НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
ім. Ігоря Сікорського

Навчально-науковий комплекс «Інститут прикладного системного аналізу»
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
“___”_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 Комп’ютерні науки
(код і назва)

на тему: Автоматизована система «Документообіг кафедри». Серверна частина

Виконала: студентка IV курсу, групи ДА-32
(шифр групи)

Никитюк Олена Олексіївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Безносик О.Ю. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль старший викладач Бритов О.А. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2017 року

Національний технічний університет України
«Київський політехнічний інститут»
ім. Ігоря Сікорського

Інститут (факультет) ННК «Інститут прикладного системного аналізу
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050101 Комп'ютерні науки
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на дипломну роботу студенту
Никитюк Олені Олексіївні
(прізвище, ім'я, по батькові)

1. Тема роботи: Автоматизована система «Документообіг кафедри». Серверна частина,
керівник роботи _____Безносик Олександр Юрійович, к.т.н_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи:
   1. Навчальний план, робочий навчальний план, форма К-3.
   2. Мова програмування Java, технології Java EE, веб-сервер Glassfish.
   3. Технології Apache POI, Hibernate Framework, Google Gson.
   4. Середовище програмування NetBeans IDE.

4. Зміст роботи
   1. Аналіз документообігу кафедри
   2. Проектування архітектури серверної частини системи
   3. Реалізація серверної частини системи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____

_____

_____

_____

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання		
2	Аналіз вимог технічного завдання		
3	Підготовка теоретичної основи роботи		
4	Аналіз технологій для вирішення задачі		
5	Розробка архітектури серверної частини		
6	Реалізація серверної частини		
7	Тестування системи та виправлення помилок		
8	Оформлення дипломної роботи		
9	Отримання допуску до захисту та подання роботи в ДЕК		

Студентка _____ Никитюк О.О.
(підпис) (ініціали, прізвище)

Керівник роботи _____ Безносик О.Ю.
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломної роботи.

АНОТАЦІЯ

до бакалаврської дипломної роботи Никитюк Олени Олексіївни
на тему «Автоматизована система «Документообіг кафедри». Серверна частина»

Робота присвячена розробці серверної частини автоматизованої системи «Документообіг кафедри». Система вирішує задачу автоматичного генерування документації на основі інформації, представленої у базі даних. В роботі викладено аналіз навчальної документації НТУУ «КПІ», на основі якого було складено алгоритм автоматичної генерації, описано технології та бібліотеки, використані для вирішення поставленої задачі. Робота містить опис принципів побудови архітектури серверної частини системи. В ході перевірки працездатності системи було отримано згенеровані навчальні плани, робочі навчальні плани та форму К-3.

В результаті виконання роботи була реалізована можливість автоматизації створення навчальної документації та взаємодії серверної частини системи з клієнтською на основі HTTP-запитів та обміну об’єктів у форматі JSON.

Розроблене програмне забезпечення разом з базою даних та клієнтською частиною може бути використано для генерування навчальної документації на кафедрах НТУУ «КПІ».

Загальний обсяг роботи 79 сторінок, 12 рисунків, 6 таблиць, 10 бібліографічних найменувань.

Ключові слова: навчальна документація, навчальний план, робочий навчальний план, форма К-3, веб-додаток, HTTP-запит, JSON.

АННОТАЦИЯ

к бакалаврской дипломной работе Никитюк Елены Алексеевны
на тему «Автоматизированная система «Документооборот кафедры».
Серверная часть»

Работа посвящена разработке серверной части автоматизированной системы «Документооборот кафедры». Система решает задачу автоматического генерирования документации на основе информации, представленной в базе данных. В работе изложено анализ учебной документации НТУУ «КПИ», на основе которого было составлено алгоритм автоматической генерации, описано технологии и библиотеки, использованные для решения поставленной задачи. Работа содержит описание принципов построения архитектуры серверной части системы. В ходе проверки работоспособности системы было получено учебный план, рабочий учебный план и форму К-3.

В результате выполнения работы было реализована возможность автоматизации создания учебной документации и взаимодействия серверной части системы с клиентской на основе HTTP-запросов и обмена объектов в формате JSON.

Разработанное программное обеспечение вместе с базой данных и клиентской частью может быть использовано для генерирования учебной документации на кафедрах НТУУ «КПИ».

Общий объем работы 79 страниц, 12 рисунков, 6 таблиц, 10 библиографических наименований.

Ключевые слова: учебная документация, учебный план, рабочий учебный план, форма К-3, веб-приложние, HTTP-запрос, JSON.

ANNOTATION

to the bachelor’s degree work of Nykytiuk Olena Oleksiivna
on the topic «Automated system of department's document workflow. Server side»

This work is dedicated to the development of the server side of the automated system of department's document workflow. The system solves the problem of automatic generation of documentation based on information presented in the database. The work outlines the analysis of the educational documentation of NTUU “KPI”, on the basis of which an algorithm for automatic generation was compiled, describes the technologies and libraries used to solve the task. The work contains a description of the principles of building the architecture of the server side of the system. During the testing of the system's efficiency, a curriculum, a work plan and a K-3 form were obtained.

As a result of the work, the ability to automate the creation of educational documentation and the interaction between the server side of the system and the client side based on HTTP-requests and exchange of objects in JSON format were implemented.

The developed software along with the database and the client side can be used to generate educational documentation at NTUU “KPI” departments.

Total of 79 pages, 12 figures, 6 tables, 10 bibliographic items.

Keywords: educational documentation, curriculum, work plan, K-3 form, web application, HTTP request, JSON.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД	–	база даних.
СУБД	–	система управління базами даних.
ВНЗ	–	Вищий Навчальний Заклад.
ЕОМ	–	електронна обчислювальна машина.
ДСТУ	–	Державний Стандарт України.
МОН	–	Міністерство освіти і науки України
МКР	–	модульна контрольна робота.
РГР	–	розрахунково-графічна робота.
РР	–	розрахункова робота.
ГР	–	графічна робота.
ДКР	–	домашня контрольна робота.
ОКР	–	освітньо-кваліфікаційний рівень.
ОКХ	–	освітньо-кваліфікаційні характеристики.
ОПП	–	освітньо-професійні програми.
КР	–	курсова робота.
КП	–	курсовий проект.
СРС	–	самостійна робота студентів.
ДЕК	–	державна екзаменаційна комісія.
НТУУ «КПІ»	–	Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».
ECTS	–	European Credit Transfer and Accumulation System (Європейська система переведення і накопичення кредитів).

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).

URL – Uniform Resource Locator (єдиний вказівник на ресурс).

JSON – JavaScript Object Notation (об’єктний запис JavaScript).

XML – eXtensible Markup Language (розширювана мова розмітки).

JAXB – Java Architecture for XML Binding (архітектура Java для зв’язування з XML).

ORM – Object-Relational Mapping (об’єктно-реляційна проекція).

DAO – Data Access Object (об’єкт доступу до даних).

CRUD – Create Read Update Delete (операції створення, читання, оновлення та видалення).

API – Application Programming Interface (програмний інтерфейс додатку).

ВСТУП

Організація роботи ВНЗ потребує великої кількості навчальної документації, яка пов’язана із забезпеченням навчального процесу. Своєчасне і безпомилкове складання цієї документації здатне суттєво підвищити ефективність роботи кафедри та уникнути проблем, які можуть виникнути через помилки та неточності у документах.

Нормативні документи містять інформацію про результати роботи кафедри ВНЗ, пов’язані з плануванням, організацією і контролем навчального процесу та складанням статистичної звітності. Навчальна документація включає в себе навчальні плани, робочі навчальні плани, форму К-3, розподіл навантаження на викладачів та інші.

Більшість навчальної документації базується на інформації з робочих навчальних планів поточного року, що дозволяє організувати автоматизацію документообігу. Система автоматичного генерування документації дозволяє істотно зменшити кількість часу, необхідного для створення нормативних документів, а також уникнути механічних помилок, пов’язаних з людським фактором.

Алгоритм автоматичного створення документації може бути побудований у результаті аналізу структури та вимог до змісту нормативних документів. Найкращим варіантом формалізації цього алгоритму є такий, що дозволяє максимальну варіативність та уникає жорсткої прив’язки до формату документів.

Отже, при розробці системи необхідно враховувати високу ймовірність появи змін у структурі нормативних документів у майбутньому та реалізувати можливість адаптації системи до цих змін для її коректної роботи. При цьому незначні зміни у вимогах до побудови та змісту документації не повинні потребувати внесення змін до самого програмного коду.

Враховуючи структурованість та об’єм інформації для складання документації, для автоматизації процесу документообігу доцільно реалізувати систему, яка взаємодіє з БД та генерує нормативні документи на основі інформації з бази. Найбільш оптимальним варіантом реалізації такої системи є веб-сервіс, архітектура якого розділена на три основні частини: підсистему взаємодії з БД, підсистему бізнес-логіки та підсистему інтерфейсу користувача.

Підсистема взаємодії з базою включає механізм перегляду, створення, зміни та видалення записів БД через веб-сервіс. Підсистема бізнес-логіки містить реалізацію автоматичного генерування документації, а також механізм взаємодії з клієнтською частиною основі HTTP-запитів. Інтерфейс користувача міститься у клієнтській частини, яка звертається до серверної частини через запити.

Підхід до проектування архітектури веб-сервісів, заснований на розділенні на три окремих модулі, є одним з найпоширеніших загальноприйнятих принципів побудови архітектури програмного забезпечення, який використовується у багатьох комерційних та некомерційних проектах. Слабка зв’язність між компонентами та чіткий розподіл відповідальності між окремими модулями значно полегшує розробку і дозволяє мінімізувати необхідну для реалізації системи кількість програмного коду.

Розроблена серверна частина системи автоматизованого документообігу разом з БД та клієнтською частиною складає єдину цілісну систему, основною метою якою є полегшення та прискорення створення нормативних документів.

Реалізація системи побудована за принципом максимальної незалежності структурних частин одне від одної і завдяки цьому дозволяє забезпечити максимальну гнучкість системи та придатність для підтримки і подальшого розширення та вдосконалення.

Отже, темою даної бакалаврської дипломної роботи є розробка серверної частини автоматизованої системи «Документообіг кафедри», необхідної для автоматизації створення нормативних документів кафедри та взаємодії із клієнтською частиною системи.

1 АНАЛІЗ ДОКУМЕНТООБІГУ КАФЕДРИ

1.1 Вступ

Організація роботи кафедри складається з великої кількості направлень, кожне з яких відображується у відповідних нормативних документах. Основними видами навчальної документації є навчальний план, робочий навчальний план, форма К-3, розподіл навантаження на викладачів та інші. Більшість навчальної документації базується на інформації, представленої у робочих навчальних планах, що дозволяє спроектувати систему, здатну автоматизувати процес створення документації.

Автоматизація створення нормативних документів дозволить зекономити час, що витрачається на підготовку документообігу, та прискорити і полегшити процес внесення змін до вже існуючих документів.

Для організації автоматизації процесу документообігу пропонується створення єдиної системи документообігу кафедри, яка складається з БД, веб-сервісу, який взаємодіє з базою та виконує генерування нормативних документів на основі інформації з БД, і клієнтської частини, яка взаємодіє із сервером через HTTP-запити і користувачем через користувацький інтерфейс.

Основними задачами системи документообігу є створення навчальних планів, робочих навчальних планів, форми К-3, розподілу навантаження на викладачів та інших нормативних документів на основі отриманої від користувача інформації, яка зберігається у БД.

Розробка системи починається з аналізу вихідних даних, які включають готові нормативні документи, для визначення основних принципів побудови документів та алгоритму їх автоматичної генерації. Після аналізу вихідних даних необхідно спроектувати гнучку архітектуру системи, яка реалізує необхідний функціонал, а також придатна для розширення та удосконалення у майбутньому.

Максимальна гнучкість системи забезпечується шляхом проектування на основі принципів створення архітектури веб-сервісів із розділенням системи на максимально незалежні одне від одної підсистеми. Такий підхід призводить до того, що внесення змін в одну частину систему мінімально впливає на інші, що дозволяє уникнути потенційних помилок та труднощів, пов’язаних з підтримкою, розширенням та удосконаленням системи.

1.2 Аналіз навчального плану

Основним нормативним документом, необхідним для здійснення організації навчального процесу, є навчальний план, який містить розподіл кредитів ECTS між дисциплінами, обсяг лекційний, лабораторних та практичних занять у годинах для кожної дисципліни, форми проведення поточного та підсумкового контролю із зазначенням семестру для кожної форми контролю, а також розподіл аудиторних годин на тиждень за семестрами [1].

Функція навчального плану полягає у відображенні планування навчального процесу для усіх ОПП кафедри. Для кожної ОПП складається окремий навчальний план. ОПП включає в себе ОКР (бакалавр, спеціаліст, магістр і доктор філософії), форму навчання (денна, заочна та очна) і спеціалізацію.

Навчальний план для кожної ОПП містить перелік усіх дисциплін (обов’язкових та вибіркових, що включають в себе дисципліни за вибором ВНЗ та дисципліни за вибором студентів), які вивчаються у межах цієї ОПП. Отже, основною одиницею інформації навчального плану є навчальна дисципліна та її властивості.

Навчальні предмети у навчальному плані представлені у вигляді таблиці, рядками якої є окремі дисципліни, а стовпцями – атрибути, на основі значень яких можна отримати комплексну інформацію про план вивчення кожної дисциплін навчального плану. Окрім цього, навчальний план містить і сумарні значення атрибутів по всіх дисциплінах ОПП.

Для кожної дисципліни начального плану заповнюються наступні відомості:

- шифр;
- повна назва;
- перелік семестрів, у яких проводиться екзамен;
- перелік семестрів, у яких проводиться залік, в тому числі враховуються і диференційовані заліки (для їх позначення до номеру семестру додається літера «д»);
- перелік семестрів, у яких виконується курсовий проект;
- перелік семестрів, у яких виконується курсова робота;
- кількість кредитів ECTS;
- загальний обсяг у годинах, який визначається як кількість кредитів, помножена на обсяг одного кредиту (30 годин);
- загальний обсяг лекційних занять у годинах;
- загальний обсяг практичних занять у годинах;
- загальний обсяг лабораторних занять у годинах;
- сумарний обсяг лекційних, практичних та лабораторних занять у годинах;
- обсяг самостійної роботи у годинах, який визначається як різниця загального обсягу дисципліни у годинах та сумарного обсягу лекційних, практичних та лабораторних занять;
- розподіл аудиторних годин на тиждень за курсами і семестрами, при цьому для кожного семестру зазначається кількість аудиторних годин на тиждень, яка визначається як сумарний обсяг аудиторних годин (лекційних, практичних та лабораторних занять), розділений на кількість тижнів у семестрі, яка зазначена для кожного семестру окремо і в загальному випадку не є однаковою для всіх семестрів та для однакових семестрів різних ОПП.

