ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Пояснювальна записка
до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: **«Розробка web-додатку «Система контролю контентом» мовою Java»**

Виконав: студент 5 курсу, групи ППЗ-52

спеціальності:

121 Інженерія програмного забезпечення

Павленко Віктор Едуардович

Керівник: Шевченко Світлана Миколаївна

Рецензент:

Нормоконтроль:

Київ 2021

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти – «Бакалавр»

Спеціальність – 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

Негоденко О.В.

_____2021року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Павленко Віктору Едуардовичу

1. Тема роботи: «Розробка web-додатку «Система контролю контентом» мовою Java»

Керівник роботи

Затверджені наказом вищого навчального закладу від __**12.03.21** _2021року **№__65 _**

2. Строк подання студентом роботи
_____1.06.2021_____

3. Вхідні дані роботи:

У дослідженні було використанно середовище розробки IDEA, також методи керуванням контетом на сторінках. Для опису модельних діаграм було використано UML. Для створення та мігрування база даних будо використанно mariaDb, та фреймворки з бібліотеками: spring, ehcache, hibernate, azure app.

4. Зміст розрахунково-пояснювальної записки

4.1 Аналіз та характеристики програмного продукту а також опис взаємодіґ компонентів та модулів.

4.2 Провести опис блок схеми по роботі різних бібліотек та фреймворків.

4.3 Здійснити вибір розробки середовища та використання актуальних мов серврних та кліенських частин.

4.4 Розробити проектні рішення по системі та її частнами.

4.5 Зробити опис розробного програмного додатку.

4.6 Розробити адміністративні маніпуляції дял користувача.

1.Титульний лист

2. Об'єкт, предмет та мета дослідження

3. Порівняльна характеристика аналогів

4.Архітектура програмного додатку

5. Графічний дизайн

6. Дата видачі завдання 19.04.21

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи до	Примітка
1.	Підбір науково-техічної літератури	19.04.21 – 25.04.21	
2.	Вимоги до встановленого ПО	25.04.21 – 26.04.21	
3.	Оцінка якості тестування до системи	27.04.21 – 30.04.21	
4.	Концепція та архітектура веб-додатку	30.04.21 – 05.05.21	
5.	Вступ, висновки, реферат	05.05.21 – 07.05.21	
6.	Розробка презентації	07.05.21 – 20.05.21	
7.	Здача роботи	01.06.21	

Студент _____Павленко Віктор Едуардович

Керівник роботи_____ Шевченко Світлана Миколаївна

РЕФЕРАТ

Обєкт сфери дослідження – програмне забезпечення для відтворення сайтів або контролювання контентом на веб-стронiках.

Предмет дослідження – це програмна веб-платформа яка дає можливість редагувати або створювати нові веб-сторінки за своїм смаком. Або використовувати вже готові шаблони для свого сайту.

Мета дослідження – забезпечення легкого застосування у розробці різних сфер сайтів та інтеграція до різних проектів з використовуванням його API.

Методи дослідження – методи створення контенту, методи контролювання доступу, методи контролювання версіями та загальною інформацією сторінок, адміністративні методи.

У роботі було створено веб-платформу яка спрощує редагування та створення різних сайтів. А також, дає можливість створення різних ролей для адміністрації сайту з різними правами. Додаток з легкістю може редагувати будь-який документ без втручання у програмний код, та роботи це у реальному часі. Програма була створена на сучасних технологіях, та з документованим API для покращеного використання його у різних проектах.

Проведено описання програмних засобів, среди розробки та бази даних. Реалізований додаток на мові Java та з використання різних фреймворків та інструментів, а саме Gradle,Spring(MVC, Data, jdbc, REST, AOP), Hibernate, Ehcache, Lombok, Js, Jquery, CSS, JSTl

Ключові слова: Java, Spring MVC, Spring Data, Spring Rest, Lombok, MariaDB, JS, JQuery (ES6), API, Ehcache. Gradle

ЗМІСТ

Вступ

Обєкт сфери дослідження – програмне забезпечення для відтворення сайтів або контролювання контентом на веб-строниках.

Предмет дослідження – це програмна веб-платформа яка дає можливість редагувати або створювати нові веб-сторінки за своїм смаком. Або використовувати вже готові шаблони для свого сайту.

Мета дослідження – забезпечення легкого застосування у розробці різних сфер сайтів та інтеграція до різних проектів з використовуванням його API.

Методи дослідження – методи створення контенту, методи контролювання доступу, методи контролювання версіями та загальною інформацією сторінок, адміністративні методи.

Сучасний світ побудований на інтернеті і зв’язаний на веб-сайтах. Бізнес-організації розробляють свої сайти для більш ефективної роботи та прибутку, тим паче під час пандемії коронавірусу. Тому використання та популярність CMS-систем дуже виросла.

Основним завданням такої системи є збір і об'єднання в єдине ціле, на основі ролей і завдань, різних джерел інформації. Ці джерела можуть бути доступні як всередині самої організації, так і поза її межами. До того ж дана система забезпечує можливість взаємодії різних співробітників, проектів і робочих груп з тими базами знань і даних, які були раніше створені, в такому вигляді і таким способом, щоб зробити процес пошуку і повторного використання максимально комфортним і звичним.

