Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

РОЗБРОБКА СИСТЕМИ УПРАВЛІННЯ ОСВІТНІМИ ПРОГРАМАМИ ФАКУЛЬТЕТУ

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення”

Керівник курсової роботи

д.т.н., доц. Глибовець А. М.

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2020 р.

Виконала студентка

Доцюк К. В.

(прізвище та ініціали)

“____” _____ 2020 р.

Київ 2020

ЗМІСТ

Вступ

Освітня програма факультету відіграє невід'ємну роль у навчальному процесі університету. Згідно з Положенням «Про організацію освітнього процесу в НаУКМА» навчальний план є нормативним документом НаУКМА, що визначає зміст та організацію освітнього процесу за конкретною спеціальністю на основі відповідної освітньо-професійної, освітньо-наукової програми. На основі навчального плану складаються робочі навчальні плани та індивідуальні навчальні плани студентів.

Нині університет отримує навчальний план як файл у форматі CSV, а зручний для прочитання людиною варіант вже формується вручну у форматі Word. За необхідності розрахунків за різними параметрами, що є у дисципліни, зокрема, кількість кредитів, належність до певної спеціальності, триместру, кількість лекційних годин та інших, необхідно вручну виокремлювати відповідні рядки даних та вести підрахунок.

Це робить актуальною переведення задачі з формування та управління навчальним планом в електронну форму. Це дозволить не тільки зекономити людський ресурс при побудові плану, але й надаватиме простий доступ до планів для викладачів, методистів та, за необхідності, інших працівників університету та студентів.

Серед технологій, що використовуються для розробки додатків такого типу, найбільш доречними є Java та Spring Framework для серверної частини, та Angular для клієнтської.

Такий вибір зумовлений, насамперед, тим, що університет уже використовує додатки, що розроблені з використанням, серед іншого, цих технологій, і ці додатки є функційними модулями електронної системи управління навчальним процесом загалом, тому новий додаток зможе простіше інтегруватися в систему, а також це спростить подальшу підтримку його функціонування, оскільки не потрібні будуть фахівці зі знаннями різних мов програмування для кожного функційного модуля.

Фреймворки Spring та Angular дозволяють спростити розробку, оскільки ці технології є популярними, існує велика кількість розробників, що їх розвиває та використовує, тому у відкритому доступі є опис багатьох підходів до вирішення різних проблем. Крім того, вони дозволяють робити додатки масштабованими та приводити у відповідність до принципів SOLID.

Робота складається з двох розділів.

Перший розділ присвячено характеристиці технологій та архітектури додатку: REST Api, що є загальноприйнятим підходом до проектування клієнт-серверних додатків; Spring, що є фреймворком, який використовується для розробки серверної частини та Angular, що є фреймворком, який використовується для розробки клієнтської частини.

Другий розділ присвячено опису предметної області та практичної реалізації поставленої задачі: надано детальний опис реалізації, подані приклади діаграм та коду.

РОЗДІЛ 1: ХАРАСТЕРИСТИКА ТЕХНОЛОГІЙ ТА АРХІТЕКТУРИ

1.1 REST Api

REST є загальноприйнятим скороченням для Representational State Transfer, або передача репрезентативного стану і являє собою підхід до архітектури мережевих протоколів, що був описаний Роєм Філдінгом у дисертації 2000 року “Архітектурні стилі та дизайни архітектур, пов’язаних з мережею”.

Загалом REST API є системою, що побудована на основі HTTP протоколу, що використовується для оформлення даних, що передаються від клієнта на сервер.

Основними обмеженнями, які визначає такий архітектурний підхід для веб-сервісу, є такі:

- Однотипний інтерфейс;
- Клієнт-серверна структура;
- Відсутність стану;
- Кешування;
- Багаторівневість;
- Запитування коду.

Однотипний інтерфейс є одним з наріжних каменів REST архітектури. Він означає, що залежності між клієнтом, сервером та проміжними рівнями між ними визначаються через спільні інтерфейси. Комунікаційна система буде зруйнована, якщо один із компонентів порушить стандарти спільних інтерфейсів.

Рой Філдінг визначив чотири основні принципи для стабільної взаємодії мережевих компонентів;

- Ідентифікація ресурсів;
- Маніпуляція ресурсами через репрезентацію;
- Самодескриптивні повідомлення;
- Гіпермедіа як основа стану додатку, або HATEOAS [1].

Ідентифікація ресурсів означає, що кожний веб-компонент повинен мати унікальний ідентифікатор. Наприклад, URI (Unified resource identifier) домашньої сторінки https://distedu.ukma.edu.ua однозначно визначає стартову сторінку ресурсу дистанційного навчання у НаУКМА.

Маніпуляція ресурсами через репрезентацію означає, що один і той самий ресурс може відображатися для клієнтів різним чином. Так, документ може бути відображений як HTML у веб-браузері та як JSON у програмі. Головна ідея полягає в тому, що репрезентація дозволяє взаємодіяти з ресурсом, але є не самим ресурсом. Ця концептуальна відмінність дозволяє репрезентувати ресурс різним чином, не змінюючи його ідентифікатор [2].

Бажаний стан ресурсу може бути репрезентовано у запиті клієнта. Наявний стан ресурсу може відображатися у повідомленні-відповіді, що повертається із серверу. Такі повідомлення можуть включати метадані, що містять додаткову інформацію про стан ресурсу, формат відображення, розмір, повідомлення тощо. HTTP повідомлення забезпечує заголовки, що організовують різні типи метаданих у одноманітні поля.

Репрезентація стану ресурсу містить посилання на пов’язані ресурси. Таким чином посилання створюють веб-павутиння між різними користувачами [2].

Клієнт-серверна архітектура дозволяє зробити незалежною розробку клієнтської та серверної сторони. Клієнт при цьому надсилає запити серверу, а сервер забезпечує надання сервісів. Сервером одночасно можуть користуватися декілька клієнтів, а комунікація між ними має регулюватися спільними правилами. Крім розділення відповідальності за зони розробки клієнт-серверна архітектура дозволяє:

1) Покращити портативність інтерфейсу користувача
2) Покращити масштабованість, спрощуючи реалізацію серверу
3) Розробити автономні компоненти, що можна тестувати [3].

Відсутність стану означає, що сервер не має запам’ятовувати стан сторони клієнта. Клієнт відповідальний за збереження актуальної для нього

інформації на своїй стороні. За потреби клієнт відправляє необхідну інформацію серверу. Це дозволяє серверу обслуговувати потреби широкої категорії користувачів, а не підлаштовуватися під вимоги лише одного. Загалом взаємодії HTTP стосуються двох типів станів:

1) Стан застосунку: дані, що зберігаються на стороні серверу і допомагають зрозуміти з контексту та попередніх запитів новий запит від клієнта;

2) Стан ресурсу: стан серверу в будь-який час.