Перелік дисциплін у навчальному плані розділений на цикли (наприклад, цикл загальної підготовки, цикл професійної підготовки тощо) , які в свою чергу поділяються на направлення (наприклад, навчальні дисципліни природничо-наукової підготовки, навчальні дисципліни базової підготовки). Направлення можуть бути обов'язковими або за вибором студентів.

Для кожного направлення після списку дисциплін, що до нього входять, зазначаються усі вищеперераховані дані про предмети (окрім шифру та назви), значення яких визначається як сума по усім предметам направлення (сумарна кількість екзаменів по усіх дисциплінах направлення, обсяг лекційних занять у годинах тощо). Аналогічно ці властивості зазначаються для кожного циклу підготовки та для усього начального плану.

У кінці документу для кожного семестру зазначається сумарна кількість годин на тиждень, екзаменів, заліків (з урахуванням диференційованих заліків), курсових проектів та курсових робіт, які виконуються у цьому семестрі.

Окрім списку дисциплін у навчальному плані також міститься інформація про форми практики та атестації випускників для відповідної ОПП. Для практики зазначається її назва, семестр та кількість тижнів, для атестації випускників – назва, форма державної атестації та семестр.

1.3 Аналіз робочого навчального плану

Робочий навчальний план має структуру, подібну до навчального плану, але є більш деталізованим. Робочі навчальні плани складаються на основі відповідних їм навчальних планів, при цьому окремі робочі навчальні плани складаються для різних років набору студентів кожної ОПП. Отже, робочий навчальний план отримується шляхом розділення відповідного навчального плану на курси. При цьому, якщо у навчальному плані можуть міститись дисципліни, які вивчаються протягом декількох семестрів та мають підсумковий контроль після кожного семестру вивчення, то у робочому навчальному плану такі дисципліни зазначаються окремими рядками для кожного семестру [2].

Основні відмінності робочого навчального плану від навчального плану полягають у наступному:

- для кожної дисципліни зазначається назва кафедри, яка викладає цю дисципліну;
- окрім екзаменів, заліків, курсових проектів та робіт представлена інформація про такі форми контролю: МКР, РГР/РР/РГ, ДКР, реферати.
- у розподілі аудиторних занять на тиждень за годинами зазначені не усі семестри відповідної ОПП, а лише ті, що охоплюються поточним робочим навчальним планом;
- у розподілі аудиторних занять на тиждень окрім сумарної кількості аудиторних занять на тиждень зазначається окремо обсяг лекційних, практичних та лабораторних аудиторних занять на тиждень у годинах;
- для практики та державної атестації випускників додатково міститься інформація про термін проведення, який включає дату початку практики (державної атестації) та дату закінчення практики (державної атестації).

Окрім вищеперерахованих відмінностей у структурі нормативного документа, робочий навчальний план містить таблицю про розподіл годин по підготовці та захисту дипломного проекту та розподіл годин з комплексного державного екзамену. Рядки цих таблиць складаються з наступних відомостей;

- вид роботи (керівництво, консультування тощо);
- норма в годинах на одного студента;
- назва кафедри, яка проводить зазначений вид роботи;
- кількість студентів (окремо зазначаються студенти бюджетної на контрактної форми навчання);
- загальна кількість годин, яка визначається як норма в годинах на одного студента, помножена на кількість студентів відповідної форми навчання (обчислюється окремо для бюджетної та контрактної форми).

1.4 Аналіз форми К-3

Форма К-3 містить розрахунок обсягу навчального навантаження кафедри відповідно робочим навчальним планам поточного року. Форма К-3 складається окремо для наступних форм навчання: денна бюджетна, денна контрактна, заочна бюджетна, заочна контрактна.

Для кожної форми навчання окремо складається два аркуші з різними типами навантаження:

- викладання дисциплін кафедри;
- інші види навчальної роботи.

Окрім цього, форма містить аркуші із сукупними відомостями про навантаження кафедри, а саме інформацію по денній формі навчання, заочній формі навчання, загальну інформацію (значення усіх полів обчислюються як сума значень аналогічних властивостей для денної та заочної форм навчання), а також окремо інформацію по інших видах навчальної роботи.

Кожен аркуш із сукупними відомостями, в свою чергу, розділений на три частини: дані про бюджетну і контрактну форми навчання окремо та сумарні значення, які обчислюються як сума відповідних відомостей бюджетної та контрактної частини.

Аркуші із сукупними відомостями не містять унікальної інформації і повністю складаються на основі даних з інших аркушів форми К-3. Отже, ці частини документа не потребують автоматичної генерації. Замість цього доцільно прописати у шаблоні документу Excel формули, що міститимуть посилання на аркуші форми з деталізованою інформацією. Тоді заповнення таких аркушів значеннями відбуватиметься автоматично за допомогою засобів Microsoft Excel після генерування частин форми К-3 з деталізованою інформацією.

Отже, аркуші із сукупними даними форми К-3 не повинні генеруватись системою і можуть бути виключені з подальшого розгляду та аналізу.

1.4.1 Частина І форми К-3

Перша частина форми К-3 містить інформацію про викладання дисциплін кафедри. Список дисциплін складається на основі робочих навчальних планів поточного року. Для кожної дисципліни зазначається наступна інформація:

- факультет, який забезпечується, назва дисципліни, її загальний обсяг в годинах, курс навчання, шифр груп, кількість студентів в кожній групі (окремо вноситься інформація про студентів бюджетної та контрактної форми навчання);
- обсяг дисципліни за семестр, при чому якщо формою підсумкового контролю дисципліни є екзамен, від загального обсягу віднімається один кредит ECTS (30 годин);
- обсяг аудиторних занять у годинах: лекційних, практичних, лабораторних та індивідуальних;
- кількість контрольних заходів: екзаменів, заліків, МКР, курсових проектів, курсових робіт, РГР/РР/ГР, ДКР, рефератів; при цьому для курсових проектів і курсових робіт зазначається кількість годин на їх виконання;
- кількість бюджетних і контрактних груп та підгруп для практичних і лабораторних занять (у цьому випадку група вважається бюджетною, якщо в ній навчається хоча б один студент-бюджетник);
- кількість студентів бюджетної та контрактної форми навчання у бюджетних та контрактних групах (зазначається у чотирьох різних колонках);
- кількість бюджетних (якщо поточна форма К-3 відноситься до бюджетної форми навчання) або контрактних (якщо поточна форма К-3 відноситься до контрактної форми навчання) потоків;
- розрахунок навчального навантаження за попередньо визначеними формулами.

Розрахунок підгруп для проведення практичних занять здійснюється з урахуванням максимальної кількості студентів у підгрупі (35). Якщо потік складається з декількох груп, сумарна кількість студентів у яких не перевищує максимальну кількість студентів у підгрупі, то практичні заняття для цих груп проводяться разом.

Аналогічно здійснюється підрахунок підгруп для проведення лабораторних занять (максимальна кількість студентів у підгрупі – 23). Якщо кількість студентів у групі менша за 12, то для розрахунку кількості підгруп така група об’єднується з іншою групою того ж потоку.

Навчальні дисципліни з робочих планів для різних ОПП записуються у формі К-3 в окремі рядки, але у тому випадку, коли кількість студентів є невеликою, для розрахунку підгруп для проведення практичних та лабораторних занять групи різних ОПП об’єднуються.

Розрахунок навчального навантаження дисципліни відбується на основі описаних вище даних за чітко визначеними формулами. Система автоматичної генерації працює таким чином, що після заповнення excel-файлу з формою К-3 навчальне навантаження розраховується у цьому файлі, після чого інформація переноситься з файлу у базу даних і використовується у подальшому для обчислення навчального навантаження на викладачів.

1.4.2 Частина ІІ форми К-3

Друга частина форми К-3 містить інформацію про інші види навчальної роботи кафедри, які включають індивідуальні заняття по дисципліні «Наукові дослідження», керівництво практикою (по кожному виду практики окремо), керівництво атестаційними роботами (окремо для кожного ОКР), консультування атестаційних робіт, рецензування атестаційних робіт, консультування перед державним екзаменом, роботу в ДЕК (захист дипломних робіт, усні та письмові екзамени для різних ОКР), керівництво аспірантами, заняття з аспірантами та консультування докторантів.

Рядки цього нормативного документу містять наступні відомості для кожного виду роботи:

- назва виду роботи;
- норма в годинах;
- назва факультету (інституту);
- курс;
- шифр груп;
- кількість студентів бюджетної або контрактної форми навчання в залежності від форми навчання поточної форми К-З;
- сумарна кількість годин, яка розраховується як добуток норми в годинах на одного студента на кількість студентів.

Інформація по кожному виду роботи зазначається окремо для першого і другого семестру.

1.5 Висновки

У цьому розділі було виконано аналіз нормативних навчальних документів НТТУ «КПІ», описано їх структуру та правила формування. На основі викладеної інформації можна зробити висновок, що описані документи мають чіткий алгоритм формування, що дозволяє автоматизувати їх складання і створити систему для автоматичної генерації документації.

На основі аналізу нормативних документів можна побудувати та реалізувати алгоритм їх генерації з інформації у БД. Враховуючи подібність навчального плану та робочого навчального плану, їх автоматичну генерацію можна спроектувати за принципом повторного використання коду, що значно зменшить кількість вихідного коду у системі, а також спростить подальшу підтримку, розширення та удосконалення системи.

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1 Вступ

Найважливішою частиною у розробці будь-якої інформаційної системи є проектування архітектури, адже правильно спроектована гнучка архітектура спрощує майбутню підтримку, розширення та удосконалення системи. Помилки, здійснені при проектуванні та виявлені на подальших етапах, потребують великих затрат для виправлення.

Саме тому вибору інструментів, мов програмування, технологій та бібліотек потрібно приділити особливу увагу. Необхідно обирати технології для розробки, які максимально задовольняють вимоги функціоналу системи, адже широкі стандартні можливості бібліотек та технологій дозволяють розробнику уникнути написання частини програмного коду через використання вже готових рішень. Широкі можливості обраних бібліотек та технологій також полегшують модернізацію та внесення змін до системи.

Вибір технологій для реалізації системи автоматизованого документообігу потрібно здійснювати з врахуванням наступних вимог:

- система повинна мати клієнт-серверну архітектуру, тому обрана мова програмування повинна максимально спростити обмін запитами між сервером та клієнтом;
- система працює з великою БД, тому потрібно зробити механізм взаємодії з БД максимально гнучким та прозорим, що дозволить зменшити кількість програмного коду та полегшить підтримку системи у майбутньому;
- система працює з файлами формату .xls, тому потрібно обирати бібліотеку, яка дозволяє максимально полегшити запис та читання таких файлів, а також має широкий функціонал для здійснення інших маніпуляцій з excel-файлами.

Іншою важливою частиною етапу проектування є визначення структурних частин архітектури системи. Максимальна незалежність частин системи забезпечує значне спрощення при виправленні помилок у системі та внесенні змін до її вихідного коду.

Для побудови правильної архітектури, яка дозволяє реалізувати весь необхідний функціонал, необхідно спочатку чітко визначити перелік функціональних вимог до системи.

2.2 Функціональні вимоги до системи

Для вибору технологій, які будуть використовуватись при розробці системи, та проектування архітектури серверної частини необхідно спочатку визначити функціональні вимоги.

Готова серверна частина системи автоматизованого документообігу повинна здійснювати наступні функції:

- вибірка даних з БД та внесення змін до неї;
- автоматизоване створення нормативних документів на основі інформації з БД;
- коректний запис згенерованої документації у файли формату .xls шляхом заповнення попередньо підготовленого шаблону відповідними позначками;
- переключення між БД різних років;
- створення БД для нових років;
- взаємодія з JavaScript-клієнтами за допомогою обмінів запит-відповідь;
- коректна обробка запитів від клієнтів: вибірка даних з БД та внесення змін до неї за запитом клієнта, відправлення готової згенерованої документації або шаблонів клієнту у вигляді файлів формату .xls, оновлення шаблонів на сервері за відповідним запитом;

2.3 Вибір технологій та бібліотек

2.3.1 Вибір мови програмування

Система автоматизованого документообігу повинна мати клієнт-серверну архітектуру, отже при виборі мови програмування необхідно керуватись можливостями та технологіями для створення веб-додатків.

Мова програмування Java з самого початку розроблялась в першу чергу для написання веб-додатків, тому її технології надають розробнику широкий спектр можливостей для розробки веб-сервісів, включаючи бібліотеки сторонніх розробників, які використовуються при роботі з Java EE.

Для взаємодії з клієнтською частиною додатку необхідно організувати обмін за принципом запит-відповідь, при цьому запити клієнта і відповіді сервера повинні бути у форматі HTTP-запитів. Такий функціонал реалізований у Java Servlet API. Клас HttpServlet призначений для обробки HTTP-запитів (методи Get, Post, Put та Delete).

Запит клієнта представлений як об’єкт класу HttpServletRequest та має набір методів, які дозволяють отримати заголовки запиту, значення атрибутів та параметрів. Значення заголовків, атрибутів та параметрів можна отримати за їхньою назвою, що значно спрощує роботу з об’єктами у форматі JSON.

Відповідь сервера представлена об’єктом класу HttpServletResponse, який також має набір методів, який дозволяє призначати заголовки, атрибути та параметри відповіді сервера, що аналогічно може бути використано для обміну об’єктами між клієнтом і сервером у форматі JSON. Також через відповідь сервера можна переправити користувача на веб-сторінку з певною URL-адресою.

Взаємодія між клієнтом та сервером відбувається наступним чином. Клієнт відправляє HTTP-запит на сервер, який на серверній стороні переформовується в об’єкт класу HttpServletRequest. С цим об’єктом взаємодіють веб-компоненти Java-додатку, отримуючи з нього необхідну інформацію за допомогою стандартних методів.

Веб-компоненти додатку взаємодіють з БД та бізнес-логікою системи. Після обробки запиту клієнта та виклику усіх необхідних для формування відповіді методів сервер генерує відповідь клієнту у вигляді об’єкту класу HttpServletResponse, у який за допомогою стандартних методів класу записуються заголовки, атрибути, параметри та їх значення.

З об’єкту класу HttpServletResponse формується відповідь сервера клієнту у форматі HTTP. Переформування запитів в об’єкти та навпаки здійснюється можливостями технологій мови програмування Java без участі розробника, у програмному коді використовується лише представлення запитів у вигляді об'єктів, які передаються як аргументи методів Get, Post, Put та Delete об’єктів класів, які розширюють клас HttpServlet.

