Override
  public List<Student> getStudentsByGroupByStudentUsername(String username) {
    return studentRepository.getStudentsByGroupByStudentUsername(username);
  }
}
```

Репозиторій доступу до бази даних

```
public interface StudentRepository extends JpaRepository<Student, Integer> {
    List<Student> findAllByGroupIs(DecGroup groupId);

    @Query("select s from Student s " +
            "inner join s.group g " +
```

```
        "where g.decGroupId = (select g.decGroupId from Student s inner join s.group g
" +
        "inner join s.user u " +
        "where u.login = :username) " + "" +
        "order by s.lastName")
    List<Student> getStudentsByGroupByStudentUsername(@Param("username")
String username);
}
```