Обмеження REST стосуються лише стану застосунку. Переваги цього обмеження полягають у тому, що спрощується масштабованість застосунку, не потрібно синхронізувати стан одразу на клієнтській та серверній стороні; підвищується надійність, оскільки застосунок може відновитися після часткової відмови. Проте, кожен запит має містити додаткову інформацію для того, щоб його можна було роспарсити.

Кешування є важливим принципом, що вимагає декларувати можливість збереження даних, до яких клієнт часто звертається. Це знижує затримку у відповіді, а також загалом “ціну” веб-зв’язку [2]. Кеш може існувати будь-де на шляху від клієнта до сервера.

Багаторівневість системи означає, що між клієнтським рівнем та рівнем серверу можна поміщати додаткові рівні, що перехоплюють комунікацію між ними та виконує додаткові функції. Такими функціями можуть бути посилення безпеки, кешування відповіді на запит, балансування завантаження [2].

Запитування коду означає, що зі сторони клієнта може надходити дозвіл на завантаження додаткового коду для розширення функціональності. Це обмеження може не бути потрібним застосунку, тому його вважають опціональним. Прикладами, що реалізують цей принцип, може бути використання Java аплетів, технологій JavaScript та Flash.

На рисунку 3.1 зображена схема застосування цих проектувальних обмежень.

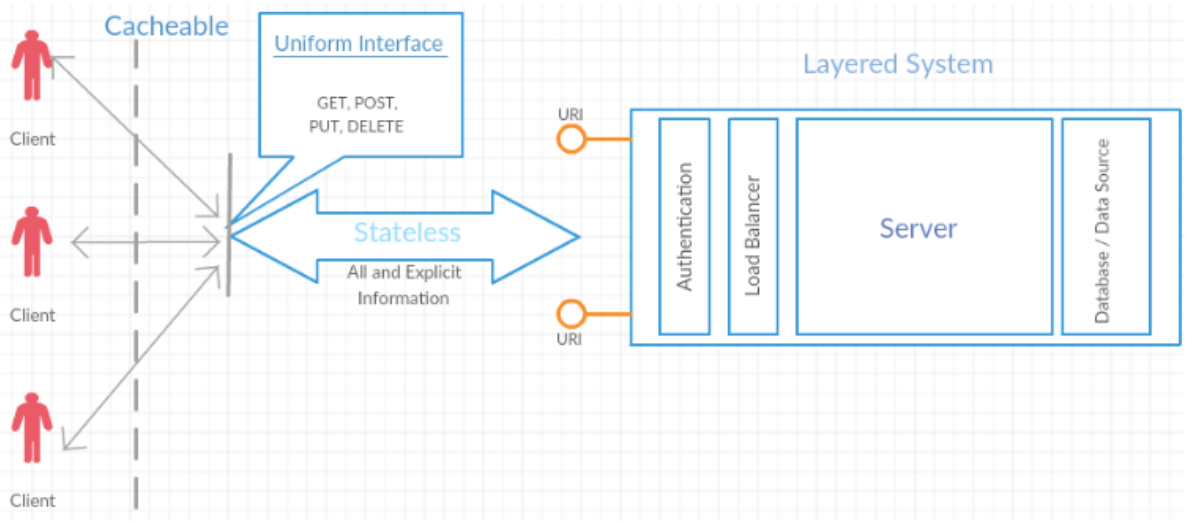


Рис. 3.1 - схема REST системи [hands on 4]

2.1 Фреймворк Spring

2.1.1 Spring

Spring з’явився у 2003 році та став відповіддю на складність перших специфікацій J2EE (платформа, що надає API та середовище для розробки та виконання корпоративного програмного забезпечення). Деякі розглядають J2EE та Spring як конкурентів, проте, розробники Spring вважають його доповненням до першого. Spring не включає всі компоненти J2EE, а інтегрується в окремі його специфікації, такі як: Servlet API, WebSocket API, Concurrency Utilities, JSON Binding API, Bean Validation, JPA, JMS, JTA/JCA. На користь цього свідчить й те, що у Spring можна використовувати впровадження залежностей та загальні анотації замість спеціальних механізмів, передбачених фреймворком [4].

Використовуючи термін Spring, можуть мати на думці:

- проект, що використовує Spring Framework, при цьому з часом нові проекти можуть будуватися на його основі;

- сукупність проектів, що створені з використанням Spring.

Основними принципами, на які спирається фреймворк, є такі:

1) Дати можливість вибору на кожному рівні

2) Врахувати різні перспективи

3) Підтримувати сильну зворотну сумісність

4) Інтуїтивно зрозумілий API

5) Високі стандарти якості коду.

Фреймворк Spring поділений на модулі і користувач може обирати, які з них використовувати.

Spring використовується для створення загальноорганізаційних додатків (enterprise application). Він підтримує Groovy та Kotlin як альтернативні мови для віртуальної Java машини, а також дозволяє створювати різні типи архітектур додатків залежно від конкретних потреб. Остання версія фреймворку вимагає мінімальну версію Java 8. Фреймворк є повністю сумісним з Tomcat 8 та 9, WebSphere 9 тощо.

Spring дозволяє підтримувати різні типи додатків. Наприклад, у великих компаніях додатки зазвичай існують протягом довгого часу, виконуються на JDK сервері та їх цикл оновлення має знаходитися під контролем розробниці. Інші додатки можуть працювати як один jar зі вбудованим сервером у хмарному середовищі або бути автономними та не потребувати серверу.

Spring постійно розвивається, існують такі проекти як Spring Boot, Spring Security, Spring Data, Spring Cloud, Spring Batch та інші.

Фреймворк Spring включає близько 20 модулів. Залежно від основних характеристик, їх можна згрупувати в основний модуль (Core Container), модуль доступу до даних (Data Access/Integration), веб-модуль (Web), аспектно орієнтований (Aspect Oriented Programming), тестовий [5]. Схему модулів Spring можна побачити на рисунку 2.1.1.1.