Схема взаємодії клієнтської частини додатку з серверною через HTTP-запити з використанням Java Servlet API представлена на рис. 2.1.

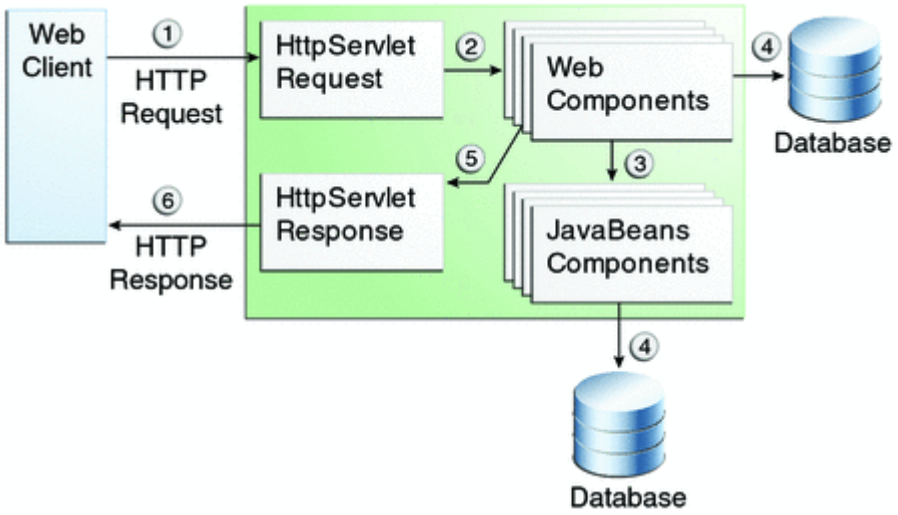

Рисунок 2.1 – Взаємодія клієнту з сервером через HTTP-запити [9]

Важливою особливістю роботи з HTTP-запитами у Java Servlet API є те, що завантаження файлів на сервер або з серверу не вимагає великої кількості коду, а здійснюється через вхідний/вихідний потоки запитів і відповідей. Тобто, можна завантажити файл на сервер прямо з вихідного потоку запиту або відправити файл клієнту, записавши його у вхідний потік відповіді сервера.

Єдиною умовою для правильного розпізнавання файлу як на клієнтській стороні, так і серверній, є коректне призначення заголовків запиту або відповіді, які визначають формат вмісту.

2.3.2 Вибір технології взаємодії з БД

Так як функціонал системи щільно пов’язаний з БД, необхідно спроектувати архітектуру серверної частини таким чином, щоб максимально спростити роботу з БД, а також залишити можливість для подальшого вдосконалення та розширенні системи. У мові програмування Java є два основних способи роботи з БД: безпосередньо за допомогою бібліотеки JDBC (Java DataBase Connectivity) та з використанням JPA (Java Persistence API).

Бібліотека JDBC – прикладний програмний інтерфейс Java, який визначає методи, за допомогою яких Java-додаток здійснює доступ до БД. JDBC – це платформо-незалежний стандарт взаємодії Java-застосунків з різноманітними СУБД, реалізований у вигляді пакета java.sql, що входить до складу Java SE. Бібліотека дозволяє виконувати SQL-запити (вибірку, створення, оновлення чи видалення даних) та процедури безпосередньо з програмного коду. Основним недоліком використання такого підходу є те, що програмний код, навантажений SQL-запитами важко підтримувати, а внесення модифікацій до БД потребує суттєвого рефакторингу вихідного коду, що призводить до великих затрат часу навіть на незначні модифікації та виправлення у структурі БД.

Java Persistence API – стандартизований інтерфейс для Java ORM фреймворків. JPA надає розробнику можливість пов’язувати класи додатку з сутностями БД [8]. Опис зв’язку класів з сутностями, а також властивостей класів із конкретними атрибутами сутностей у базі відбувається або в окремому файлі мовою XML або за допомогою анотацій, які проставляються у програмному коді. За такого підходу розробнику не потрібно використовувати SQL-запити для взаємодії з БД, вибірка та модифікація даних відбувається на рівні класів та об’єктів, які пов’язаними з сутностями БД.

Одним з найбільш розповсюджених фреймворків, які реалізують такий підхід, є Hibernate Framework. На рис. 2.2 зображено схему взаємодії прикладного додатку з БД при використанні фреймворку Hibernate.

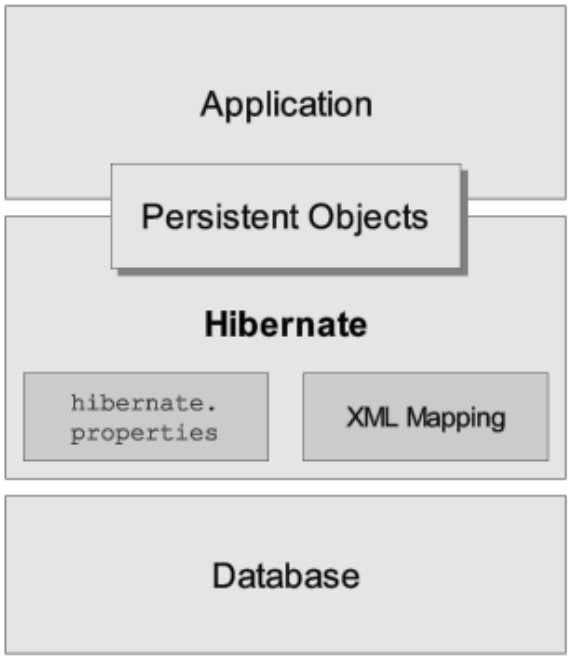

Рисунок 2.2 – Принцип роботи Hibernate Framework [8]

Для роботи з БД у системі використовуватиметься Hibernate Framework. Такий підхід зменшить кількість програмного коду у порівнянні з використанням тільки JDBC та полегшить підтримку системи у майбутньому, так як використання JPA дозволяє писати чистий та зрозумілий код. Для кожної сутності БД буде створено клас, який буде пов’язано із відповідною сутністю за допомогою анотацій JPA.

Вибірка, створення, оновлення та видалення даних відбуватиметься за допомогою об’єкти DAO-класів, яке міститимуть CRUD-методи. Використання узагальнення при створенні DAO-класів дозволить використовувати об’єкти одного і того ж DAO-класу з різними параметрами типу для всіх класів-сутностей, що суттєво полегшує розробку та значно зменшує необхідну кількість програмного коду, дозволяючи використовувати один і той самий код для роботи з усіма класами-сутностями, наявними у системі.

Також завдяки використанню абстракції (проектування DAO як інтерфейсу з конкретною реалізацією HibernateDAO) можна легко змінювати спосіб взаємодії додатку з БД через використання іншої реалізації DAO, при цьому більша частина програмного коду, в якому використовуються взаємодія з БД, не потребує внесення змін.

2.3.3 Вибір технології роботи з файлами

Для роботи з файлами формату .xls при розробці додатків мовою програмування Java найбільш розповсюдженими бібліотеками є ApachePOI, docx4j та JXLS.

Apache POI – проект компанії Apache Software Foundation, який надає функціонал для читання та внесення змін у файли форматів Microsoft Office, такі як Word, PowerPoint та Excel. Одним з компонентів бібліотеки є HSSF (Horrible SpreadSheet Format), призначений для читання та запису файлів формату Microsoft Excel [7].

Бібліотека docx4j – продукт з відкритим вихідним кодом для створення та внесення змін до файлів Microsoft Open XML (Word, PowerPoint, Excel). Бібліотека використовує JAXB для створення представлення об’єктів у форматі XML.

JXLS – бібліотека з відкритим вихідним кодом для створення Excel-файлів при розробці додатків мовою програмування Java. JXLS використовує спеціальну розмітку в документах формату Microsoft Excel для визначення форматування та компоновки вихідних даних.

Для реалізації автоматичного генерування документації було обрано бібліотеку Apache POI. Вибір обґрунтовується тим, що Apache Software Foundation надає розробникам широкий вибір інструментів. Програмне забезпечення під ліцензією Apache активно розвивається і підтримується розробниками, що забезпечує регулярний вихід нових версій бібліотек, які містять новий функціонал та виправлення помилок попередніх версій продуктів.

Бібліотека Apache POI зарекомендувала себе як найбільш надійна, зручна та багатофункціональна у порівнянні з невеликими бібліотеками інших розробників. Широкий функціонал бібліотеки забезпечує можливість розширення та покращення системи автоматизованого документообігу у майбутньому.

2.4 Побудова архітектури додатку

2.4.1 Загальні положення

В загальному випадку правила проектування архітектури веб-сервісів вимагають відокремлення трьох основних компонентів:

- Рівень представлення (Presentation Layer) відповідає за взаємодію з користувачем. У цій частині системи відбувається представлення даних користувачеві у зручній та наочній формі, а також забезпечується механізм роботи користувача з даними через користувацький інтерфейс системи. Для веб-сервісу зазвичай цей компонент представлений у вигляді веб-сторінки, а взаємодія з сервісом відбувається за рахунок запитів JavaScript-клієнту.
- Рівень бізнес-логіки (Business Logic Layer) містить основний функціонал системи. Цей компонент після отримання запиту від клієнту виконує необхідні дії з даними та формує відповідь сервера, яка може містити значення примітивних типів, представлення об’єктів у форматі JSON або XML та бінарні об’єкти (файли різних форматів).
- Рівень доступу до даних (Data Access Layer) необхідний для створення, перегляду, оновлення та видалення даних з бази за допомогою об’єктів-сутностей, які пов’язані із сутностями бази, та об’єктів DAO. Цей компонент безпосередньо взаємодіє з БД та забезпечує ізоляцію рівнів представлення та бізнес-логіки від бази, гарантуючи таким чином те, що до БД не буде внесено змін, які можуть порушити її цілісність, так як усі запити до БД верифікуються шаром бізнес-логіки.

Такий архітектурний підхід забезпечує максимальну незалежність компонентів сервісу одне від одного, що підвищує гнучкість та відмовостійкість системи. При такому розділенні зберігається цілісність даних завдяки тому, що рівень представлення та рівень бізнес-логіки не мають доступу безпосередньо до БД і не можуть самостійно вносити до неї зміни [10]. Також зміни в одному компоненті не впливають на інші підсистеми, що дозволяє уникнути появи помилок у системі при модифікації користувацького інтерфейсу, внесення змін до методів бізнес-логіки, переході на іншу СУБД тощо.

Компоненти системи взаємодіють один з одним через інтерфейси з чітко визначеним переліком методів. На рис. 2.3 зображено схему взаємодії між рівнями веб-сервісу.

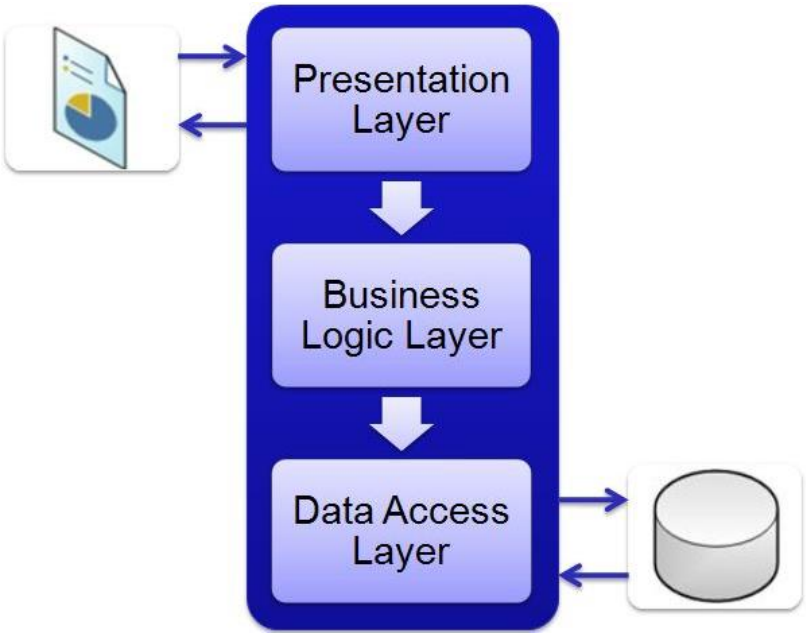

Рисунок 2.3 – Взаємодія між компонентами веб-сервісу [10]

В межах дипломної роботи виконується розробка серверної частини системи, яка включає рівень доступу до даних та рівень бізнес-логіки, причому бізнес-логіка системи включає використання Java Servlet API для реалізації взаємодії з клієнтською частиною системи через обміни HTTP-запитами між клієнтом та сервером.

У даному випадку API сервісу буде використовуватись JavaScript-клієнтом, але реалізація системи виконана таким чином, що не накладає обмежень на представницький рівень, тобто API серверної частини може використовуватись іншим веб-сервісом, мобільним додатком тощо.

2.4.2 Архітектура рівня доступу до даних

Рівень взаємодії з даними у системі включає два основних стандартних компонента: класи-сутності та класи DAO. Класи-сутності пов’язані із сутностями БД за допомогою анотацій Java Persistence API. Об’єкти класів DAO виступають у ролі взаємозв’язку між класами-сутностями та БД, дозволяючи виконувати вибірку з БД та внесення змін до неї через стандартний набір методів (відомих як CRUD-методи) з використанням об’єктів класів-сутностей.

Всі класи-сутності реалізують інтерфейс IDatabaseEntity, який є інтерфейсом-маркером для позначення класів-сутностей, а також включає наступні абстрактні методи, реалізація яких є обов’язковою для всіх класів-сутностей:

- public int getId () – повертає значення ідентифікатора сутності;
- public void setId (int id) – встановлює значення ідентифікатора сутності рівним значенню змінної id.

Для виконання дій над даними бази використовується DAO. У системі реалізовано узагальнене параметризоване DAO, тобто не потрібно виконувати окрему реалізацію для кожного класу-сутності, одна реалізація DAO може бути застосована для всіх класів-сутностей.

Методи для виконання доступних дій над даними бази визначені у інтерфейсі IDAO <T extends IDatabaseEntity>. Тип параметру для узагальнення визначено таким чином, щоб забезпечити роботу усіх реалізацій цього інтерфейсу лише з класами-сутностями, тобто з тими класами, які реалізують інтерфейс IDatabaseEntity і, відповідно, є відображеннями сутностей БД у системі.