Системи бувають абсолютно різні. Деякі системи орієнтовані тільки на вирішення конкретних завдань (ведення блогів, інтернет магазини, форуми), інші є універсальними і надають розробником зручне середовище проектування і програмування для розробки різних програм. У сучасному світі є проблема зі створення проектів або сайтів за смаком клієнта, тому що багато організацій пропонують вже готові шаблони сайту.

Так це прискорює швидкість розробки бізнес сайту, але багатьом такий підхід не подобається, тому що потрібен унікальний продукт для конкурентоспроможності. Тому було вирішено створити веб-додаток CMS системи, яка буде допомагати створювати сайти за своїм смаком або використовувати та модернізувати вже готові шаблони під свій смак. Це з легкостю вирішіть всі проблеми сучасного веб програмування та суперечність між розробкою свого унікального продукту.

1 АНАЛІЗ ТА ХАРАКТЕРИСТИКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Актуальність

Популярність використання веб ресурсів на сьогодні виросло у десятки разів. Це пов’язано не тільки з розвиванням різних технологій, а також із пандемією у світі. Усі проекти та бізнеси повинні працювати не виходячи з дому, а тому більшість ресурсів продається або використовується в інтеренті та на веб платформах. Інформація має тенденцію до зростання, причому в геометричній прогресії - її обсяг в сучасному світі подвоюється приблизно за 5 років. До того ж таке швидке поширення даних поставило на чільне місце їх актуальність: хто перший опублікував, той і отримав основний потік читачів. Сайти стали складніше, і спосіб ручного внесення - з розміткою, абзацами, заголовками, ілюстраціями, займає великий час.

Якщо власнику сайту потрібно було внести зміни - наприклад додати банер або нову кнопку в меню сайту, то зробити це, скажімо, для 10 сторінок - було занадто багато, але ще реально. Але кількість сторінок сайту стало вимірюватися тисячами. Повторити одну і ту ж операцію тисячу разів? А якщо банер втратив актуальність і його потрібно зняти - ще тисячу разів? А потім ще потрібно підключитися по FTP і завантажити всі нові версії сторінок, це займає дуже багато часу та ресурсів з боку розробки та дуже дорого для обновлення тому що усі операції над сайтом повинен роботи розробник. Але давати розробнику задачі просто змінювати монолітну роботу не приведе до покращення бізнесу.

Тому для вирішення цих багатогранних та глобальних проблем було створення платформи або додатку, який при пару запитів змінював та вирішував проблеми.

Сучасні CMS дозволяють заощадити час і сили, роблячи необхідні налаштування через власну систему параметрів движка. Сторінки такого магазину

не містяться на сервері, а всі запити в пошукових системах обробляються безпосередньо, коли програма відправляє http-запит на сервер.

1.2 Порівняння аналогів

CMS для інтернет-магазину - це найважливіший інструмент для управління контентом на сайті. Під різні цілі існує безліч різних CMS, тому вибрати відповідну систему не так просто. Щоб не втратити гроші, час і сили на установку невідповідної платформи, необхідно заздалегідь вибрати оптимальний варіант. Є багато типів систем які мають свої вигоди та недолікі. Кожна система має свою технічну підтримку, а тому не може використовуватися де завгодно. Зрівняємо сім найпопулярніших систем:

WooComerce - одна з найпоширеніших систем для електронної торгівлі, яка дозволяє запустити магазин на базі движка WordPress. Стандартна функціональність плагіна розширюється завдяки доповненням, тому його можна використовувати для вирішення різних завдань - в тому числі для побудови великих магазинів. Встановлюючи WooComerce на WordPress, ви отримуєте в своє розпорядження все, що потрібно для управління магазином.

У стандартній комплектації плагін дозволяє створювати каталоги, формувати категорії для фільтрації при пошуку, налаштовувати онлайн-оплату, регулювати скидочную систему, вибирати способи доставки і т.д. І так ми бачимо великі мінуси цієї системи з завантаженням та налаштуванням. Потрібно мати при собі багато плагінів, бібліотек для використання окремого функціоналу, а тому це буде не зручно та складно у використанні. Але така система має у собі усі необхідні функції для комфортного налаштування.

Magento - движок з відкритим вихідним кодом, призначений для створення великих інтернет-магазинів. Його головні характеристики - функціональність і гнучкість. Magento підтримує управління декількома сайтами з одного інтерфейсу. Серед особливостей CMS також можна виділити встановлену систему поділу прав користувачів, корисну при роботі над сайтом в команді, і вбудований

конструктор сторінок. Він робить редагування контенту трохи більш простим, але для глибокої кастомізації движка все ще потрібні спеціальні навички і час на вивчення документації. Magento - вимогливий движок, якому потрібно надійний хостинг. Про це теж потрібно пам'ятати при виборі CMS для створення інтернет-магазину. Також є безкоштовна версія.

OpenCart - популярна CMS, тому на більшості хостингів вона розгортається з адміністративної панелі в один клік. Це істотно прискорює процес установки і створення онлайн-магазину. OpenCart безкоштовна CMS - її можна завантажити з вільної ліцензії з офіційного сайту, після чого отримувати оновлення. Єдині витрати, які доведеться понести, - це оренда домену та хостингу.