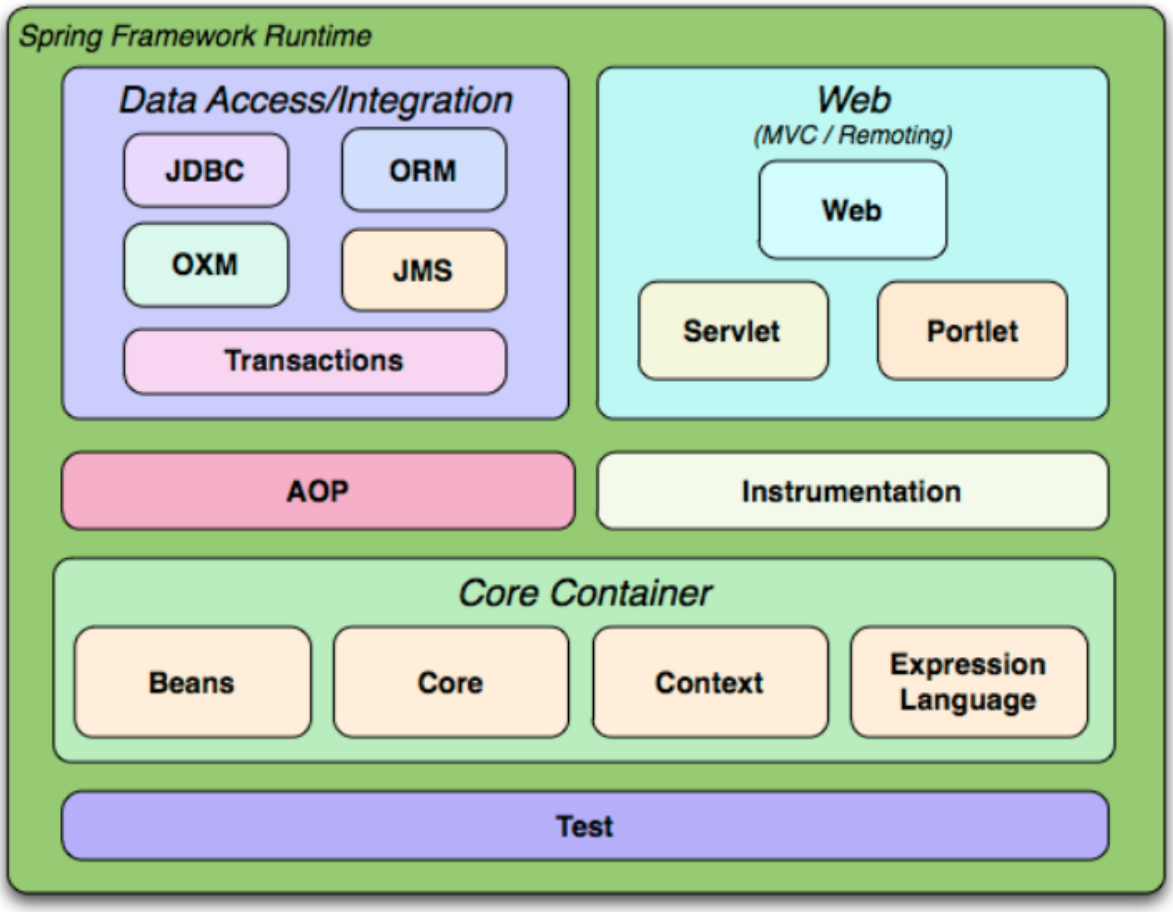


Рисунок 2.1.1.1 – Модулі Spring [5].

Основний контейнер складається з модулів Core, Beans, Context та Expression.

Він включає найважливіші частини фреймворку та реалізує зворотній контроль (inversion of control) і впровадження залежності (dependency injection). Головну роль тут відіграє BeanFactory, що відповідає за впровадження патерну фабрика. Це дозволяє відділити конфігурацію та специфікацію залежностей від програмної логіки.

2.1.2 Spring Boot

Спочатку Spring Boot можна було конфігурувати лише використовуючи розширювану мову розмітки (далі - XML). Проте, пізніше додали можливість

Java-конфігурації. Також можна комбінувати два підходи навіть в одному додатку.

Для спрощення початкової конфігурації проекту, Spring Boot робить більшу її частину без участі користувача. Зокрема він автоматично визначає різні компоненти, використовуючи певні критерії, наприклад:

доступність певного класу в шляху до класів;

- наявність або відсутність спрінг біну;
- наявність системної властивості;
- відсутність конфігураційного файлу.

Наприклад, якщо в шляху класів є вбудований драйвер бази даних, а DataSource відкрито сконфігуровано не було, Spring Boot автоматично зареєструє бін DataSource, використовуючи налаштування, що є в пам’яті.

Spring Boot дозволяє використовувати автоконфігурацію. Ключовою для цього є анотація @EnableAutoConfiguration, яка, зазвичай, використовується у класі, що запускає додаток. Ця анотація дозволяє конфігурувати контекст фреймворку тим, що сканує шлях до класів-компонентів та реєструє біни, що відповідають різним критеріям.

Для впровадження залежностей найчастіше використовується анотація @Autowired. Також є можливість використовувати @Resource та @Inject, проте, лише перша анотація належить саме фреймворку Spring [9]. Принципова різниця між ними полягає лише в порядку перевірки шляхів для знаходження бінів. Проте, для @Autowired та @Inject порядок однаковий: спочатку перевіряє збіг за типом, потім за специфікатором, і нарешті за іменем. Для @Resource визначений порядок: ім’я, тип, специфікатора.

Анотації позбавляють необхідності писати зайвий код та роблять його більш простим та читабельним, але можуть приховати залежності від користувача класу та зменшують переносимість додатку, оскільки він стає сильно прив’язаним до фреймворку Spring.

1.2 Фреймворк Angular

Ангуляр є платформою та фреймворком для побудови односторінкових клієнтських додатків з використанням HTML та TypeScript. Основні функції в ньому реалізовані як бібліотеки Typescript, що користувачка імпортує у свої додатки.

Основними блоками, з яких складається Angular є модулі NgModules, які забезпечують контекст для компіляції компонентів. Зазвичай додаток складається з хоча б одного кореневого модуля та включає багатьох функційних модулів.

Компоненти визначають представлення (views), які можна модифікувати залежно від потреб програми. Також вони використовують сервіси, які реалізують специфічні функції, які напряму впливають на представлення.

Сервіси та компоненти є звичайними класами. Декоратори дозволяють їх розрізняти через метадані, які “повідомляють” Ангуляр про те, як їх використовувати: асоціюють компонент з шаблоном, який визначає представлення. Шаблон поєднує звичайний HTML з директивами та розміткою, що дозволяє Ангуляру модифікувати HTML перед рендерингом і показом. Сервіс дозволяє реалізувати впровадження залежностей.

Окремо можна виділити сервіс Router, що дозволяє визначити поєднання компонентів та представлень ієрархічно.

Модулі Ангуляра - NgModules - доповнюють модулі JavaScript. Вони декларують контекст компіляції для певного набору компонентів, які об’єднані певним полем застосування чи робочим процесом, NgModule може також асоціювати компоненти зі спорідненим кодом, наприклад, сервісами, для формування функційних юнітів.

Кожен додаток, написаний з використанням Ангуляра, має кореневий модуль, який зазвичай називають AppModule, який запускає додаток. Він має таку назву, оскільки може включати багато дочірніх модулів в ієрархії будь-

якої глибини. А функційних модулів у додатка зазвичай багато. Кореневий модуль зв’язує усі компоненти з об’єктною моделлю документу.