В інтерфейсі IDAO визначені наступні методи, які дозволяють виконувати маніпуляції над даними з БД та повинні бути перевизначені класами, які реалізують інтерфейс IDAO:

- public List<T> getAll () – повертає список, що містить об’єкти, які відповідають усім записам пов’язаної таблиці;
- public T get (int id) – повертає об’єкт, який відповідає запису пов’язаної таблиці з ідентифікатором, рівним значенню змінної id;
- public T update (T instance) – оновлює запис пов’язаної таблиці, який відповідає об’єкту instance (тобто, має такий самий ідентифікатор) та повертає об’єкт, який відповідає цьому запису;
- public boolean create (T instance) – створює у пов’язаній таблиці запис, який відповідає об’єкту instance; повертає значення true у разі успішного створення запису, інакше – false;
- public boolean delete (int id) – видаляє з пов’язаної таблиці запис з ідентифікатором, рівним значенню змінної id; повертає значення true у разі успішного видалення запису, інакше – false;

У системі використовується реалізація HibernateDAO інтерфейсу IDAO, яка використовує функціонал Hibernate Framework для визначення вищевказаних методів.

У компоненті бізнес-логіки доступ до методів DAO відбувається через інтерфейс IDAO, тобто у разі використання іншої реалізації цього інтерфейсу необхідно лише замінити конкретну реалізацію при створенні об’єктів типу IDAO. При цьому всі частини коду, які використовують методи DAO, не потребують змін.

На рис. 2.4 зображено узагальнену діаграму ієрархії класів та інтерфейсів для рівня доступу до даних. З діаграми видно, що інтерфейс IDAO реалізується класом HibernateDAO. Інтерфейс IDAO та його реалізації використовують об’єкти класів, що реалізують інтерфейс IDatabaseEntity, тобто об’єкти класів-сутностей.

На діаграмі для спрощення представлено клас EntityClass, який реалізує інтерфейс IDatabaseEntity та замінює в зображеній ієрархії класи-сутності. В програмному коді інтерфейс IDatabaseEntity реалізується конкретними класами-сутностями, кожен з яких має перелік своїх властивостей (полів) та методів.

Кількість та зміст класів-сутностей визначається сутностями БД, їх атрибутами, форматом даних атрибутів, обмеженнями (первинними та вторинними ключами). Зв’язок між класами сутностями, який відповідає зв’язку таблиць БД, встановлюється за допомогою анотацій JPA, проставлених до відповідних членів класу. Заміть значень вторинних ключів у класах-сутностях використовується безпосередньо посилання на сутність, на яку вказує відповідний вторинний ключ.

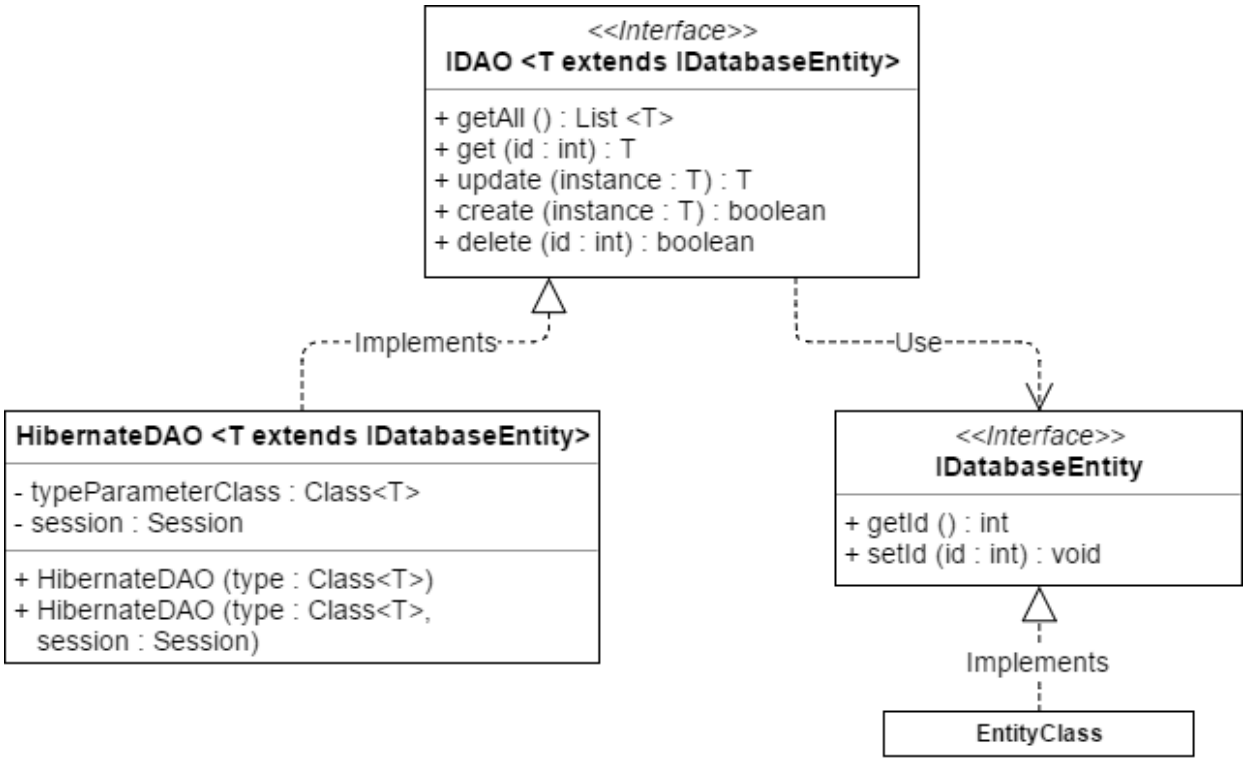

Рисунок 2.4 – Спрощена ієрархія класів рівня доступу до даних

Отже, запропонована архітектура рівня доступу до даних дозволяє уникнути появи SQL-запитів у програмному коді та реалізувати взаємодію з БД виключно на рівні класів і об’єктів з використанням JPA та Hibernate Framework.

2.4.3 Архітектура рівня бізнес-логіки

Бізнес-логіка веб-сервісу містить основний функціонал системи. У цьому компоненті відбувається обробка запитів клієнта та відповідні до запиту дії з даними через рівень доступу до даних, а також генерування документації з інформації, наявної у БД.

Цей компонент системи складається з двох основних частин: API сервера та реалізації функціоналу генерування документації.

API сервера представлене набором URL-посилань, кожне з яких відповідає HTTP-методам класів, які розширюють клас HttpServlet. Кожен метод приймає запит від клієнта, формує та відправляє відповідь. Обмін об’єктами між клієнтом та сервером відбувається у форматі JSON.

Для внесення змін до БД потрібен набір класів, які приймають від клієнта об’єкт у форматі JSON та команду (читання, оновлення, створення, видалення) і виконують необхідні дії через DAO відповідного класу-сутності. Більша частина функціоналу таких класів буде однакова завдяки використанню абстракції, тому можна уникнути написання великої кількості однакового коду через винесення подібного функціоналу в один параметризований абстрактний суперклас AbstractEntityController <T extends IDatabaseEntity>, який розширює клас HttpServlet. Конкретні реалізації цього класу будуть відповідати класам-сутностям з відповідними параметрами. Кожен з цих класів повинен визначати наступні методи:

- protected T getInstance (HttpServletRequest request) – формує об’єкт класу з представлення у форматі JSON, яке записане у запиті клієнта; створений об’єкт може бути використаний для внесення змін до БД;
- protected void getDropDownList (HttpServletResponse response) – повертає масив даних у форматі JSON, які можуть бути використані для створення випадаючого списку; кожен об’єкт масиву містить значення (ідентифікатор запису) та строкове представлення, яке буде відображуватись користувачеві.

Окрім редагування БД сервер повинен надавати користувачеві можливість завантажувати згенеровану документацію, шаблони, а також дозволяти користувачеві завантажити власний шаблон певного нормативного документу на сервер. Для виконання цих дій потрібно створити окремий клас DownloadController, який також розширюватиме клас HttpServlet та міститиме весь вищеописаний функціонал.

На рис. 2.5 зображено спрощену ієрархію класів, які вирішують задачу взаємодії сервера з клієнтом через HTTP-запити. Клас EntityController на діаграмі представляє будь-який клас-контролер, який створено для перегляду та редагування даних таблиці БД, що пов’язана з відповідним класом-сутністю у програмному коді. До класів-контролерів звертається клієнт через їх URL та передає об’єкти у форматі JSON через тіло HTTP-запиту. URL кожного класу-контролеру зазначений у стандартній анотації @WebServlet, яка є частиною Java Servlet API [9].

Рисунок 2.5 – Спрощена ієрархія класів API сервера

Іншим компонентом рівня бізнес-логіки системи є підсистема, яка містить реалізацію функціоналу генерування документації з інформації, наявної у БД. Генерування документації відбувається через вибірку взаємопов’язаних даних з БД за допомогою об’єктів DAO (для кожного класу-сутності використовується окреме DAO з відповідною типізацією параметра T та аргументом конструктора типу Class<T>) за алгоритмом побудови нормативних документів. Усі дані, що використовуються, представлені у форматі об'єктів класів-сутностей. Запис згенерованої документації у файл здійснюється з використанням бібліотеки Apache POI.

Генерування кожного документу виконано в окремому класі. Кожен клас-генератор реалізує інтерфейс IDocumentGenerator з наступники методами:

- public void generate() – виконує формування документу за відповідним алгоритмом та запис результату у файл, шлях до якого передається об’єкту у конструкторі під час створення;
- public Workbook getWorkbook() – повертає книгу Excel, у яку було записано згенерований нормативний документ.

Реалізаціями інтерфейсу IDocumentGenerator є класи CurriculumGenerator, WorkplanGenerator та FormK3Generator. Враховуючи подібність алгоритму автоматичного генерування навчального плану та робочого навчального плану, клас WorkplanGenerator реалізує інтерфейс IDocumentGenerator, розширюючи клас CurriculumGenerator, який реалізує цей інтерфейс.

Використання абстракції дозволяє звертатись до класів-генераторів через їх загальний інтерфейс та викликати методи інтерфейсу у тих частинах програмного коду, які використовують генератори. Потрібна реалізація методу generate() визначається за допомогою поліморфізму під час виконання.

В процесі генерування кожного нормативного документу відбувається прохід по всіх аркушах файлу, вилучення з них відомостей, необхідних для генерування, автоматичне створення документів за отриманими відомостями та запис результату у файл.

Для виконання запису даних у файл реалізації інтерфейсу використовують класи, які розширюють клас AbstractDocumentElement, що представляють різні елементи документу: таблиці, колонки, окремі секції тощо. Аналогічно попередньому випадку, застосування принципу абстракції дозволяє уніфікувати звернення до різних частин документу через суперкласи.

Узагальнену ієрархію класів для генерування нормативних документів представлено на рис. 2.6. Реалізований підхід до проектування архітектури дозволяє додавати нові генератори до системи без необхідності внесення змін до її структури. Для того, щоб дозволити користувачеві роботу з новими генераторами, потрібно лише додати їх у DownloadController.

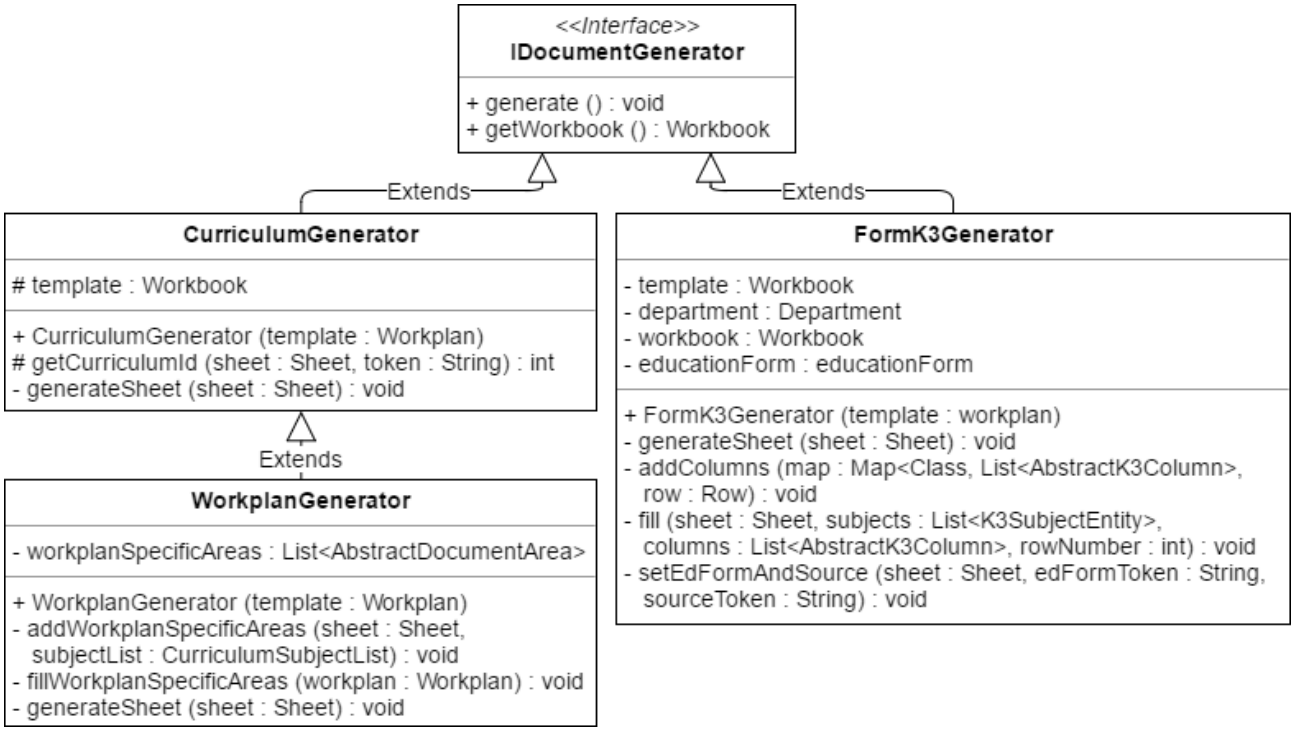

Рисунок 2.6 – Спрощена ієрархія класів для генерування документації

2.5 Висновки

У цьому розділі було виконано аналіз функціональних вимог до серверної частини системи автоматизованого документообігу, визначено загальні властивості її складових компонентів.

Було наведено перелік необхідних технологій та бібліотек для забезпечення максимального спрощення реалізації системи та наведено список найпоширеніших рішень для реалізації потрібного функціоналу. На основі аналізу альтернативних рішень було визначено оптимальні технології та бібліотеки, які можна використовувати для реалізації системи.