Головна особливість OpenCart - будова за принципом MVC, який передбачає поділ даних, інтерфейсу і логіки на три компонента. Їх можна кастомизировать окремо, що в умілих руках дає можливість змінювати движок як це потрібно веб-майстру.

Shop-Script - движок від конструктора Webasyst, який пропонує розміщення сайту на хмарному хостингу або покупку ліцензії для створення магазину на власному сервері. Це платне рішення, тому для роботи знадобиться досить серйозний бюджет. Shop-Script можна використовувати в двох режимах.

Перший - візуальне редагування, яке підходить для початківців веб-майстрів.

Другий - режим для розробників, що володіють навичками програмування.

У ньому доступний редактор коду, який дозволяє створювати нові блоки і вносити зміни в файли шаблону, використовуючи мови HTML, CSS і JavaScript.

Функції управління магазином представлені в візуальному режимі, тому для їх застосування не потрібні спеціальні навички. Движок пропонує всі стандартні можливості для додавання товарів, формування замовлень, залучення покупців і відстеження їх активності через різні звіти.

Joomla - універсальна CMS, яка підходить для створення багатосторінкових сайтів. Одне з головних переваг движка - масштабованість. Ви можете почати з невеликого проекту, а потім розширювати асортимент товарів.

Для додавання магазинної функціональності необхідно встановити один з ecommerce-модулів: HikaShop, J2Store, Eshop, VirtueMart. Joomla - SEO-дружня CMS. У власника сайту є доступ до всіх інструментів зовнішньої і внутрішньої оптимізації. Стандартні можливості движка дозволяють налаштувати індексацію, переадресацію, вказати опису та ключові слова. Для додавання додаткових можливостей є SEO-модулі, платні і безкоштовні.

Drupal - безкоштовна система з цікавою моделлю поширення. Є ядро, в якому містяться базові можливості движка. Однак для створення на його основі сайту необхідно додавання модулів, причому їх список залежить від типу проекту. Щоб трохи прискорити процес розробки, користувачі створюють тематичні збірки - наприклад, OpenStore для інтернет-магазинів. Це не єдине рішення, є й інші. Ви також можете зробити збірку самостійно, підключивши до ядру необхідні модулі і налаштувавши вручну їх роботу. При роботі з розширеннями у Drupal є одна особливість.

Модулі працюють в зв'язці, використовуючи функціональність один одного.

Drupal - повністю безкоштовна система. Модулі для розширення функціональності, в тому числі додавання інструментів управління продажами, теж здебільшого мають ліцензію на вільне поширення.

Складання також безкоштовні - їх можна скачати з офіційного сайту Drupal. Тому серед обов'язкових витрат залишається тільки оплата хостингу і домену.

1.3 Аналіз засобів програмної реалізації

Для розробки основного серверу програми було використана мова Java та різні фреймворки. Проект розробляться на операційній системі Ubuntu.20.10. Головна причина використання мови Java, це те що я маю досвід у розробці різних проектів з використання цієї мови.

А також Java дозволяє писати програми під будь-яку платформу, та ця мова є об'єктно-орієнтована, що також грає багато значення. За допомогою Java,

можливо писати будь які проекти різного напрямку, починаючи від консолі та використання форм закінчуючи веб-сайтами та різного жанру ігор.

Проект створювався у середовищі під назвою “IntelliJ Idea Ultimate” від компанії JetBrains. Це середовище має легкий інтерфейс у використанні та дуже великий функціонал для розробки різних додатків. Також середовище має плагіни для швидкого вивчення усього функціоналу та налаштувань.

Програмне забезпечення JetBrains IntelliJ IDEA - це провідна середовище швидкої розробки на мові Java. IntelliJ IDEA являє собою високотехнологічний комплекс тісно інтегрованих інструментів програмування, що включає інтелектуальний редактор вихідних текстів з розвиненими засобами автоматизації, потужні інструменти рефакторинга коду, вбудовану підтримку технологій J2EE, механізми інтеграції з середовищем тестування Ant / JUnit і системами управління версіями, унікальний інструмент оптимізації та перевірки коду Code Inspection, а також інноваційний візуальний конструктор графічних інтерфейсів.

Унікальні можливості JetBrains IntelliJ IDEA позбавляють програміста від вантажу рутинної роботи, допомагають своєчасно усунути помилки і підвищити якість коду, піднімаючи продуктивність розробника на нову висоту.

Повна версія IntelliJ Idea Ultimate підтримує багато функціоналу, а саме:

- розумне автодоповнення, інструменти для аналізу якості коду, зручна навігація, розширені рефакторингом і форматування для Java, Groovy, Scala, HTML, CSS, JavaScript, CoffeeScript, ActionScript, LESS, XML і багатьох інших мов;
- підтримка всіх популярних фреймворків і платформ, включаючи Java EE, Spring Framework, Grails, Play Framework, GWT, Struts, Node.js, AngularJS, Android, Flex, AIR Mobile і багатьох інших;
- інтеграція з серверами додатків, включаючи Tomcat, TomEE, GlassFish, JBoss, WebLogic, WebSphere, Geronimo, Resin, Jetty і Virgo;

- інструменти для роботи з базами даних і SQL файлами, включаючи зручний клієнт і редактор для схеми бази даних;
- інтеграція з комерційними системами управління версіями Perforce, Team Foundation Server, ClearCase, Visual SourceSafe;
- інструменти для запуску тестів і аналізу покриття коду, включаючи підтримку всіх популярних фреймворків для тестування.