Модулі Ангуляра можуть імпортувати функції інших модулів та дозволяти експорт своїх функцій.

Загалом модулі допомагають структурувати код і підвищити ефективність виконання програми за рахунок лінивого завантаження, для мінімізації кількості коду, що має бути завантажена при запуску застосунку.

NgModule створюється як клас, позначений декоратором @NgModule(). Цей декоратор є функцією, що приймає один об’єкт метаданих, чиї властивості описують модуль. Одними з найважливіших властивостей є такі:

- декларації : компоненти та директиви, що належать модулю;
- експорти: підмножина декларацій, що повинні бути видимими та можуть бути використані шаблонами компонент, що задекларовані у відповідному модулі;
- імпорти : інші модулі, чиї експортовані класи мають використовуватися у цьому модулі.

Модулі забезпечують контекст компіляції для компонентів. Кореневий модуль зазвичай містить кореневий компонент, який створюється під час завантаження, хоча будь-який NgModule може включати будь-яку кількість додаткових компонентів, які можуть бути завантажені через роутер або створені через шаблон. Компоненти, що належать модулю, поділяють контекст компіляції.

Компонент і його шаблон визначають представлення. Компонент може включати ієрархію представлень. Таким чином можна конфігурувати довільні частини відображень.

Компонент позначається декоратором @Component. Найбільш важливими конфігураторами компонент є: селектор: CSS селектор, що повідомляє ангуляр про те, що треба помістити екземпляр цього компонента там, де знайдеться відповідний тег у шаблоні HTML, templateUrl.

Шаблон дозволяє змінювати елементи HTML перед тим, як вони будуть продемонстровані на екрані. Для цього він поєднує HTML з розміткою Ангуляра. Існує два види зв’язування даних:

Зв’язування подій дозволяє додатку реагувати на ввід користувача, оновлюючи дані додатку;

Зв’язування властивостей, що дозволяє вирахувати дані у додатку і вже потім показати їх за допомогою HTML.

Ангуляр підтримує подвійний зв’язок (two-way data binding). Це означає, що зміни в DOM, наприклад, вибір користувачки, відображаються у даних програми.

Шаблони Ангуляра є динамічними. Коли Ангуляр обробляє їх, він трансформує DOM відповідно до інструкцій, наданих директивами. Вони є класами з декоратором @Directive(). Фактично компонент теж є директивою і його декоратор є просто розширенням декоратора директиви. Йому дали окрему назву через важливу роль концепції компоненти.

Крім цього існують структурні директиви, які заміняють елементи DOM для того, щоб додати спеціальну логіку для обробки представлень; а також атрибутні директиви, наприклад, ngModel, що впроваджує подвійне зв’язування.

Сервіс є широкою категорією і може реалізувати певну, зазвичай вузьку, функцію, що потребує додаток.

Ангуляр відділяє компонент від сервісу, оскільки вони служать різній меті. Компонент має лише забезпечувати “зовнішню” взаємодію користувача з додатком. Він може передавати сервісу завдання з отримання даних із сервера, валідації вводу користувача, логування тощо. Провадження залежностей допомагає використовувати один сервіс у декількох компонентах.

Для визначення класу як сервісу потрібен декоратор @Injectable(), що одразу надає метадані, що дозволяють впроваджувати відповідний сервіс як залежність. Для цього Ангуляр створює інжектор для всього додатку при його

завантаженні. Користувач окремо його не створює. Інжектор створює залежності і підтримує контейнер їх екземплярів, які використовуються повторно, якщо це можливо. Також в цьому процесі бере участь постачальник (Provider), що є об’єктом, який “повідомляє” інжектору, як отримати певну залежність. Для кожної залежності потрібно створювати такий провайдер. У випадку сервісу провайдер співпадає із самим класом сервісу.

Під час створення нового екземпляру класу компонента, Ангуляр визначає, яких сервісів чи інших залежностей він потребує, звертаючись до типів параметрів конструктора.

РОЗДІЛ 2: РЕАЛІЗАЦІЯ РОЗРОБКИ СИСТЕМИ УПРАВЛІННЯ ОСВІТНІМИ ПРОГРАМАМИ ФАКУЛЬТЕТУ

2.1 Опис предметної області

Для демонстрацій можливостей Spring Boot та Angular було обрано проект з реалізації системи управління навчальними планами.

Для входу в систему використовується поштова адреса Microsoft Outlook, за допомогою якої користувачі авторизуються, а також визначаються їхні ролі. Всього таких ролей 3: адміністратор, методист, викладач. До унікальних прав адміністратора належить право додавати навчальний план та дисципліни. До унікальних прав методиста належить право змінювати дисципліни. Викладачі можуть переглядати плани за різними критеріями.

Завантажувати новий план можна у вигляді файлу у форматі CSV. Завантажуючи план, потрібно обрати рік, для якого він буде дійсним. Фільтрувати дисципліни можна за різними критеріями:

• кількість кредитів;

• кількість лекційних годин;

• кількість семінарських годин;

• кількість годин самостійної роботи;

• номер семестру;

тип спеціальності, до якої належить дисципліна.

При спробі змінити кількість кредитів дисципліни, або видалити дисципліну, робиться перевірка на відповідність суми кредитів дисциплін відповідного формату після змін мінімально необхідному обсягу.

Дисципліни можуть бути таких форматів:

нормативні навчальні дисципліни;

дисципліни професійної та практичної підготовки;

дисципліни вільного вибору студента;

практика;

атестація.

Дисципліна пов’язана із триместром, в якому вона викладається. Таких триместрів може бути декілька, а всього їх 12. Для триместрів, у якому викладається дисципліна, визначається кількість годин, що припадає на кожний тиждень у цьому триместрі.

По закінченню прослуховування дисципліни може бути декілька видів підсумкового контролю, а саме:

• екзамен;

• залік;

• атестація;

• захист тези.

2.2 Практичні аспекти реалізації задачі

Додаток реалізовано відповідно до основних архітектурних принципів REST та включає клієнтську та серверну частини. На клієнтській стороні використовується Angular, а на серверній Spring Boot. В якості системи управління базою даних була обрана PostgreSQL.

Серверна частина реалізована як Spring Boot додаток та включає декілька рівнів, основні з яких характерні для стандартного патерну Model-View-Controller.

Класи організовані в декілька пакетів, їхній список показано на рисунку 2.2.1.