Також у розділі було проаналізовано загальні принципи побудови додатків з клієнт-серверною архітектурою, описано призначення компонентів архітектури додатку та виконано розбиття серверної частини системи на основні функціональні модулі. Для кожного функціонального модуля наведено загальні принципи його реалізації та діаграму ієрархії класів та інтерфейсів для найвищого рівня абстракції відповідного програмного модуля.

3 РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

3.1 Вступ

Реалізація функціоналу серверної частини системи виконується на основі описаної у попередньому розділі тришарової архітектури.

У процесі реалізації відбується як написання визначених на етапі проектування архітектури класів та інтерфейсів, так і розширення та доповнення попередньо визначеної архітектури. Якщо на етапі проектування архітектури визначались класи та інтерфейсі, що належать до найвищого рівня абстракції, то етап реалізації передбачає роботу з нижчими рівнями абстракції та конкретними реалізаціями, тобто застосовується низхідний підхід до проектування інформаційних систем.

В процесі опису реалізації, окрім розділення модулів програмної системи на рівень доступу до даних та рівень бізнес-логіки, бізнес-логіка додатку поділяється на підсистему автоматичного генерування документації та підсистему взаємодії з клієнтською частиною додатку через обміни HTTP-запитами.

Отже, реалізація системи включає написання та налагодження трьох основних компонент:

- механізм взаємодії з БД;
- механізм автоматичного генерування документації з наявної у БД інформації;
- механізм взаємодії з клієнтом через обміни HTTP-запитами.

Кожен з елементів декомпозиції системи потребує окремого детального опису. При написанні вихідного коду системи взаємозв’язок між класами додатку вибудовувався таким чином, щоб уникнути тісних зв’язків між компонентами серверної частини системи. Такий підхід підвищує придатність системи до підтримки та розширення у майбутньому.

3.2 Реалізація взаємодії з БД

Рівень взаємодії з БД реалізовано з використанням Java Persistence API та Hibernate Framework. Архітектура цього компонента була описана у попередньому розділі (див. рис. 2.4). Для рівня взаємодії з БД можна виділити два типи класів: класи-сутності та класи DAO.

Класи-сутності пов’язані із сутностями БД за допомогою анотацій Java Persistence API. Основними використаними анотаціями є наступні:

- @Entity – позначення класу як сутності;
- @Table – пов’язує клас із таблицею БД, назва якої зазначена у дужках;
- @Inheritance – використовується для створення ієрархії класів із сутностей БД; у даному випадку використовується для створення ієрархії класів на основі однієї таблиці;
- @DiscriminatorColumn – позначає атрибут таблиці, який визначає, до якого класу з ієрархії відноситься запис;
- @DiscriminatorValue – позначає значення DiscriminatorColumn для поточного класу;
- @Id – позначає член класу як унікальний ідентифікатор сутності;
- @Column – позначає член класу як атрибут відповідної сутності з БД, назва якого зазначена у дужках;
- @ManyToOne – позначає член класу як об’єкт сутності, яка пов’язана з поточною сутністю зв’язком типу many-to-one; при цьому член класу повинен мати тип, який пов’язаний з відповідною таблицею у БД;
- @JoinColumn – визначає, який саме атрибут таблиці вказує на ідентифікатор сутності, яка пов’язана із поточною таблицею зв’язком many-to-one;
- @OneToMany – позначає член класу як об’єкт сутності, яка пов’язана з поточною сутністю зв’язком типу one-to-many; при цьому член класу є колекцією того типу, який пов’язаний з відповідною таблицею у БД.

Класи для відображення сутностей були створені відповідно до сутностей наявної БД. Опис класів-сутностей рівня взаємодії з БД наведено нижче.

AcademicGroup – клас сутності «academic_group». Містить інформацію про академічні групи: шифр, кількість студентів бюджетної і контрактної форм навчання, рік початку навчання, посилання на спеціалізацію, ОКР, форму навчання і робочий навчальний план групи.

Control – клас сутності «control». Містить інформацію про контрольні заходи: семестр та посилання на тип контрольного заходу і навчальний предмет.

ControlDictionary – клас сутності «dict_control». Містить інформацію про типи контролю: назву (екзамен, залік, курсова робота тощо).

Curriculum – клас сутності «curriculum», який відповідає записам у таблиці з позначкою «curriculum». Містить інформацію про навчальні плани: назву та колекції об’єктів класів CurriculumSubject і Workplan, які пов’язані з поточним навчальним планом;

CurriculumSubject – клас сутності «curriculum_subject». Містить посилання на навчальний план (робочий навчальний план) та навчальний предмет. Фактично реалізує зв’язок many-to-many між класом Curriculum та класом Subject, окрім цього містить додаткове поле: шифр навчального предмета у навчальному плані (робочому навчальному плані).

Cycle – клас сутності «cycle». Містить інформацію про розділи навчального плану (робочого навчального плану): назву та колекцію об’єктів класу Section, які пов’язані з поточним розділом.

Department – клас сутності «department». Містить інформацію про кафедри: назву та колекцію об’єктів класу Specialization, які пов’язані з поточною кафедрою.

DiplomaPreparation – клас сутності «diploma_preparation». Містить інформацію про підготовку та захист дипломного проекту: норму в годинах на одного студента, посилання на вид роботи, кафедру і робочий навчальний план.

EducationForm – клас сутності «form_of_education». Містить інформацію про форми навчання: назву (денна, заочна та очна).

Practice – клас сутності «practice». Містить інформацію про проходження практики: назву, семестр, тривалість у тижнях, дату початку та закінчення практики.

Qualification – клас сутності «qualification». Містить інформацію про ОКР: назву (бакалавр, магістр, спеціаліст та доктор філософії).

Section – клас сутності «section». Містить інформацію про підрозділи навчального плану (робочого навчального плану): назву, логічну позначку «за вибором студентів» та посилання на розділ.

Specialization – клас сутності «specialization». Містить інформацію про спеціалізації: назву та посилання на кафедру.

StateCertification – клас сутності «state_certification». Містить інформацію про проведення державної атестації: назву, семестр, дату початку та закінчення атестації.

StudyLoadResults – клас сутності «study_load_results». Містить інформацію про результати розрахунків навчального навантаження у формі К-3 та використовується для розподілу навчального навантаження на викладачів. Члени класу StudyLoadResults, як і атрибути відповідної таблиці, відповідають колонкам правої частини форми К-3 з результатами розрахунків навчального навантаження.

Subject – клас сутності «academic_subject». Містить інформацію про навчальні предмети: семестр початку викладання дисципліни, тривалість викладання предмету у семестрах, кількість годин на проведення лекцій, практичних та лабораторних занять, кількість кредитів ECTS. Серед членів класу є посилання на об’єкт класу SubjectDictionary, який містить основну інформацію про предмет, а також колекції об’єктів класів Control та CurriculumSubject, які пов’язані з поточним навчальним предметом.

SubjectDictionary – клас сутності «dict_subjects». Містить інформацію про елементи словника предметів: назву, посилання на підрозділи у навчальному плані та робочому навчальному плані, посилання на тип предмета, кафедру та об’єкт класу SubjectDictionary, який є загальним предметом для поточного предмета. Також у класі наявні колекції об’єктів класів SubjectDictionary (предмети, для яких поточний предмет є загальним) та Subject (академічні предмети, які пов’язані з поточним елементом словника).

SubjectType – клас сутності «subject_type». Містить інформацію про типи предметів: назву.

WorkType – клас сутності «type_of_work». Містить інформацію про види роботи по підготовці та захисту дипломного проекту: назву.

Workplan – клас сутності «curriculum», який розширює клас Curriculum і відповідає записам у таблиці з позначкою «workplan». Окрім членів класу, наслідуваних від класу Curriculum, має і власні поля: посилання на практику, державну атестацію та навчальний план, до якого відноситься поточний робочий навчальний план. Також містить колекції об’єктів класів CurriculumSubject, DiplomaPreparation та AcademicGroup, які пов’язані з поточним робочим навчальним планом.

Вибірка, модифікація, створення та видалення записів таблиць БД реалізуються через методи DAO-класів, які реалізують параметризований інтерфейс IDAO <T extends IDatabaseEntity> та мають значення параметра, що відповідає класу додатку, який відображує потрібну таблицю БД. На даний момент у системі існує одна реалізація інтерфейсу IDAO з використанням ORM Hibernate Framework – клас HibernateDAO <T extends IDatabaseEntity>.

Іншим пов’язаним із взаємодією з БД функціоналом є переключення між БД різних років, яке включає в себе дві підзадачі:

- переключення на існуючу БД;
- створення БД для нового року.

Функціонал переключення з поточної БД на БД певного року реалізовано у методі public static boolean switchDatabase (int year) класу DatabaseSwitcher. Метод приймає рік БД і визначає, чи існує БД цього року у системі. Якщо так, то виконується переключення і повертається значення true. Якщо ні, то виконується перевірка наявності БД попереднього року. Якщо такої БД у системі немає, то не виконується жодних дій і повертається значення false. У тому випадку, коли така БД знайшлась у системі, створюється БД потрібного року і заповнюється даними, отриманими з БД попереднього року.

Створення нової БД виконується через прописаний у програмному коді SQL-запит.

Структура нової БД (таблиці, їх атрибути та властивості атрибутів, зв'язки між таблицями тощо) створюється автоматично завдяки засобам HibernateFramework. Джерелом інформації про структуру БД при її генеруванні є класи-сутності.

Перенесення інформації до нової БД виконано у класі DatabaseCloner. Спочатку за допомогою рефлексії створюється список усіх класів додатку, які реалізують інтерфейс IDatabaseEntity. Потім для кожного з цих класі виконується вибірка даних і записується у відображенні, де ключем клас, а значенням – список об’єктів класу, отриманий через взаємодію з БД.

Після формування відображення виконується закриття поточної сесії (тобто, сесії з БД попереднього року) і відкриття сесії, пов’язаної з новою БД. Для кожного елементу відображення, окрім академічних груп, інформація копіюється у нову БД у незмінному вигляді. Для цього було створено власний генератор ідентифікаторів, який використовується усіма класами-сутностями. Генератор працює таким чином, що якщо ідентифікатор сутності, яку необхідно створити, більший за 0, то запис створюється з цим ідентифікатором, а якщо ідентифікатор рівний або менший за 0 – ідентифікатор генерується засобами Hibernate Framework. Копіювання даних за старої БД у нову з незмінними ідентифікаторами дозволяє зберегти цілісність зв’язків між об’єктами.

Для перенесення інформації про академічні групи виконується коригування даних, а саме:

- видалення випускних груп минулого року;
- внесення до БД нових для поточного року груп та автоматичне визначення і запис у БД їх шифрів;
- перерозподіл робочих планів (зміщення на один курс).

3.3 Реалізація автоматичного генерування документації

3.3.1 Загальні положення

Загальна архітектура компоненту системи, призначеного для автоматичного генерування документації з наявної у БД інформації наведена у попередньому розділі (див. рис. 2.6). Далі буде розглянуто генерування кожного виду нормативної документації окремо. Іншими словами, буде описано механізм роботи кожного класу, який розширює клас IDocumentGenerator, а також систему допоміжних класів, необхідних для реалізації генерування документації.

Автоматичне генерування будь-яких нормативних документів складається з двох основних етапів: вибірка даних з БД та запис отриманих результатів у документ. Також для деяких видів нормативних документів може бути присутнім ще один етап: обробка інформації, яка виконується після вибірки даних з БД, але перед записом результату у документ.

Вибірка даних з БД відбувається через об’єкти класів-сутностей та об’єкти DAO рівня взаємодії з БД. Обробка інформації здійснюється за допомогою чітко визначених алгоритмів, реалізованих у програмному коді. Реалізація етапу запису результатів у документ потребує використання бібліотеки Apache POI та системи власноруч створених допоміжних класів, які представляють різні частини документу. Використання такої системи класів спрощує процес пошуку позначок, заповнення логічно відокремлених структурних частин та виконання інших дій з документом.

3.3.2 Допоміжні класи для представлення частин документу

Ієрархію допоміжних абстрактних класів із зазначенням переліку їх абстрактних методів зображено на рис. 3.1.

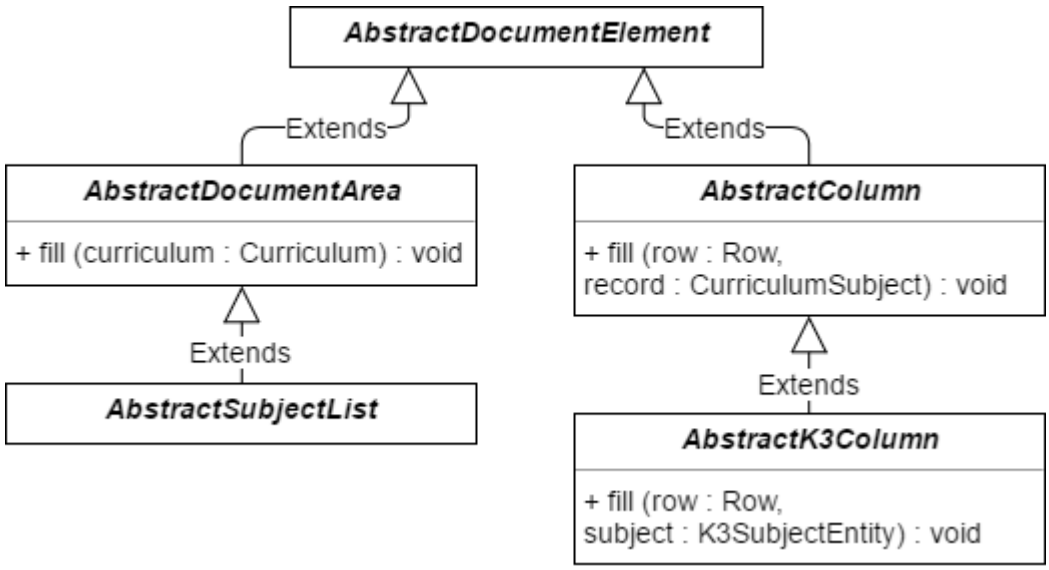

Рисунок 3.1 – Ієрархія абстрактних класів елементів документу

На найвищому рівні абстракції у цій системі знаходиться клас AbstractDocumentElement, який є узагальненим представленням будь-якого елементу аркуша книги Excel.

Клас AbstractDocumentElement має лише одного неабстрактного нащадка – клас DocumentSection, який представляє підрозділ навчальних предметів у навчальному плані (робочому навчальному плані) і використовується для пошуку рядка аркушу, з якого починається перелік предметів певного підрозділу.