Тому майже всі розробники мови використовують саме це середовище для створення свого проекту. Давайте розглянемо інструкцію як можно скачати IntelliJ Idea Ultimate. Це потрібно роботи тільки з офіційного сайту JetBrains. Але можно використовувати і сторонні ресурси але це не рекомендовано. Після встановлення середовища, ми повинні мати JDK для роботи с Java.

JDK – це бібліотека яка в собі необхідні компоненти для роботи, та має віртуальну машину яка має назву JVM, вона потрібна для компілювання коду до рівня розуміння процесором що ми робимо. Відкриваємо середовище та вибираємо основу зборки проекту maven, встановлюємо версію мови восьму та вибираємо архітектуру проекту як веб, це буде виглядати таким чином:

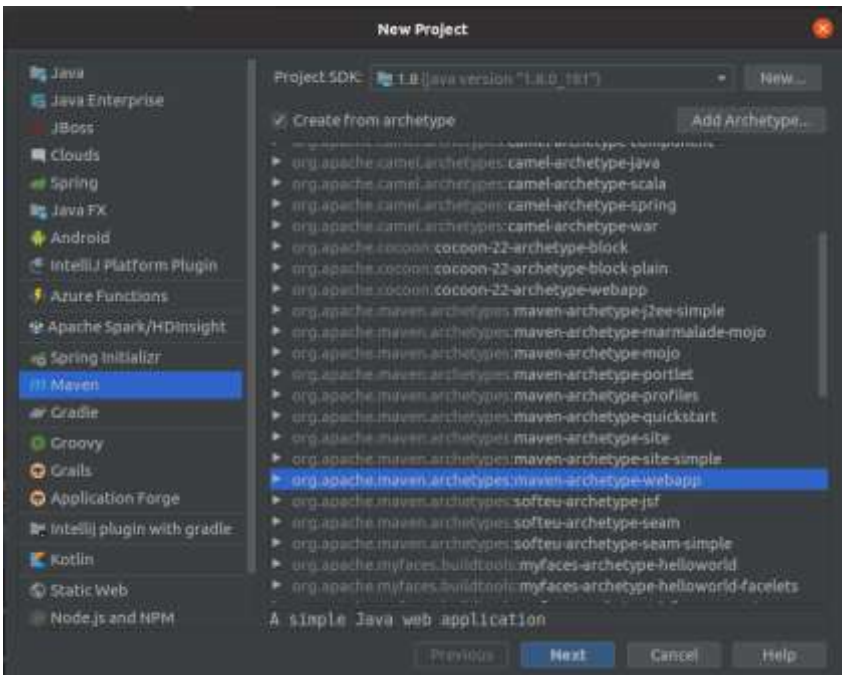


Рисунок 1.3.1 – Вибір архітектури

Далі пишемо назву проекту та визначаємо його розташування у нашому комп’ютері та нажимаємо продовжити:

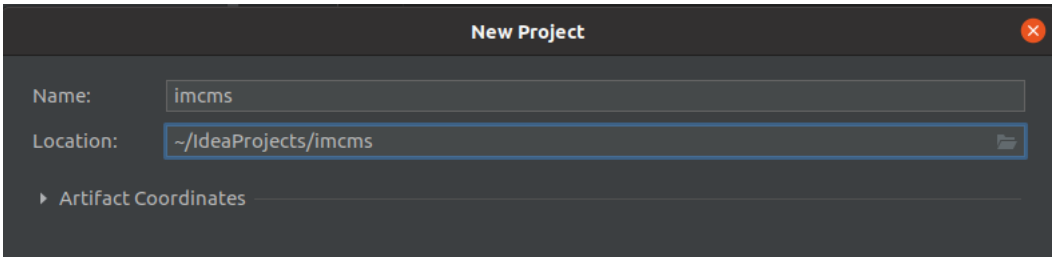


Рисунок 1.3.2 – Назва проекту

У наступному вікні ми можемо вибрати версію зборки проекту, але рекомендовано використовувати лише нову версію яка автоматично налаштується і буде мати такий вигляд:

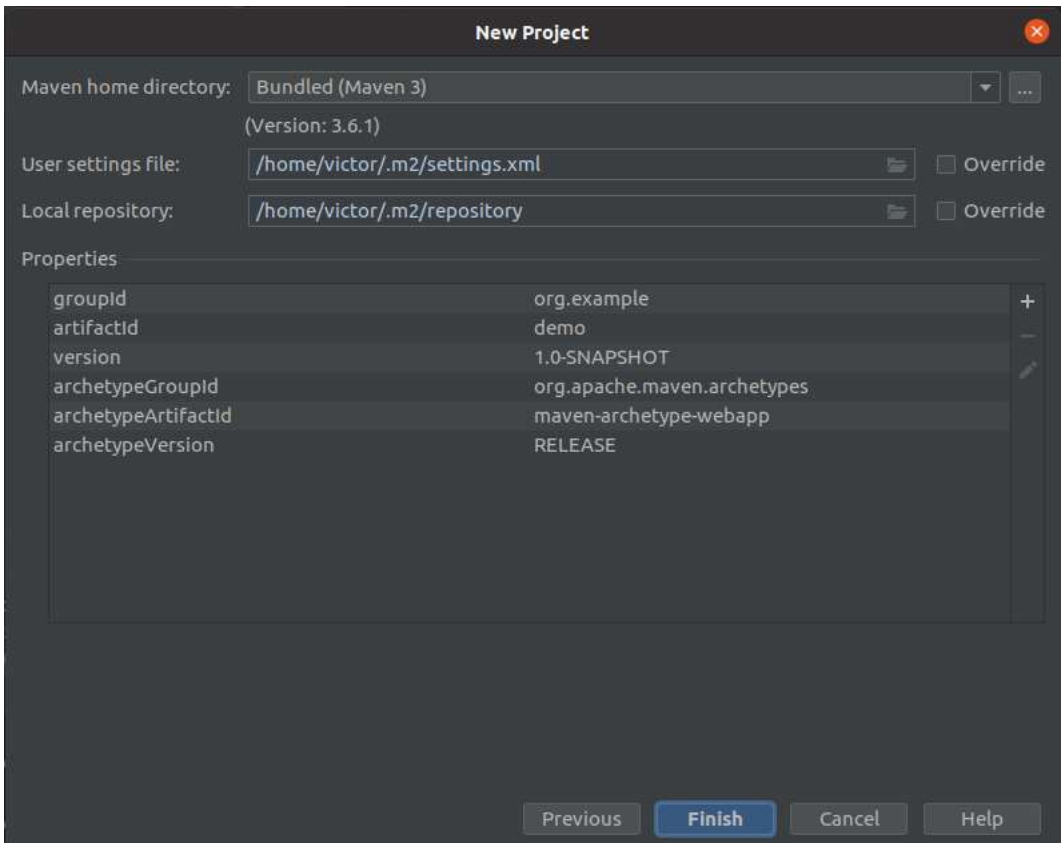


Рисунок 1.3.3 – Налаштування версії збірки проекту

Натискаємо finish та бачимо архітектору для розробки веб додатків:

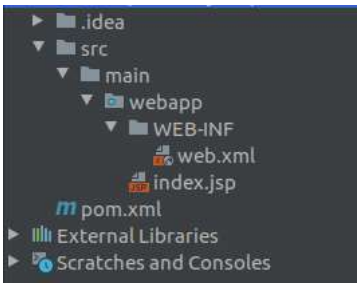


Рисунок 1.3.4 – Стартова архітектура

Для того щоб наш проект почав працювати на локальному сервері, потрібно створити та налаштувати локальній сервер, де ми будемо разгортати нашу прикладну програму. Для цього використовуємо Tomcat.

Tomcat - це контейнер сервлетів з відкритим вихідним кодом, який також виконує функцію веб-сервера.

Заходимо у панель додати конфігурації та вибираємо tomcat server local, бачимо вікно:

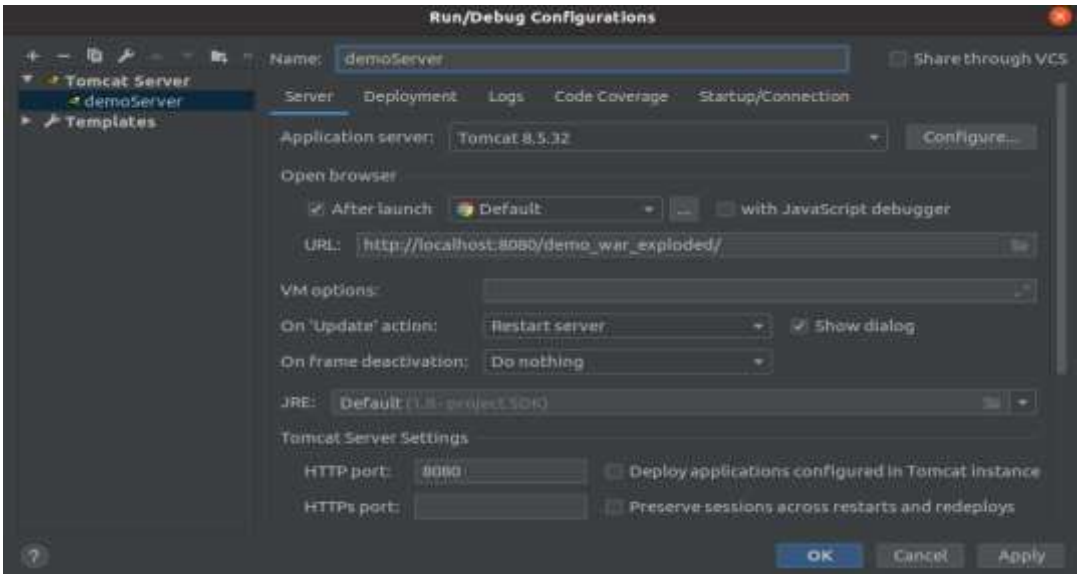


Рисунок 1.3.5 – Налаштування локального серверу

Встановлюємо ім’я нашого сервера, налаштовуємо версію, порти, браузер на якому буде автоматично відкриватися вікно. Також потрібно не забути вказати версію war файлу, яку ми будемо розгортати на нашому сервері. War файл має у собі усі конфігурації та файли для коректного роботи нашої прикладної програми. Як тільки закінчили усі налаштування натискаємо apply → ok. Тепер ми маємо налаштований веб-сервер для нашої прикладної програми. Та можемо запустити програму для тесту и побачимо стандартний напис “Hello World”. Запуск здійснюється за допомогою кнопки яка розташована на панелі