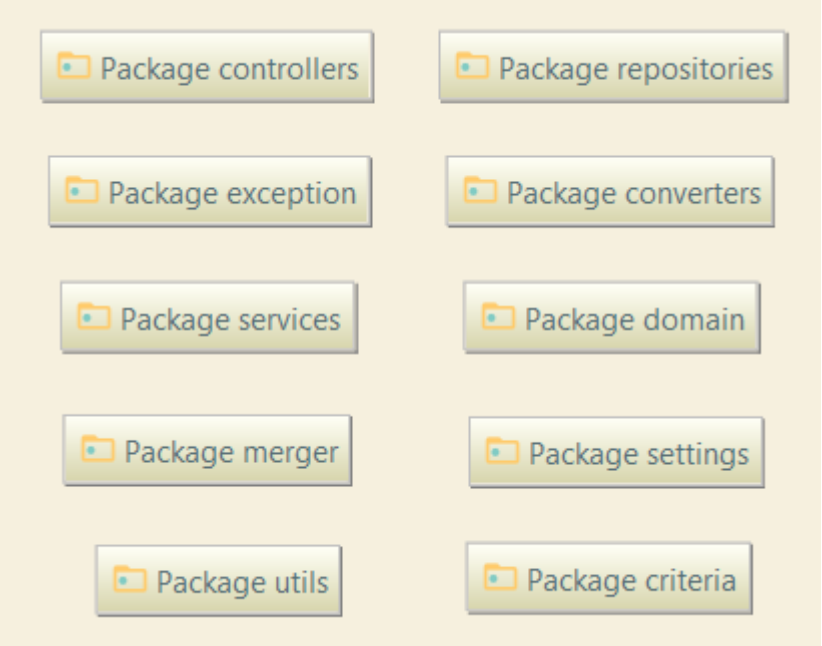


Рисунок 2.2.1 - список програмних пакетів

На рисунку 2.2.2 можна побачити, як виглядає пакет з контролерами. Усі класи, крім CSVContoller, розширюють BaseApiContoller, в якому визначені базові функції управління даними, які ще називають акронімом CRUD.

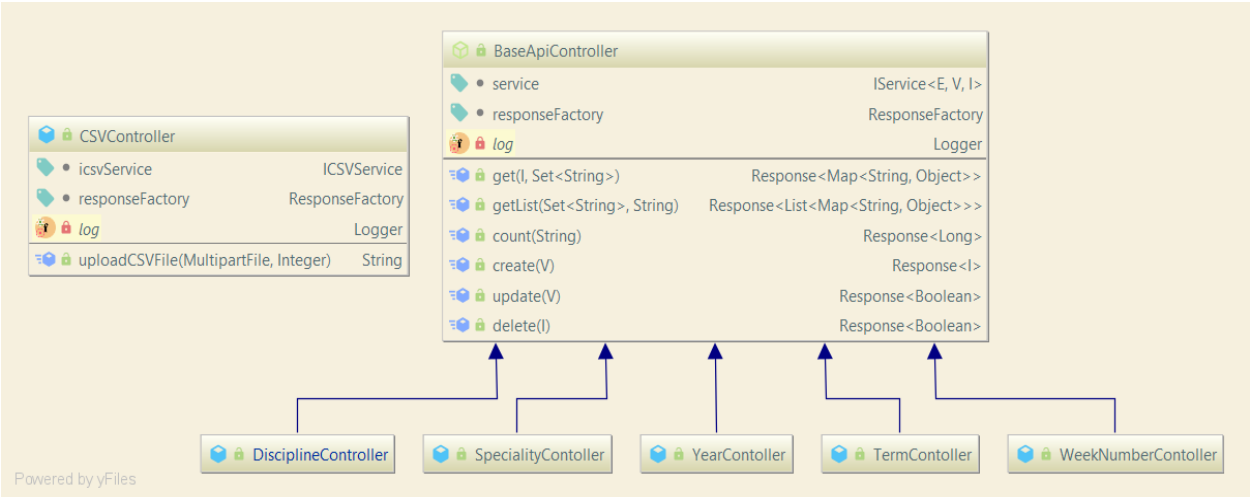


Рисунок 2.2.2 - діаграма класів-контролерів

Приклад методу з класу BaseApiContoller можна побачити на рисунку 2.2.3. Для його конфігурації використані звичні для Spring Boot анотації:

@RequestMapping, @ResponseBody, @RequestParam. Перша з них дозволяє визначити шлях, за яким можна звертатися до методу та тип запиту (у прикладі це GET).

@ResponseBody говорить контролеру про те, що об'єкт, який повертається, автоматично серіалізується в JSON перед тим, як передається до HttpResponse об'єкту [6].

@RequestParam означає, що параметр методу повинен бути зв'язаним з параметру веб-запиту. У Spring MVC параметри запиту можуть пов'язуватися з параметрами, що вказуються безпосередньо у запиті (query parameters), даними, що надходять у формі, а також файлами [7].

```
@ApiOperation("Get list of entities by restriction")
@RequestMapping(
        value = "/",
        method = RequestMethod.GET
)
public @ResponseBody
Response<List<Map<String, Object>>>
getList(
        @ApiParam(value = "fields that should be returned in response map")
        @RequestParam(value = "fields", required = false, defaultValue = Fields.DEFAULT) Set<String> fields,
        @ApiParam(value = "Entity restrictions")
        @RequestParam(value = "restrict", required = false) String restrict
) throws BaseException {
    return responseFactory.get(service.getList(fields, restrict));
}
```

Рисунок 2.2.3 - метод getList()

Зазначений метод повинен повернути список дисциплін за певними критеріями, якщо такі будуть вказані. Крім того, можна вказувати, які саме поля для дисципліни необхідно повернути. Це робиться за допомогою параметрів fields та restrict, що відповідно передаються як список об'єктів типу String та один об'єкт типу String, що повинен мати формат JSON.

Для повернення результату параметри передаються спочатку до методу сервісу. За допомогою анотації @Autowired реалізовано принцип

впровадження залежностей. Таким чином контролер може звертатися до необхідних йому інтерфесу IService та ResponseFactory.

IService імплементує клас BaseSevice, який, у свою чергу, розширюють класи сервісів, що відповідають окремим сутностям. Кожній сутності відповідає пакет, в якому є два інтерфейси (власне сервісу та валідатор) та їх імплементації. Діаграму класів-сервісів можна побачити на рисунку.