Клас AbstractDocumentArea розширює клас AbstractDocumentElement і відповідає деякій структурно відокремленій області документа. У цьому класі визначено абстрактний метод public void fill (Curriculum curriculum), який приймає у якості аргументу навчальний план (робочий навчальний план) і записує отриману з об’єкта інформацію у документ. Тип даних, які вилучаються з об’єкту curriculum залежить від реалізації класу.

Клас AbstractDocumentArea має наступні конкретні реалізації:

- DiplomaPreparationArea – таблиця робочого навчального плану з інформацією про розподіл годин з підготовки та захисту дипломного проекту;
- PracticeArea – таблиця робочого навчального плану з інформацією про проходження практики;
- StateCertificationArea – таблиця робочого навчального плану з інформацією про проведення державної атестації.

В свою чергу, клас AbstractSubjectList розширює клас AbstractDocumentArea. Цей клас призначений для позначення списку предметів та сукупності деякої інформації про них, яку можна отримати з БД. Одним з членів класу є колекція об’єктів класу AbstractColumn, які повинні бути заповнені у рамках поточного списку предметів.

Клас AbstractSubjectList має дві конкретні реалізації – CurriculumSubjectList та WorkplanSubjectList, які відповідно представляють список предметів у навчальному плані і в робочому навчальному плані. Через подібність формування навчального плану і робочого навчального плану основний функціонал автоматичного генерування реалізовано у класі AbstractSubjectList.

Класи CurriculumSubjectList і WorkplanSubjectList перевизначають деякі методи свого суперкласу, що дозволяє застосовувати поліморфізм під час виконання. Основною відмінністю між реалізаціями класу AbstractSubjectList є те, які саме колонки заповнюються під час запису у документ.

Клас AbstractColumn є абстрактним нащадком класу AbstractDocumentElement. Як видно з назви, цей клас використовується для позначення колонок у документі, які повинні бути заповнені певними даними для кожної навчальної дисципліни. У цьому класі оголошено абстрактний метод public void fill (Row row, CurriculumSubject record), який витягує потрібну інформацію з об’єкта record і записує її у потрібний рядок документа.

Клас має наступні конкретні реалізації:

- CipherColumn – колонка шифру предмета у поточному навчальному плані (робочому навчальному плані);
- ControlColumn – колонка, яка містить номер семестру контрольного заходу для предмета (або порожню клітинку, якщо такий контрольних захід відсутній для даної дисципліни); тип контрольного заходу передається як аргумент конструктору;
- DepartmentColumn – колонка з назвою кафедри, що викладає дисципліну;
- HoursColumn – колонка з відомостями про кількість кредитів ECTS, кількість годин лекцій, практичних або лабораторних занять з предмету; у якості аргументу конструктора передається посилання на метод класу Subject, який повертає потрібне значення (Subject::getEcts(), Subject::getLections() тощо); для реалізації такої поведінки було створено функціональний інтерфейс SubjectProperty з методом public double getValue (Subject subject);
- SemesterColumn – колонка з підсумковими відомостями про предмет, які стосуються лише конкретного семестру (права частина навчального плану та робочого навчального плану); номер семестру та кількість тижнів у семестрі передається у конструкторі класу;
- SimpleColumn – проста колонка, яка може містити будь-яке значення;
- TitleColumn – колонка з назвою предмета.

Клас AbstractColumn розширюється класом AbstractK3Column, який відповідає специфічним колонкам форми К-3. У цьому класі оголошено метод public void fill (Row row, K3SubjectEntity subject), який витягує потрібну інформацію з об’єкту допоміжного класу, переданого у якості аргументу, і записує її у потрібний рядок документа. Допоміжний клас K3SubjectEntity містить інформацію про дисципліну, необхідну для заповнення форми К-3.

Клас AbstractK3Column має наступні конкретні реалізації:

- ControlK3Column – колонка, яка містить кількість заходів контролю певного типу для предмету; у тому випадку, коли типом контролю є курсова робота або курсовий проект, вказується кількість годин на виконання роботи; тип контролю передається у конструкторі класу;
- FullTitleColumn – колонка з назвою предмета у форматі назви для форми К-3 – «назва факультету – назва предмету – обсяг предмету в годинах – номер курсу – перелік груп», при цьому обсяг предмету в годинах визначається як кількість кредитів ECTS, помножена на обсяг одного кредиту (30 годин), а групи, яким викладається предмет, перераховуються через кому у форматі «шифр групи (кількість студентів бюджетної форми навчання + кількість студентів контрактної форми навчання);
- GroupsK3Column – колонка, в якій зазначається кількість академічних груп або підгруп для проведення практичних чи лабораторних занять; тип фінансування (бюджетні/контрактні) та тип групи (академічна, практична, лабораторна) передаються у конструкторі;
- HoursK3Column – колонка з відомостями про кількість кредитів ECTS, кількість годин лекцій, практичних та лабораторних занять з предмету;
- K3StreamColumn – колонка, яка містить кількість бюджетних (для бюджетної форми К-3) або контрактних (для контрактної форми К-3) потоків;
- NumberK3Column – колонка з порядковим номером предмета у формі;
- StudentsK3Column – колонка, яка може містити кількість студентів бюджетної або контрактної форми навчання у бюджетних або контрактних групах; при цьому джерело фінансування групи та студентів, кількість яких визначається, передаються у конструкторі класу.

3.3.3 Генерування навчального плану та робочого навчального плану

Для створення навчального плану виконується прохід по всіх аркушах шаблону. Для кожного аркуша визначається ідентифікатор навчального плану, який відповідає цьому аркушу. Далі для кожної сторінки навчального плану створюється об’єкт класу CurriculumSubjectList, для якого викликається метод public void fiil (Curriculum curriculum), де curriculum – об’єкт навчального плану, отриманий з БД за визначеним ідентифікатором.

Аналогічно генерується робочий навчальний план, але для кожної сторінки плану створюються об’єкти класів WorkplanSubjectList, PracticeArea, StateCertificationArea та DiplomaPreparationArea, кожен з яких заповнює відповідну частину сторінки робочого навчального плану.

Усі дані, необхідні для заповнення як навчального плану, так і робочого навчального плану, можуть бути отримані безпосередньо з БД і не потребують додаткової обробки перед записом у документ.

3.3.4 Генерування форми К-3

Для створення форми К-3 здійснюється прохід по всіх аркушах шаблону. Для кожного аркушу визначається джерело фінансування (бюджет або контракт) та форма навчання (денна або заочна), після чого формується список об’єктів класу K3SubjectEntity. У допоміжному класі K3SubjectEntity міститься посилання на навчальний предмет (об’єкт класу Subject) та деяка інша необхідна для заповнення форми інформація, що не міститься у БД і заповнюється за алгоритмом.

Для формування списку предметів здійснюється пошук усіх предметів з робочих навчальних планів, які задовольняють двом умовам:

- предмет викладається тією кафедрою, для якої складається форма К-3;
- предмет викладається у хоча б одній групі з формою навчання, що еквівалентна формі навчання поточного аркушу форми К-3.

Після формування списку виконується попередня обробка даних, яка полягає в розбитті академічних груп на підгрупи для проведення практичних та лабораторних занять за алгоритмом, описаним в першому розділі цієї роботи.

У випадках, коли один і той самий предмет записується у двох різних рядках форми К-3, але викладається одночасно у всіх групах, зазначених в обох цих рядках, для другого рядка видаляється значення кількості годин для проведення лекцій. Аналогічно видаляється значення кількості годин для проведення практичних та/або лабораторних занять, якщо при розбитті на підгрупи за алгоритмом виявилось так, що для усіх груп, зазначених в обох рядках, практичні та/або лабораторні заняття проводяться одночасно. Для предметів, записаних у трьох чи більше рядках форми К-3 діють аналогічні правила.

3.4 Реалізація взаємодії з клієнтською частиною додатку

Узагальнена архітектура класів, які розширюють клас HttpServlet і реалізують механізм взаємодії серверної частини додатку з клієнтською через обміни HTTP-запитами представлена у попередньому розділі (див. рис. 2.5). Для організації передачі об’єктів у запитах у форматі JSON також було створено низку класів, призначених для серіалізації об’єктів певних типів у формат JSON.

3.4.1 Реалізація обробки HTTP-запитів

Обробка запитів від клієнтської частини додатку виконується у класах, які розширюють клас HttpServlet. Розглянемо спочатку запити, які обробляються класами, що розширюють клас AbstractEntityController, тобто реалізують механізм редагування БД з клієнтської машини.

Підкласи цього абстрактного класу мають назви, які формуються за принципом «назва класу-сутності» + «controller». Для кожного класу-сутності у системі існує відповідний клас-контролер, що забезпечує можливість перегляду та редагування вмісту відповідної таблиці через інтерфейс користувача.

Так як функціонал всіх підкласів класу AbstractEntityController є дуже подібним, основна реалізація виконана у суперкласі, а у підкласи винесені методи для створення об’єкту з його представлення у форматі JSON та формування випадаючого списку з об’єктів відповідного класу.

Під час обробки запиту у класі AbstractEntityController, насамперед, із запиту вилучається значення параметру «action», який може приймати наступні значення:

- list – запит на отримання списку об’єктів класу, отриманих шляхом вибірки з БД усіх записів відповідної таблиці;
- create – запит на створення запису;
- update – запит на модифікацію запису;
- delete – запит на видалення запису;
- dependencyList – запит на отримання списку об’єктів класу, отриманих вибіркою з відповідної таблиці БД записів за певною умовою;
- dropdownList – запит на отримання списку об’єктів класу у форматі «ідентифікатор – строкове представлення».

Після визначення потрібної дії виконується відповідний запит до БД, результат якого відправляється клієнту у відповіді сервера у форматі JSON.

Функціонал завантаження готових документів і шаблонів з сервера, а також оновлення шаблонів на сервері реалізований у класі DownloadController. Для визначення змісту запиту необхідно визначити значення наступних параметрів:

- downloadGenerated – завантаження готової згенерованої документації з сервера;
- downloadTemplate – завантаження шаблонів нормативних документів з сервера;
- uploadTemplate – завантаження шаблонів нормативних документів на сервер.

Завантаження файлів з сервера реалізовано за допомогою запису файлу у вихідний потік об’єкту класу HttpServletResponse, з якого формується відповідь сервера. Завантаження файлів на сервер відбувається через зчитування файлу з вхідного потоку об’єкту класу HttpServletRequest, яким представлено запит клієнта.

Можливість переключення між БД різних років реалізована у класі YearChangeController, який також розширює клас HttpServlet. Необхідна дія контролера визначається як значення параметру «action», який може приймати наступні значення:

- list – повернення списку усіх років, для яких у системі існують БД;
- switch – переключення на БД року, зазначеного у запиті як значення параметру «years».

Після визначення змісту запиту виконується обробка цього запиту за допомогою виклику методів класу DatabaseSwitcher і формування відповіді сервера.

3.4.2 Реалізація представлення об’єктів у форматі JSON

Для представлення об’єктів у форматі JSON використовується бібліотека Gson від Google. При серіалізації створюється об’єкт класу Gson, який має метод public void toJson (Object src), де src – об’єкт, представлення якого необхідно отримати [4].

Об’єкт класу Gson створюється за допомогою методу public Gson create () класу GsonBuilder. Для того, щоб серіалізація максимально задовольняла вимоги системи, потрібно виконати початкові налаштування об’єкту класу GsonBuilder.

При передачі об’єктів класів-сутностей виникає необхідність використовувати для серіалізації лише деякі поля, для чого над обраними поля потрібно проставити анотацію @Expose, а при створенні GsonBuilder викликати для нього метод public Gson excludeFieldsWithoutExposeAnnotation (), який виключає поля без цієї анотації з процесу серіалізації.

Інший аспект серіалізації об’єктів класів-сутностей у додатку обумовлений особливостями роботи Hibernate Framework. Колекції об’єктів, які представляють зв’язок типу one-to-many однієї сутності з іншою, мають прописану в анотації @OneToMany властивість – FetchType, яка може приймати наступні значення:

- eager – завантаження об’єктів колекції одразу під час завантаження кореневої сутності; суттєвим недоліком є те, що для об’ємних БД з великою кількістю зв’язків завантаження навіть одного об’єкта може стати дуже ресурсомісткою операцією, так як при цьому виконується завантаження усіх пов’язаних з цим об’єктом сутностей;
- lazy – відкладена ініціалізація, завантаження об’єктів колекції при першому зверненні до них з програмного коду; до першого звернення об’єкти з відкладеною ініціалізацією представлені у формі проксі-об’єктів (HibernateProxy).

Виходячи з того, що додаток використовує досить велику БД, а класи-сутності містять значну кількість зв’язків з іншими об’єктами інших класів-сутностей, було обрано стратегію відкладеної ініціалізації для економії часу обробки запитів та оперативної пам’яті.

При виконанні серіалізації об’єкта з відкладеною ініціалізацією, який раніше не було ініціалізовано у програмному коді, Gson намагається серіалізувати не сам об’єкт, а об’єкт класу HibernateProxy, що викликає помилки у роботі програми. Для вирішення цієї проблеми було написано клас HibernateProxyTypeAdapter, який розширює клас TypeAdapter <HibernateProxy> бібліотеки Gson. У цьому класі виконується попередня обробка об’єктів типу HibernateProxy та їх ініціалізація перед переформуванням об’єкту у представлення у форматі JSON. Для того, щоб прив’язати розроблений клас-адаптер до об’єкту класу GsonBuilder потрібно викликати у цього об’єкта метод public GsonBuilder registerTypeAdapterFactory (TypeAdapterFactory factory).

Серіалізація деяких об’єктів класів-сутностей потребує окремого підходу, який не реалізовано у бібліотеці Gson, а саме – серіалізація пов’язаних об’єктів (тобто, членів класу, які є іншими сутностями, пов’язаними з поточною сутністю зв’яком one-to-one або many-to-one) не як окремих об’єктів, а у вигляді ідентифікатора сутності. Для реалізації такого підходу бібліотека Gson надає розробникам можливість визначати власні класи для серіалізації та десеріалізації.

Суперкласом для всіх серіалізаторів класів-сутностей є абстрактний клас AbstractEntitySerializer <T extends IDatabaseEntity>, який реалізовує інтерфейс JsonSerializer <T>, описаний у бібліотеці Gson, але не перевизначає абстрактний метод public JsonElement serialize (T t, Type type, JsonSerializationContext jsc), при цьому відповідальність за визначення цього методу покладається на нащадків класу.