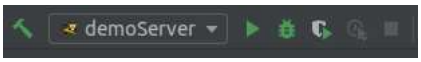


Рисунок 1.3.6 – Кнопки старту

При старті и успішного виконані програми ми повинні мати надпис у вікні серверу, що наш файл war успішно був розгорнутий на серверу, як це виглядає на рисунку нижче:

Рисунок 1.3.7 – Інформація успішного розгортання на сервері

Для розгортання проекту та підтягування усіх можливих залежностей використовується maven або gradle. Для більш масштабних проектів рекомендують використовувати gradle, так як у нього менше проблем с інтеграцією та налаштуванням на різних машинах. Gradle також дозволяє використовувати для проектів структуру каталогів, що відрізняється від конвенції Maven але має достатньо складну для розуміння підтягування різних залежностей, але він більш функціональний тому переводимо проект з maven на gradle. Архітектура не зміниться але замість pom.xml файлу ми будемо мати build.gradle, де и будемо встановлювати залежності.

Також середовище має додаткові налаштування для зборки усіх підключених компонентів та для очищення непотрібного ресурсу на проекті.

Один із головних та часто використовуваних функцій є clean, build and test. Розглянемо опис кожного із них. Clean - завдання очищення визначається плагіном java, і він просто видаляє папку buildDir, таким чином очищаючи все, включаючи залишки попередніх збірок, які більше не актуальні. Невиконання цієї вимоги може привести до нечистої збірці, яка може бути порушена через артефактів збірки, створених попередніми збірками. Test – завдання виконання тестування написаних тестів у підрозділі, автоматично запускає усі тести та чекає їх завершення. Build – завдання для зборки проекту, яка створює папку build де будуть знаходитись усі налаштування, конфігурації, логи, та кінцевий вигляд прикладної програми на сервері.

Кожна веб програма повинна мати сховище для зберігання даних на сервері. На сьогодні маємо багато різних бази даних для збереження інформації. Наш проект використовує базу даних Mysql. MySQL - вільна реляційна система управління базами даних. Ця реляційна система має зручний інтерфейс для використання та також дозволяє використовувати свій функціонал через терміновані функції. Треба встановити базу даних на останній версії та підключити її до нашого проекту. При установці бази даних ми будемо мати стандартного користувача який має рутові права. Не рекомендується використовувати цього користувача на різних проектах, так як він може заходити та міняти структуру бази даних без запрошення на підтвердження, тому це небезпечно! Потрібно власноруч створити користувача та надати йому належні права. Заходимо до терміналу під рутовим користувачем - «sudo mysql –u root –p», тепер ми знаходимось у середовищі бази даних. Для того щоб створити нового користувача та надати йому нові права потрібно виконати наступні команди:

- «CREATE USER 'imcms'@'%' IDENTIFIED BY 'imcms';»
- «GRANT insert, delete, create, update, select ON *.* TO 'imcms'@'%';»

База даних mysql має різні привілегії на права для користувачів тому потрібно розглянути кожного з них:

- All Privileges: Обліковий запис користувача має повний доступ до бази даних;
- Insert: Користувач може заповнювати стовпчики таблиць;
- Delete: Користувач може видаляти рядки з таблиць;
- Create: Користувач може створювати таблиці та базу даних;
- Drop: Користувач може видаляти таблиці та базу даних;
- Select: Користувач має доступ до зчитування та отримання данніх с таблиць та бази даних;
- Update: Користувач може обновлювати рядки таблиць;
- Grant Option: Користувач може обновлювати привілегії для інших користувачів.

І від тепер у проекті ми будемо використовувати нового користувача який має усі необхідні привілегії для роботи с базою даних та для нашої прикладної програми.

Тепер нам потрібно створити нову схему бази для збереження наших таблиць с даними. Тому переходимо до mysql терміналу та вводимо наступну команду, «create database imcms». Тобто зараз ми маємо схему де будемо створювати таблиці. І таким же способом ми створюємо тестову схему для тестів, «create database imcms_test». Тепер ми маємо дві робочі схеми та маємо змогу підключити ці схеми до нашого проекту. Для підключення, будемо використовувати файли properties, які будуть знаходитись під папкою resources-conf. Структура даних для збереження використовується як Map, а тобто ключ-значення. А тому у файлі буде міститись ключ-назва ресурсу та значення сам ресурс, як зображено знизу:

Рисунок 1.3.8 – Властивості підключення до БД

И тепер зараз ми практично налаштували підключення до бази даних. Але для того щоб воно почало працювати потрібно почати використовувати ці конфігурації. Для цього нам допоможе Spring фреймворк.