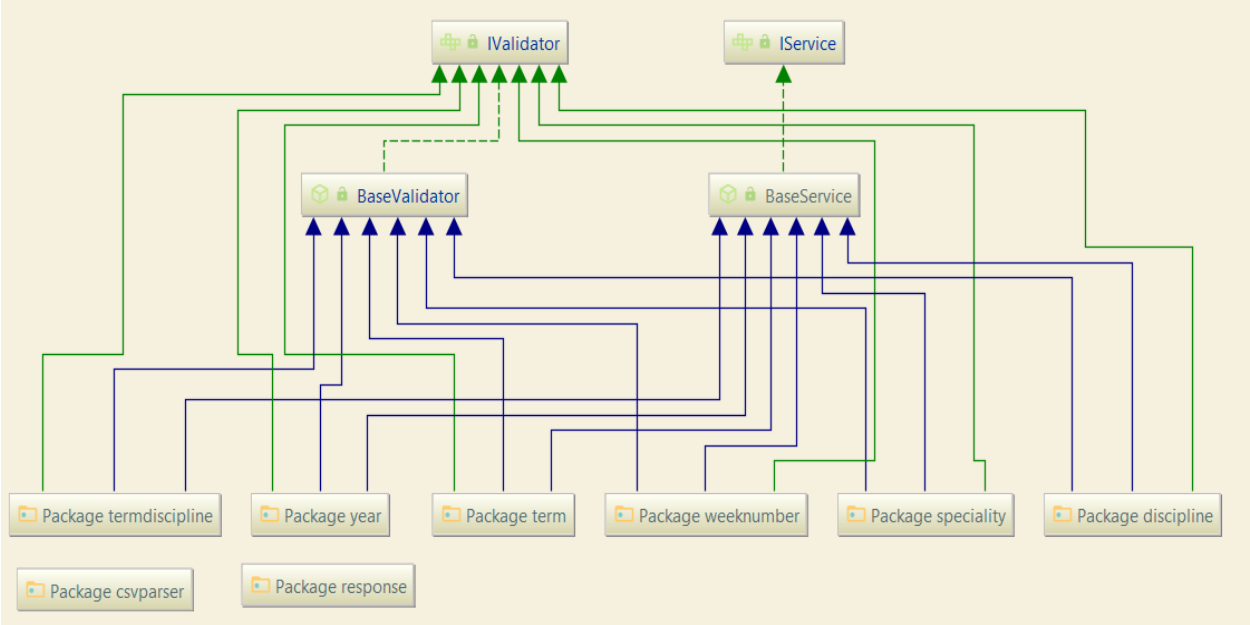


Рисунок 2.2.4 - діаграма сервісів

На прикладі методу getList() можна прослідкувати, як відбувається обробка запиту. У самому класі BaseService існує декілька перевизначень цього методу. Це зроблено для того, щоб спочатку розпарсити об’єкт, що визначає критерії, який був переданий як параметр restrict. Згодом викликається метод, в який вже передається об’єкт типу Criteria.

Запити з використанням Criteria дозволяють будувати запити без використання безпосередньо мови SQL. Замість цього використовується об’єктно-орієнтований підхід і побудова запитів здійснюється програмно. Оскільки однією з важливих вимог до системи є можливість фільтрації дисциплін за різними полями, використання Criteria є особливо доречним.

Для цього створено репозиторій ICriteriaRepository та клас, що його імплементує. Там здійснюється виклик методів для повернення результату. Приклад методу, що викликається для повернення списку дисциплін, показано на рисунку 2.2.5.

```
@Override
public <T> List<T> find(Criteria<T> criteria) {
    Query query = criteria.createQuery(entityManager);
    if(criteria.getOffset() > 0) {
        query.setFirstResult(criteria.getOffset());
    }
    if(criteria.getLimit() > 0) {
        query.setMaxResults(criteria.getLimit());
    }
    return query.getResultList();
}
```

Рисунок 2.2.5 - Метод find() класу CriteriaRepositoryImpl

Цей метод викликає createQuery, що формує запит, використовуючи обмеження, що були задані для конкретного типу сутності. Ці обмеження створюються як предикати. У класі Criteria задано початкове значення для зсуву та ліміту. Ці два поля дозволяють реалізувати пагінацію на клієнтській частині без створення додаткових запитів.

Після повернення результат передається методу класу Converter, який також є інтерфейсом, який імплементують конвертери, що відповідають різним базовим сутностям. Конвертер використовується для вибору лише тих полів, що визначив клієнт у запиті.

В разі створення чи оновлення полів сутності, в свою чергу, використовується клас Merger. Його роль полягає у приведення

представлення, яке отримано в тілі запиту, у відповідність із сутністю для того, щоб мати можливість безпосередньо додати її в базу даних.

Класи представлення створені, в основному, для можливості подальшого розширення та покращення функціоналу системи, оскільки з їхньою допомогою можна здійснювати конфігурацію сутностей відповідно до потреб клієнтської сторони. Це може додатково навантажити серверну частину в частині обробки даних перед їхнім збереженням або оновленням, але може зменшити загальну кількість запитів, які необхідно робити клієнту.

Для потреб системи необхідно створити можливість завантаження дисциплін з файлу. Для цього створено контролер CSVContoller, який наразі містить лише один метод для завантаження навчального плану як файлу у форматі CSV.

Для обробки файлу використовується бібліотека Apache Commons CSV. Вона є простою і зручною у використанні. Достатньо створити екземпляр класу CSVParser та передати йому екземпляр класу Reader, а також агумент, що дозволяє робити запит до колонок файлу за заголовком колонки. Далі викликається метод класу CSVServiceImpl. Його структуру можна побачити на рисунку 2.2.6. Там формується view для дисципліни та, шляхом звернення до відповідних сервісів, заповнюються дані з кожного рядка файлу.