У класі AbstractEntitySerializer реалізовано один допоміжний метод protected void addProperty (JsonElement element, String prop, IDatabaseEntity obj), який додає до представлення об’єкту у форматі JSON (element) властивість з визначеним ім’ям (prop) та значенням, яке дорівнює ідентифікатору об’єкту obj.

Для серіалізації об’єктів класів-сутностей було створено наступних нащадків класу AbstractEntitySerializer:

- AcademicGroupSerializer;
- ControlSerializer;
- CurriculumSubjectSerializer;
- DiplomaPreparationSerializer;
- SectionSerializer;
- SpecializationSerializer;
- SubjectDictionarySerializer;
- SubjectSerializer;
- WorkplanSerializer.

Усі інші сутності додатку можуть бути серіалізовані стандартними засобами бібліотеки Gson.

Для прив’язки серіалізатора до об’єкту GsonBuilder необхідно викликати метод public GsonBuilder registerTypeAdapter (Type type, Object typeAdapter).

3.5 Висновки

У цьому розділі було виконано опис реалізації системи автоматизованого документообігу. Для кожного компоненту системи було наведено механізм його реалізації та перелік основних класів компоненту з описом їх функціоналу та призначення.

В результаті для компоненту взаємодії з БД було отримано детальний опис класів-сутностей, а також механізму переключення між БД та створення нових БД для заданих користувачем років.

Для підсистеми автоматичного генерування документації було описано набір допоміжних класів, які використовуються для роботи з окремими частинами аркушів, що дозволяє структурувати роботу з файлами і спростити процес внесення змін до алгоритму генерації нормативних документів. Для кожного виду документації було наведено алгоритм генерування та запису у файл для отримання готового документу.

Для підсистеми взаємодії з клієнтською частиною додатку було виконано опис усіх розроблених класів-сервлетів, структури HTTP-запитів з описом параметрів запитів, а також процес переформування об’єктів у представлення у форматі JSON за допомогою бібліотеки Gson.

Розроблена система повністю задовольняє функціональним вимогам, а завдяки гнучкій архітектурі та використанню абстракції різних рівнів є придатною для розширення та удосконалення у майбутньому.

4 ЕКОНОМІКО-ОРГАНІЗАЦІЙНИЙ РОЗДІЛ

4.1 Вступ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для автоматизації документообігу. Серверна частина додатку реалізована мовою Java з використанням технологій Java EE та сторонніх бібліотек у середовищі розробки NetBeans IDE.

Програмний продукт є крос-платформним і призначений для використання на серверах під управлінням операційної системи сімейства Windows або Unix.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями [3].

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій. Фактично цей метод працює за таким алгоритмом:

- Визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають.

На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і, відповідно, витрати.

- Для кожної функції визначаються повні річні витрати й кількість робочих часів.
- Для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- Після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.2 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки серверної частини системи автоматизованого документообігу. Оскільки основні проектні рішення стосуються всієї серверної частини системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту.

Відповідно цьому варто обирати і систему показників якості програмного продукту. Технічні вимоги до продукту наступні:

- функціонування на серверах із стандартним набором компонент;
- забезпечення задовільної швидкості обробки великих об’ємів даних з БД у реальному часі;
- забезпечення простоти взаємодії з клієнтською частиною додатку або іншими веб-сервісами, які використовують API розробленої системи;
- придатність до розширення, модифікації та удосконалення у майбутньому, яка забезпечується гнучкою архітектурою та зрозумілістю вихідного коду;
- мінімізація часових і матеріальних витрат на впровадження програмного продукту.

4.2.1 Обґрунтування функцій програмного продукту

Головна функція F0 – розробка серверної частини системи автоматизованого документообігу, яка генерує нормативні документи з наявної у БД інформації та взаємодіє з клієнтською частиною додатку через HTTP-запити, виконуючи вибірку і модифікацію даних БД за запитом клієнта, а також генерацію нормативних документів. Виходячи з конкретної мети, можна виділити наступні основні функції:

- F1 – вибір технології взаємодії з БД;
- F2 – вибір бібліотеки для роботи з файлами формату .xls;
- F3 – вибір формату представлення об’єктів у HTTP-запитах.

Кожна з вищезазначених основних функцій може мати декілька варіантів реалізації:

- Функція F1:
  а) бібліотека JDBC;
  б) JPA та Hibernate Framework.
- Функція F2:
  а) бібліотека docx4j;
  б) бібліотека Apache POI.
- Функція F3:
  а) формат XML;
  б) формат JSON.

4.2.2 Варіанти реалізації основних функцій

Варіанти реалізації вищеперерахованих основних функцій представлено на рис. 4.1 у вигляді морфологічної карти системи.

Морфологічна карта відображує всі можливі комбінації варіантів реалізації основних функцій, які складають повну множину варіантів програмного продукту.

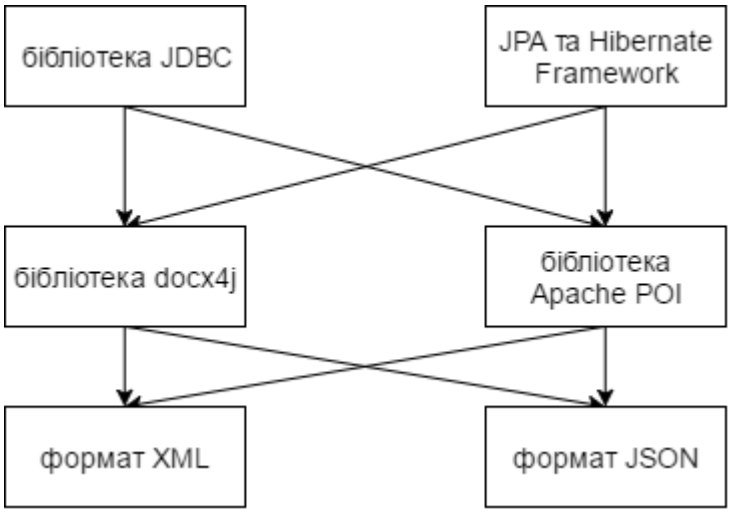

Рисунок 4.1 – Морфологічна карта

На основі наведеної морфологічної карти побудовано позитивно-негативну матрицю варіантів основних функцій (табл. 4.1).

Таблиця 4.1 – Позитивно-негативна матриця

Основна функція	Варіант реалізації	Переваги	Недоліки
F1	А	Простий у використанні, майже не потребує додаткових знань, окрім мови Java та SQL	Реалізація не є гнучкою; програмний код складний для сприйняття і підтримки
	Б	Гнучкий; дозволяє писати чистий і зрозумій код	Потребує додаткового вивчення JPA
F2	А	Можливість використання XML для генерації	Обмежений функціонал, недостатній для поставленої задачі
	Б	Широкий функціонал	Складність застосування
F3	А	Має високий рівень безпечності	Складність; займає більше пам’яті
	Б	Простота; займає невелику кількість пам’яті	Є менш безпечним, ніж XML

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, так як вони не відповідають поставленим перед програмним продуктом задачам. Розглянемо відповідність варіантів необхідному функціоналу системи по кожній функції окремо:

- функція F1: оскільки придатність системо до розширення і удосконалення у майбутньому є одним з критично важливих пунктів вимог до системи, який потребує максимальної простоти і гнучкості програмного коду, варіант б) має бути відкинутий;
- функція F2: варіант б) має бути відкинутий через те, що при його застосуванні буде неможливо реалізувати повний функціонал системи автоматизованої генерації документів;
- функція F3: формат представлення об’єктів в HTTP-запитах не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти а) та б) гідними розгляду.

Таким чином, будемо розглядати наступні варіанти реалізації програмного продукту:

- F1a – F2a – F3a;
- F1a – F2a – F3б.

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування параметрів програмного продукту

4.3.1 Опис параметрів

На підставі даних про основні функції, які повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X1 – час обробки даних – відображає час, який необхідний для формування представлення списку об’єктів в обраному форматі і навпаки;
- X2 – об’єм пам’яті для збереження даних – відображає об’єм пам’яті, необхідний для зберігання об’єкта в обраному форматі;
- X3 – час передачі даних по мережі – відображає час, за який HTTP-запит, який містить представлення списку об’єктів в обраному форматі, буде передано локальною мережею;
- X4 – потенційний об’єм програмного коду – відображає об’єм програмного коду, який необхідно реалізувати.

4.3.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі умов, що характеризують експлуатацію системи (табл. 4.2). Оцінка часу обробки даних, об'єму пам'яті для збереження та часу передачі мережею виконувалася для обробки списку предметів з БД у форматі, придатному для передачі у запиті.

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	Умовне позначення	Одиниці виміру	Значення параметра		
			гірше	середнє	краще
Час обробки даних	X1	мс	400	250	100
Об’єм пам’яті для збереження даних	X2	Кб	50	35	20
Час передачі даних по мережі	X3	мс	800	500	200
Потенційний об’єм програмного коду	X4	кількість рядків	10000	7500	5000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

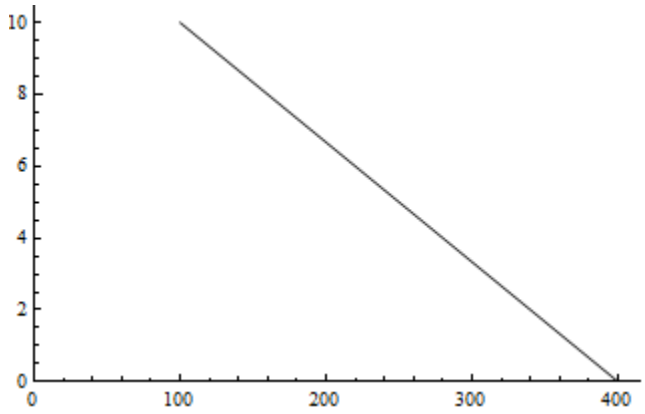

Рисунок 4.2 – X1, час обробки даних

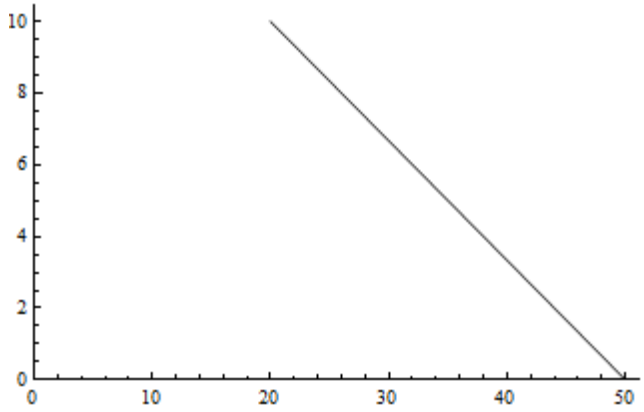

Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

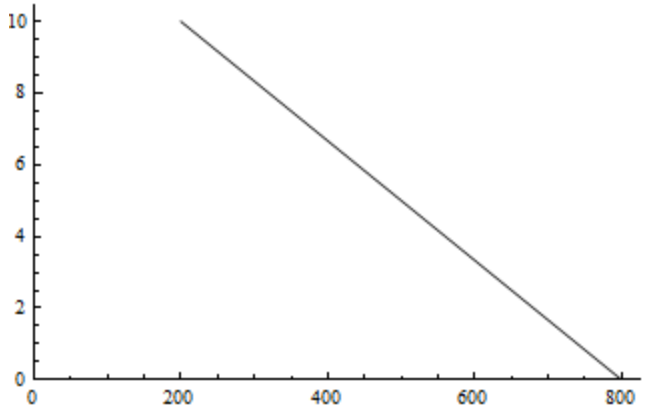

Рисунок 4.4 – X3, час передачі даних по мережі

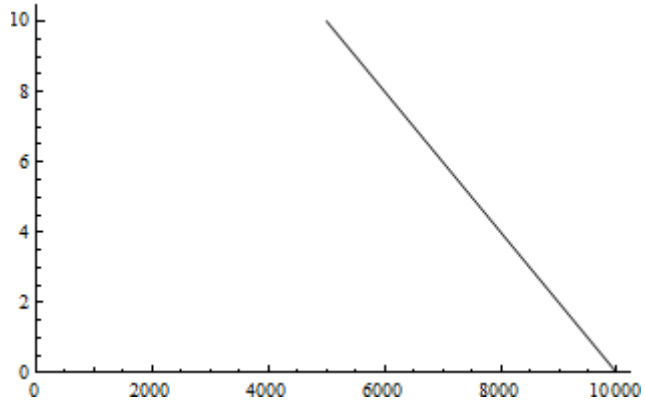

Рисунок 4.5 – X4, потенційний об’єм програмного коду

Побудовані графічні характеристики параметрів відображують залежність рівня оптимальності рішення від значення відповідного параметра.

4.3.3 Аналіз експертного оцінювання

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка серверної частини системи автоматизованого документообігу, яка реалізує функціонал генерування нормативної документації з наявної у БД інформації та взаємодії з клієнтом через HTTP-запити та обміни об’єктами у форматі XML або JSON у запитах.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування методом попарного порівняння наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							$\sum R_i$	Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Час обробки даних	мс	2	2	1	1	2	1	1	10	-7,5	56,25
X2	Об'єм пам'яті для збереження даних	Кб	4	4	4	4	4	3	3	26	8,5	72,25
X3	Час передачі даних по мережі	мс	3	3	3	3	3	4	4	23	5,5	30,25
X4	Потенційний об'єм програмного коду	кількість рядків	1	1	2	2	1	2	2	11	-6,5	42,25
	Разом		10	10	10	10	10	10	10	70	0	201

Для перевірки ступеня достовірності експертних оцінок, визначимо наступні параметри, необхідні для виконання подальших розрахунків ступеня достовірності:

- сума рангів кожного з параметрів і загальна сума рангів по всім параметрам:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

- середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5;$$

- відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T,$$

сума відхилень по всім параметрам повинна дорівнювати 0:

$$\sum_{i=1}^{N} \Delta_i = 0;$$

- загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 201.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 201}{7^2(4^3 - 4)} = 0{,}82 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, який дорівнює 0,67.

Проведемо попарне порівняння всіх параметрів (табл. 4.4).