На проекті також використовуються багато різних фреймворків. Вони потрібні для зменшення написання коду та для зручності у підтримці великого коду. Також фреймворки полегшують життя розробнику, тому що не потрібно писати великі конфігурації для проекту які розташовані у різних файлах. Це у рази зменшує пошук та розуміння коду різними розробниками. Проект використовує на стороні сервера фреймворки Spring MVC, Spring Data JPA, Spring Jdbc, Spring AOP, Spring Rest, Hibernate, JUnit5. Розглянемо кожний модуль фреймворка окремо. Spring Jdbc и Spring Data JPA ці фремворки схожі між собою, вони виконують одну и ту функцію, але Spring data більш розвинутий та має свої великі переваги.

Іноді нам не потрібно генерувати під капотом великі запроси у базу даних та ми хочемо зробити свої запити до бази, у цьому нам допомогає Spring Jdbc. Він має різну імплементацію для роботи с різними запитами до бази даних. В різних ситуаціях це економить час роботи з базой даних. Spring Data JPA також потрібен для роботи с об’єктами для бази даних, але головна перевага є вже написані запити для використання даних. Тому не потрібно писати великі запити на отримання об’єкту с бази. Це все зробить фреймворк під капотом.

Але для початку треба створити конфігураційний клас під назвою DBConfig, та позначити його анотацією Configuration. Це дозволить фреймворку

зрозуміти що цей клас має у собі конфігурації для проекту і тому при старті почне проводити там аналіз. У класі ми повинні вказати папку де будуть знаходитись наші sql запити для мігрування бази даних, а також ми повинні вказати напрямок де будуть знаходитись файл для запуску по версіям запитів. Це необхідно для того, щоб коли ми будемо використовувати або інтегрувати нашу програму з іншими, то нам не потрібно буде кожен раз створювати нову базу з таблицями.

Тепер у класі починаємо робити підключення до бази даних для нашого файлу server.properties.

Для початку роботи з базою необхідно створити або мати хоча б одну таблицю. Таблиця це сутніть або певний клас. Так як ми використовуємо фреймворки то клас і таблиця повинні мати один і той самий вигляд. Тобто усі поля повинні мати один и той самий тип даних, зв’язування, назву, для того щоб Hibernate правильно під налаштував дані.

Hibernate - це ORM фреймворк для Java з відкритим вихідним кодом. Ця технологія є вкрай потужної і має високі показники продуктивності. Hibernate створює зв'язок між таблицями в базі даних і Java-класами і навпаки. Це позбавляє розробників від величезної кількості зайвої, рутинної роботи, в якій вкрай легко припуститися помилки і вкрай важко потім її знайти. Схематично це можна зобразити таким чином:

Рисунок 1.3.9 – Схема роботи Hiberante

Hibernate має у собі багато особливостей та переваги у використанні, а саме:

- забезпечує простий API для запису та отримання Java-об'єктів в / з БД;
- мінімізує доступ до БД, використовуючи стратегії fetching;
- не вимагає сервера додатки;
- дозволяє нам не працювати з типами даних мови SQL, а мати справу з звичний нам типами даних Java;
- піклується про створення зв'язків між Java-класами і таблицями БД за допомогою XML-файлів не вносячи зміни в програмний код;
- якщо нам необхідно змінити БД, то достатньо лише внести зміни в XML-файли.

Hibernate підтримує такі API, як JDBC, JNDI, JTA. JDBC забезпечує найпростіший рівень абстракції функціональності для реляційних БД. JTA і JNDI, в свою чергу, дозволяють Hibernate використовувати сервери додатків J2EE. Якщо ми розглянемо будову самого Hibernate більш детально, що цей же малюнок буде виглядати наступним чином:

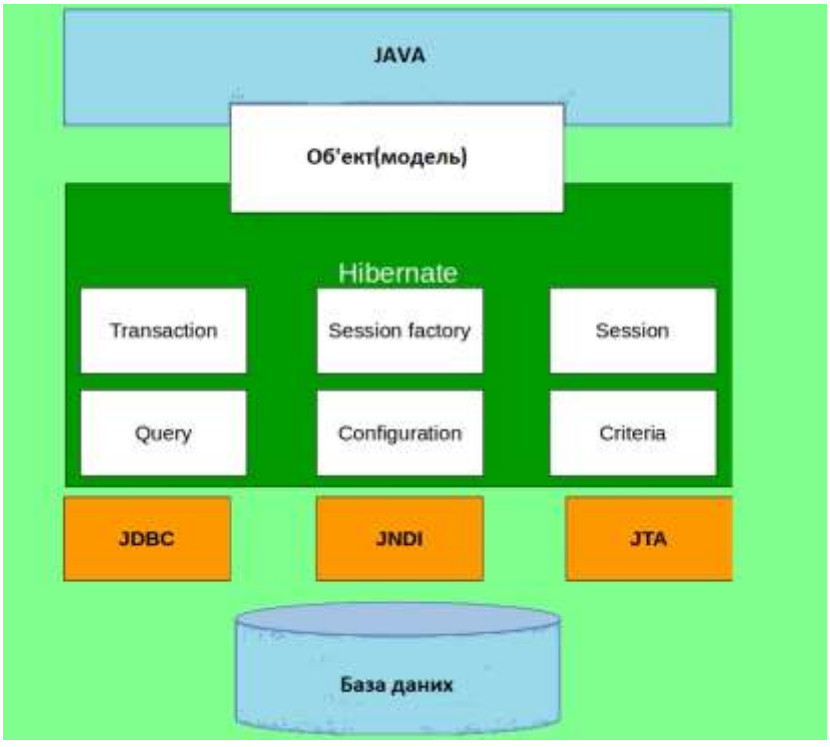


Рисунок 1.3.10 – Схема взаємодії конфігурації з БД

Transaction - цей об'єкт є робочу одиницю роботи з БД. У Hibernate транзакції обробляються менеджером транзакцій.