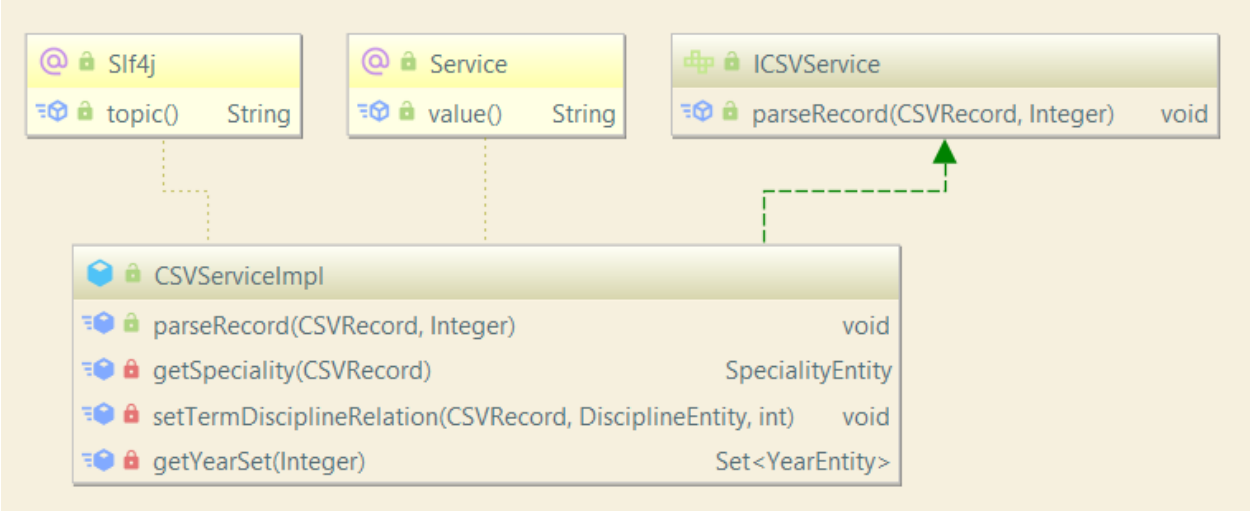


Рисунок 2.2.6 - Діаграма класу CSVServiceImpl

Нижче рівня сервісів знаходяться класи-репозиторії, що створені у відповідність до кожної основної сутності. Вони імплементують інтерфейс BaseRepository, а через нього розширюють клас JpaRepository. Так як JpaRepository надає доступ до CRUD методів без необхідності їх безпосередньої імплементації, а використання Criteria дає змогу фільтрувати результати за потрібними критеріями, необхідність написання запиту з використанням SQL не виникала.

У якості СКБД була обрана PostgreSQL. На рисунку 2.2.7 можна побачити схему бази даних.

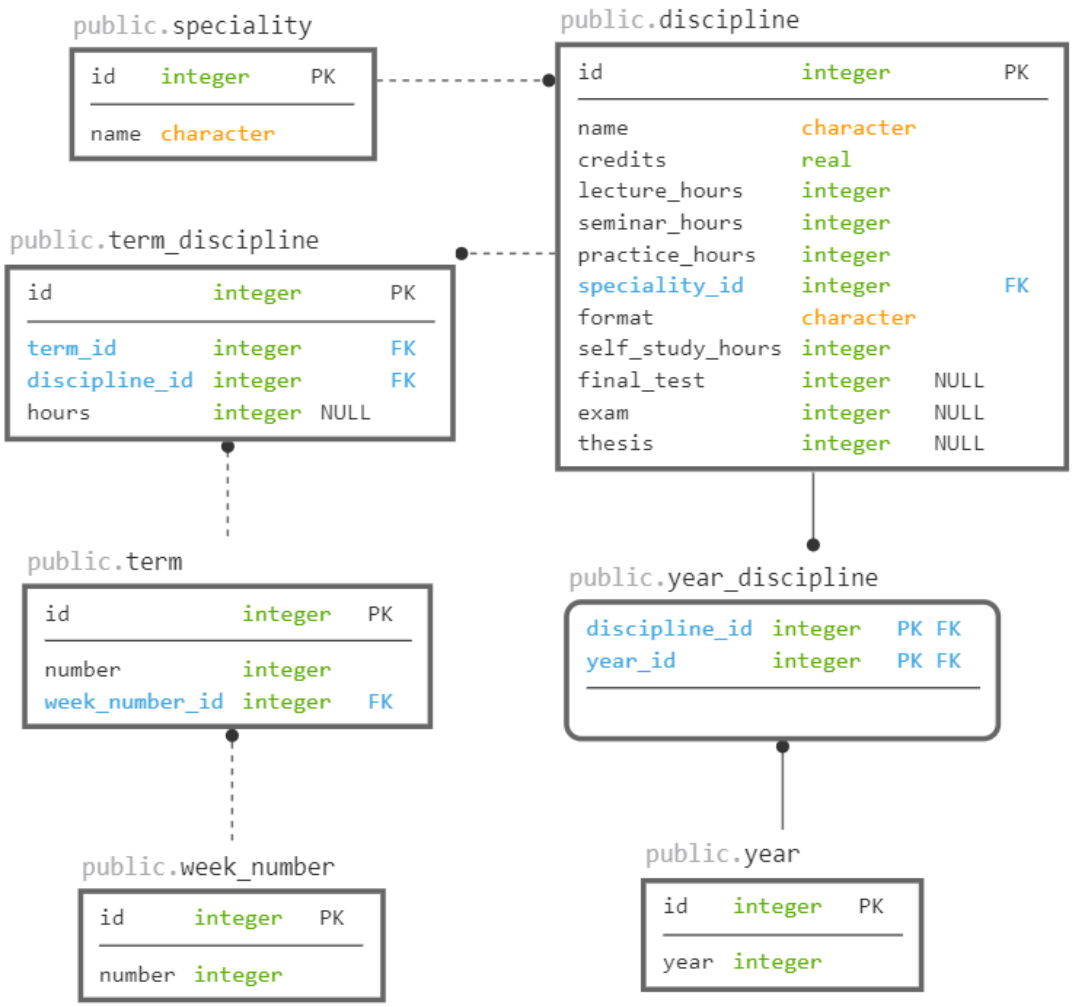


Рисунок 2.2.7 - Схема бази даних

Для реалізації клієнтської сторони використовувався фреймворк Angular. Створено, зокрема, такі компоненти:

• відображення картки з інформацією про дисципліни;

• відображення форми для додавання дисципліни;

• фільтрація та сортування за різними полями;

• відображення кнопок для завантаження нової дисципліни та дисциплін за обраними фільтрами;

• відображення статистики по дисциплінах, що повернулися за запитом.

Компоненти у проекті, зазвичай, складаються з трьох частин: файлів розширення html, sass та ts, які, відповідно визначають структуру сторінки, стиль та логіку функціонування динамічного наповнення. Формат SASS є розширенням CSS і його перевага в тому, що він дозволяє використовувати, зокрема, змінні, міксини, імпорти тощо [8].

В додатку А можна побачити діаграму класів і залежностей найбільшого компоненту - curricula-form-area.

Для виконання деяких вузьконаправлених функцій були створені декілька сервісів: сервіс, що відповідає за створення форми, що збирає дані, введені користувачем для відправки на серверну частину, який використовується лише компонентою discipline-form-dialog; сервіс для побудови форми з полями дисципліни для відображення на картці з інформацію про неї, а також сервіс для сортування результатів за різними полями, які використовуються спільно декількома компонентами пакету curricula-form-area.

Для обробки запитів було створено базовий сервіс та сервіси для завантаження файлу, управління запитів, пов’язаних з дисципліною, сервісом, триместрів та спеціальністю. Ці сервіси знаходяться на найвищому рівні додатку, оскільки використовуються різними компонентами. Діаграму пакету http-service, що містить вказані сервіси, можна побачити в додатку Б.

Пакет модуль містить класи, створені для зручності моделювання та управління даними. Крім типів, що відповідають основним сутностям на

фронті (discipline, speciality та term), створені моделі для відображення дисципліни в діалоговому вікні, полів для фільтрації та сортування, а також класи-обгортки для обробки http-запитів.