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1,5
X1 і X3	>	>	>	>	>	>	>	>	1,5
X1 і X4	<	<	>	>	<	>	>	>	1,5
X2 і X3	<	<	<	<	<	>	>	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається за формулою:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } \mathrm{X}_i > \mathrm{X}_j \\ 1{,}0 \text{ при } \mathrm{X}_i = \mathrm{X}_j \\ 0{,}5 \text{ при } \mathrm{X}_i < \mathrm{X}_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i}$$

$$\text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються ітеративно разів доти, поки значення, отримані на поточній ітерації, не будуть незначно відрізнятися від значень попередньої ітерації (не більше, ніж на 2%). Починаючи з другої ітерації відносні оцінки розраховуються з врахуванням результатів попередніх ітерації за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'},$$

$$\text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Результати розрахунків за вищенаведеними формулами занесемо у таблицю 4.5. Як видно з отриманої таблиці, різниця значень коефіцієнтів вагомості на третій ітерації розрахунків у порівнянні з другою ітерацією не перевищує 2%, тому можна вважати отриманий результат задовільним і більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

X_i	X_j				I ітерація		II ітерація		III ітерація	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1,0	1,5	1,5	1,5	5,5	0,34	21,25	0,36	77,875	0,36
X2	0,5	1,0	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
X3	0,5	1,5	1,0	0,5	3,5	0,22	12,25	0,21	44,875	0,21
X4	0,5	1,5	1,5	1,0	4,5	0,28	16,25	0,27	59,125	0,27
Всього:					16	1	59	1	216	1

4.4 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Для основних функцій F1 та F2 було визначено, що лише один варіант реалізації кожної з цих функцій задовольняє функціональним вимогам до програмного продукту. Отже, вибір варіантів для функцій F1 та F2 було здійснено на попередньому етапі, тому для цих функцій не потрібно проводити розрахунок рівня якості.

Отже, розрахунок рівня якості буде проведено для кожного варіанта реалізації основної функції F3 по усіх чотирьох визначених параметрах якості з врахуванням коефіцієнта вагомості параметрів.

Коефіцієнт технічного рівня для кожного варіанта реалізації програмного продукту (таблиця 4.6) розраховується за наступною формулою [5]:

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} ,$$

де n – кількість параметрів, $K_{вi}$ – коефіцієнт вагомості i–го параметра, B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F3(X1)	А	350	1,67	0,36	0,6
	Б	200	6,67		2,4
F3(X2)	А	40	3,33	0,16	0,53
	Б	20	10		1,6
F3(X3)	А	400	6,67	0,21	1,4
	Б	250	9,17		1,93
F3(X4)	А	7500	5	0,27	1,35
	Б	7000	6		1,62

За даними з таблиці 4.6 визначаємо рівень якості кожного з варіантів за наступною формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}].$$

Отримані значення рівня якості для кожного з варіантів дорівнюють наступним:

$K_{K1} = 0,6 + 0,53 + 1,4 + 1,35 = 3,88,$

$K_{K2} = 2,4 + 1,6 + 1,93 + 1,62 = 7,55.$

Як видно з результатів розрахунків, рівень якості кращим при застосуванні другого варіанту реалізації ПП, для якого коефіцієнт технічного рівня має найбільше значення.

4.5 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості. Всі варіанти включають в себе три окремих завдання:

1. Розробка архітектури серверної частини додатку.
2. Розробка системи автоматичного генерування документації з наявної у БД інформації.
3. Розробка системи взаємодії з клієнтською частиною додатку на основі обмінів HTTP-запитами.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б, а завдання 3 – до групи В. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1, а алгоритми завдання 2 та завдання 3 – до групи 3.

Для реалізації завдання 1 використовується нормативно-довідкова інформація, а завдання 2 та 3 використовують інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється за формулою:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.1)$$

де T_P – трудомісткість розробки ПП, $K_П$ – поправочний коефіцієнт; $K_{СК}$ – коефіцієнт на складність вхідної інформації, K_M – коефіцієнт рівня мови програмування; $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм, $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації: $K_П$ = 1,7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації: $K_{СК}$ = 1,16.

Оскільки при розробці використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ} = 0,9$. При розробці не використовується стандартне математичне забезпечення, тому $К_{СТ.М} = 1$. Тоді, за формулою (4.1), загальна трудомісткість виконання першого завдання дорівнює:

$$Т_1 = 90 \cdot 1,7 \cdot 1,16 \cdot 0,9 \cdot 1 = 159,732 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни – Б) трудомісткість та коефіцієнти приймають наступні значення: $Т_Р = 19$ людино-днів, $К_П = 0,75$, $К_{СК} = 1,07$, $К_{СТ} = 0,8$, $К_{СТ.М} = 1$. Загальна трудомісткість за формулою (4.1) дорівнює:

$$Т_2 = 19 \cdot 0,75 \cdot 1,07 \cdot 0,8 \cdot 1 = 12,198 \text{ людино-днів.}$$

Для третього завдання (використовується алгоритм третьої групи складності, степінь новизни – В) при першому варіанті реалізації (використання формату XML) трудомісткість та коефіцієнти приймають наступні значення: $Т_Р = 12$ людино-днів, $К_П = 0,5$, $К_{СК} = 1$, $К_{СТ} = 0,9$, $К_{СТ.М} = 1$. Значення коефіцієнту використання стандартних модулів і прикладних програм $К_{СТ}$ обумовлене тим, що формат XML не є популярним як формат представлення об’єктів у HTTP-запитах, процес переформування об’єкта у його представлення і навпаки і складним (в т.ч. через відсутність стандартних засобів для представлення масивів та списків), а бібліотеки для роботи з форматом XML є складними для вивчення і використання. Загальна трудомісткість за формулою (4.1) дорівнює:

$$Т_{З(І)} = 12 \cdot 0,5 \cdot 1 \cdot 0,9 \cdot 1 = 5,4 \text{ людино-днів.}$$

Для третього завдання при другому варіанті реалізації (використання формату JSON) завдяки різноманіттю готових рішень та бібліотек для роботи з форматом JSON, які задовольняють функціональним вимогам системи і не

вносять надлишкової складності до архітектури системи коефіцієнт використання стандартних модулів і прикладних програм приймає значення: $К_{СТ}$ = 0,6. Загальна трудомісткість за формулою (4.1) дорівнює:

$$Т_{З(І)} = 12 \cdot 0,5 \cdot 1 \cdot 0,6 \cdot 1 = 3,6 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_{І} = (159,732 + 12,198 + 5,4) \cdot 8 = 1418,64 \text{ людино-годин,}$$
$$Т_{ІІ} = (159,732 + 12,198 + 3,6) \cdot 8 = 1404,24 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант І.

В розробці беруть участь два програмісти з окладом 12000 грн., один архітектор програмного забезпечення з окладом 24000 грн. Визначимо зарплату за годину за формулою:

$$С_{Ч} = \frac{М}{T_m \cdot t} \text{ грн.,}$$

де М – місячний оклад працівників; T_m – кількість робочих днів на місяць; t – кількість робочих годин на день.

Тоді значення витрат на оклад працівників за годину дорівнюватиме:

$$С_{Ч} = \frac{12000 + 12000 + 24000}{3 \cdot 20 \cdot 8} = 100 \text{ грн.}$$

Розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_{Ч} \cdot Т_{i} \cdot К_{Д},$$

де $С_{Ч}$– величина погодинної оплати праці програміста; $Т_{i}$ – трудомісткість відповідного завдання; $К_{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I. } С_{ЗП} = 100 \cdot 1418,64 \cdot 1,3 = 184423,2 \text{ грн.}$$

$$\text{II. } С_{ЗП} = 100 \cdot 1404,24 \cdot 1,3 = 182551,2 \text{ грн.}$$

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

$$\text{I. } С_{ВІД} = С_{ЗП} \cdot 0,22 = 184423,2 \cdot 0,22 = 40573,1 \text{ грн.}$$

$$\text{II. } С_{ВІД} = С_{ЗП} \cdot 0,22 = 182551,2 \cdot 0,22 = 40161,26 \text{ грн.}$$

Визначимо витрати на оплату однієї машино-години ($С_М$). Для однієї ЕОМ, що обслуговує одного програміста з коефіцієнтом зайнятості 0,2 отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 12000 \cdot 0,2 = 28800 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 28800 \cdot (1 + 0,3) = 37440 \text{ грн.}$$

Відрахування на єдиний соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0,22 = 37440 \cdot 0,22 = 8236,8 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 18000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,1 \cdot 0,25 \cdot 18000 = 4950 \text{ грн.},$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$ – річна норма амортизації; $Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,1 \cdot 18000 \cdot 0,05 = 990 \text{ грн.},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 116 – 20) \cdot 8 \cdot 0.8 = 1465,6 \text{ годин},$$

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$ – коефіцієнт використання приладу протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1465,6 \cdot 0,12 \cdot 1,94 = 341,19 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу; $К_З$ – коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0,67 = 18000 \cdot 0,67 = 12060 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть дорівнювати:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н,$$

$$С_{ЕКС} = 37440 + 8236,8 + 4950 + 990 + 341,19 + 12060 = 64017,99 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 64017,99 / 1465,6 = 43,68 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_М = C_{М\text{-}Г} \cdot T,$$

$$I.\ C_М = 43{,}68 \cdot 1418{,}64 = 61966{,}2 \text{ грн.},$$

$$II.\ C_М = 43{,}68 \cdot 1404{,}24 = 61337{,}2 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0{,}67,$$

$$I.\ C_Н = 184423{,}2 \cdot 0{,}67 = 123563{,}54 \text{ грн.},$$

$$II.\ C_Н = 182551{,}2 \cdot 0{,}67 = 122309{,}30 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_{Н,}$$

$$I.\ C_{ПП} = 184423{,}2 + 40573{,}1 + 61966{,}2 + 123563{,}54 = 410526{,}04 \text{ грн.},$$

$$II.\ C_{ПП} = 182551{,}2 + 40161{,}26 + 61337{,}2 + 122309{,}30 = 406358{,}96 \text{ грн..}$$

4.6 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою [6]:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$$К_{ТЕР1} = 3{,}88 / 410526{,}04 = 0{,}95 \cdot 10^{-5},$$

$$К_{ТЕР2} = 7{,}55 / 406358{,}96 = 0{,}19 \cdot 10^{-4}.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 0{,}19 \cdot 10^{-4}$.

4.7 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, розробленого в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати та інші.

Після виконання ФВА серверної частини системи можна зробити висновок, що з альтернатив, які залишились після першого відбору двох варіантів виконання ПП оптимальним є другий варіант реалізації програмного продукту (використання формату JSON). У цього варіанту виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 0{,}19 \cdot 10^{-4}$.

Обраний за допомогою ФВА варіант реалізації програмного продукту має наступні параметри:

- використання JPA та Hibernate Framework для взаємодії з БД;
- використання бібліотеки Apache POI для роботи з excel-файлами;
- представлення об’єктів у HTTP-запитах у форматі JSON.

Описаний варіант виконання серверної частини системи автоматизованого документообігу реалізує увесь необхідний функціонал та має хороші технічні показники. Також, у порівнянні з іншим розглянутим варіантом, обраний варіант потребує написання меншої кількості коду та має меншу вартість реалізації.

ВИСНОВКИ

У даній роботі виконано аналіз структури нормативних документів НТУУ «КПІ», на основі якого було розроблено алгоритми для автоматичного генерування нормативних документів, необхідний для організації навчального процесу НТУУ «КПІ».

Технології та інструменти для розробки серверної частини документообігу було обрано після вивчення можливих альтернативних рішень та визначення переваг та недоліків різних варіантів реалізації. Для основних використаних технологій наведено обґрунтування вибору та перелік переваг їх використання у порівнянні з іншими альтернативами.

Після визначення набору інструментів та технологій для реалізації серверної частини системи було виконано розбиття системи на модулі згідно загальноприйнятим правилам побудови архітектури веб-додатків. Для кожного модуля здійснено проектування ієрархії класів та інтерфейсів найвищого рівня абстракції, визначення функціональних вимог, механізму виконання поставлених перед компонентом задач та його взаємодії з іншими частинами додатку.

Структурними частинами системи є рівень взаємодії з БД, рівень бізнес-логіки та рівень представлення. Кожен рівень є максимально відокремленим від інших для забезпечення слабкої зв’язності між окремими модулями системи, в результаті чого була отримана система з гнучкою архітектурою. Завдяки такому підходу до проектування значно полегшується підтримка системи, а також її розширення та удосконалення у майбутньому.

Наступним етапом виконання роботи була реалізації спроектованої системи з використанням визначеного набору технологій. Для усіх компонентів системи було наведено детальний опис механізмів їх роботи та функціоналу розроблених у межах компонента класів та інтерфейсів. Логіка програми відповідає принципу, згідно з яким кожен клас виконує лише одну задачу.

Розроблена серверна частина разом з БД і клієнтською частиною складає єдину систему, яка повністю задовольняє функціональним вимогам. Отриманий програмний продукт може використовуватись на кафедрах НТУУ «КПІ» для автоматизації процесу створення нормативних документів та внесення змін до них. Використання системи автоматизованого документообігу значно скорочує час, необхідний для формування документів, що забезпечують організацію навчального процесу.

Для процесу розробки системи застосовано апарат функціонально-вартісного аналізу, в ході чого було обчислено техніко-економічні показники різних варіантів реалізації системи та обрано найкращу альтернативу.

Подальше розширення системи має відбуватись у напрямку збільшення кількості нормативних документів, що генеруються додатком. Система розроблена таким чином, що додавання нового функціоналу є максимально спрощеним і не впливає на роботу реалізованих працюючих модулів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Головенкін В.П. Рекомендації щодо розробки навчальних планів / В. П. Головенкін. – К. : НТУУ «КПІ», 2012. – 23 с. – 250 прим.
2. Головенкін В.П. Рекомендації щодо розробки навчальних та робочих навчальних планів за новими напрямами підготовки бакалаврів / В. П. Головенкін, А. Д. Лемешко – К.: ІВЦ “Видавництво «Політехніка»”, 2007. – 24 с.
3. Методичні вказівки до виконання організаційно-економічного розділу дипломних проектів / Уклад. В.Є. Богданюк, К.В. Березовський, В.П. Пашин та ін. - К.: НТУУ "КПІ", 1999. - 66 с.
4. Обзор Gson - работаем с JSON в Java [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://www.javenue.info/post/gson-json-api.
5. Пашин В.П. Функционально-стоимостный анализ конструкторско-технологических решений. - К.: РДЭНТП «Знание» УССР, 1989. - 22 с.
6. Пашин В.П. Управление качеством изделий на основе функционально-стоимостного анализа. - К.: «Технология и организация производства», 1989. - №1. с. 17-19.
7. Apache POI [Електронний ресурс] – Режим доступу до ресурсу: https://poi.apache.org/
8. JPA 2 Annotations [Електронний ресурс] – Режим доступу до ресурсу: http://www.objectdb.com/api/java/jpa/annotations.
9. Servlet API [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/servlet-api.
10. Three-Tier Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.techopedia.com/definition/24649/three-tier-architecture.