Висновки

В результаті виконання роботи був розроблений веб-додаток для управління та зручного доступу до освітніх програм університету, зокрема:

- реалізована можливість завантаження файлу у форматі CSV, який автоматично додається до бази даних дисциплін та пов’язується з навчальним роком, для якого такий план дійсний;
- реалізована можливість переглядати список усіх дисциплін, а також обирати критерії пошуку: за іменем, триместром, роком, спеціальністю, кількістю кредитів, лекційних, семінарських годин, годин для самостійного навчання, кількістю екзаменів та заліків;
- можна здійснювати сортування за обраними критеріями;
- можна редагувати поля дисципліни та видаляти дисципліну;
- при редагуванні чи видаленні дисципліни здійснюється перевірка на відповідність кредитів нового навчального плану мінімально необхідній.

Додаток надає базові можливості для управління освітніми програмами університету, тому першочергову задачу було виконано. Можливими шляхами покращення, є, зокрема, розширення кількості критеріїв валідації дисципліни при оновленні та видаленні, надання можливості порівнювати плани різних років, надання можливості копіювати навчальні плани попередніх років для створення нових, можливості збереження фільтрів. Завдяки зручній структурі програми, ці та інші функції можуть бути додані найближчим часом.

Перелік прийнятих скорочень

CRUD – create, remove, update, delete.

CSS – cascading style sheets.

CSV – comma separated values.

HATEOAS - Hypermedia as the Engine of Application State

HTTP – hypertext transfer protocol.

JSON – JavaScript Object Notation.

MVC – Model-View-Controller.

SASS - syntactically awesome style sheets.

SLQ – structured query language.

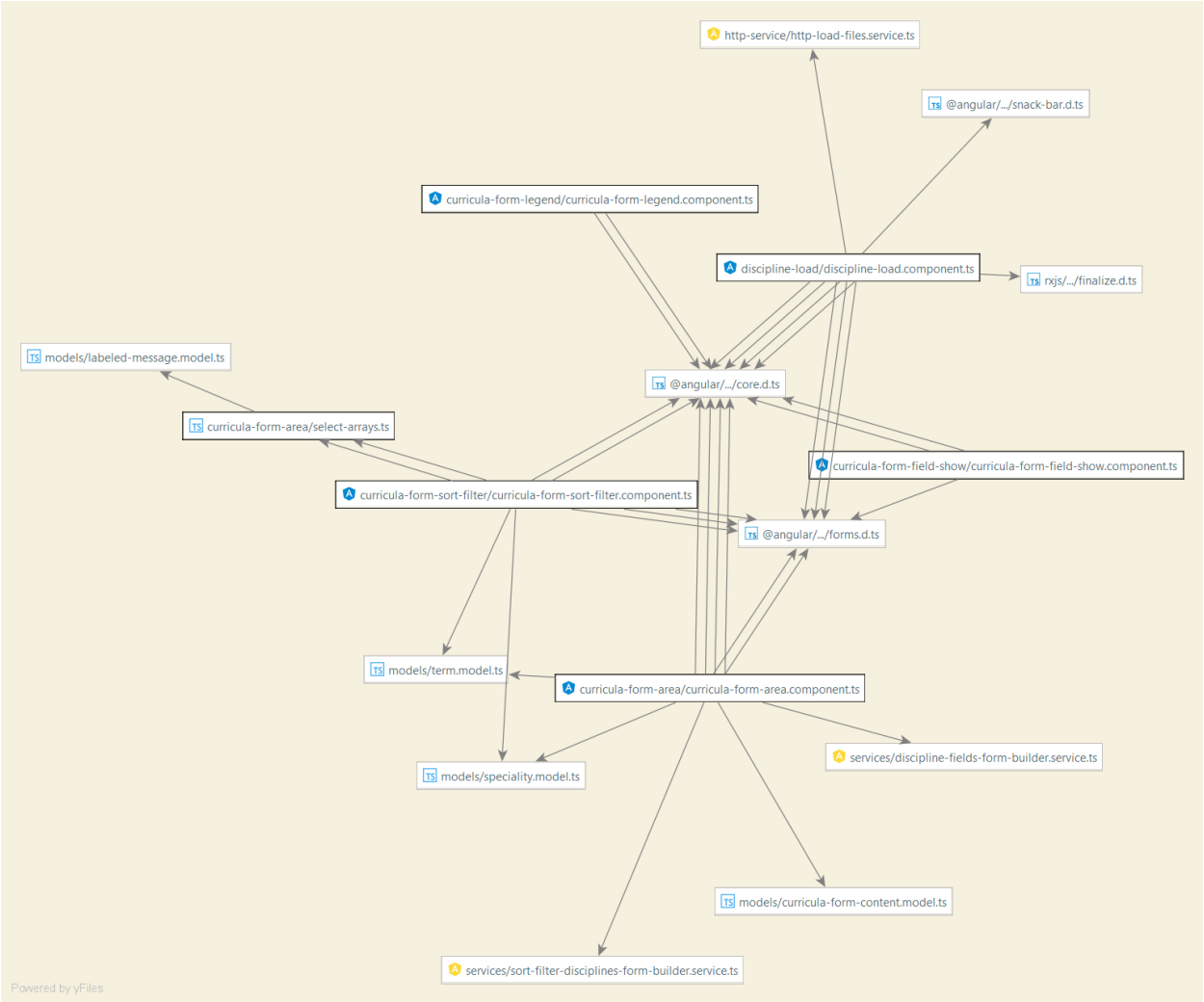
SOLID – акронім для принципів дизайну: Single responsibility, Open–closed, Liskov substitution, Interface segregation, Dependency inversion.

Список використаної літератури

1. Roy Fielding. Architectural Styles and the Design of Network-based Software Architecture. [Електронний ресурс] - https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.
2. Mark Massé (2012) REST API Design Rulebook, 1005 Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc. [Електронний ресурс].
3. Harihara Subramanian, Pethuru Raj (2019) Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs, 35 Livery Street, Birmingham: Pakt Publishing Ltd.[Електронний ресурс].
4. History of Spring and the Spring Framework [Електронний ресурс] - https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html#overview-history.
5. Introduction to Spring Framework. Modules. [Електронний ресурс] - https://docs.spring.io/spring/docs/3.0.0.M4/reference/html/ch01s02.html
6. Spring's RequestBody and ResponseBody Annotation. [Електронний ресурс] - https://www.baeldung.com/spring-request-response-body.
7. Annotation Type RequestParam. [Електронний ресурс] - https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestParam.html.
8. Основи Sass. [Електронний ресурс] - https://sass-scss.ru/guide/.
9. Wiring in Spring: @Autowired, @Resource and @Inject. [Електронний ресурс] - https://www.baeldung.com/spring-annotations-resource-inject-autowire.

Додаток А

Діаграма компоненту curricula-form-area та його залежностей



Додаток Б

Діаграма класів пакету http-service