SessionFactory - найважливіший і найважчий об'єкт (зазвичай створюється в єдиному еземпляре, при запуску прілоденія). Нам необхідна як мінімум одна SessionFactory для кожної БД, кожен з яких налаштовується окремим конфігураційним файлом.

Session - сесія исползуется для отримання фізичного з'єднання з БД. Зазвичай, сесія створюється при необхідності, а після цього закривається. Це пов'язано з тим, що ці об'єкти вкрай легковажні. Щоб зрозуміти, що це таке, модно сказати, що створення, читання, зміна і видалення об'єктів відбувається через об'єкт Session.

Query - цей об'єкт використовує HQL або SQL для читання / запису даних з / в БД. Примірник запиту використовується для зв’язування параметром запиту, обмеження кількості результатів, які будуть повернуті і для виконання запиту.

Так як усі налаштування в БД були налаштовані ми можемо створити клас сутність для бази даних, та спочатку створемо таблицю. Все будемо робити у міграції, а тому створюємо файл sql, та маємо вигляд запиту як нижче:

```
DROP TABLE IF EXISTS `users`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `login_name` varchar(128) NOT NULL,
  `login_password` varchar(15) NOT NULL,
  `first_name` varchar(64) NOT NULL,
  `last_name` varchar(64) NOT NULL,
  `title` varchar(64) NOT NULL,
  `company` varchar(64) NOT NULL,
  `address` varchar(128) NOT NULL,
  `city` varchar(64) NOT NULL,
  `zip` varchar(64) NOT NULL,
  `country` varchar(64) NOT NULL,
  `county_council` varchar(128) NOT NULL,
  `email` varchar(128) NOT NULL,
  `external` int(11) NOT NULL,
  `active` int(11) NOT NULL DEFAULT '1',
  `create_date` datetime NOT NULL,
  `language` varchar(3) NOT NULL,
  `session_id` varchar(128) DEFAULT NULL,
  `remember_cd` varchar(40) DEFAULT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE KEY `UQ__users__login_name` (`login_name`),
  UNIQUE KEY `users_remember_cd_unq` (`remember_cd`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
```

Рисунок 1.3.11 – Скрипт створення користувача

Тепер ми можемо створювати сутність з назвою User для цієї таблиці. Але для початку краще створити абстракцію для цього класу, для того щоб ми використовували поліморфізм та абстрагувались від конкретного класу.

Створюємо абстрактний клас UserData. Та створюємо абстрактні методи для майбутніх успадкувань. Тепер коли ми маємо абстракцію, ми можемо створити сутність, та над шапкою класа ставимо анотації як Data, Entity, NoArgConstructor, Table(name=”users”),EqualsAndHashCode(callsuper=false), Cache. Тепер можемо описувати кожну колонку окремо. И тепер наша сутність має такий вигляд:

```java
@Data
@Entity
@NoArgsConstructor
@Table(name = "users")
@ToString(exclude = "password")
@EqualsAndHashCode(callSuper = false)
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class User extends UserData implements Serializable {

    private static final long serialVersionUID = 5707282362269284484L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Integer id;
    @NotNull
    @Column(name = "login_name")
    private String login;

    //And etc..
```

Рисунок 1.3.12 – Модель сутність користувача

Для того щоб витягнути користувача з бази даних нам потрібні запити, їх можна прописати власноруч або використати Data JPA та його реалізацію. Тому потрібно створити репозиторій, який буде мати назву UserRepository, це буде інтерфейс який успадковує JpaRepository. Spring data jpa підтримує написання власноруч запитів над методами, з використанням мови HQL або рідною мовою sql. Але мова HQL більш універсальна для java розробників, тому що можливо використовувати об’єкти напряму у запиті. HQL (Hibernate Query Language) - це об'єктів-орієнтований мова запитів, який вкрай схожий на SQL.

Відмінність між HQL і SQL полягає в тому, що SQL працює таблицями в базі даних (далі - БД) і їх столбацамі, а HQL - з зберігаються об'єктами (Persistent Objects) і їх полями (атрибутами класу). А тобто це зменшує написання коду та збільшує продуктивність розуміння коду. Реалізуємо методи пошуку користувача по його login name та додамо транзакцію до кожного методу, див.рис 1.3.13:

```
@Repository
@Transactional(Transactional.TxType.SUPPORTS)
public interface UserRepository extends JpaRepository<User, Integer> {

    User findByLogin(String login);

    User findByLoginAndActiveIsTrue(String login);

    User findByLoginIgnoreCase(String login);

    @Query("SELECT u FROM User u WHERE LOWER(u.email) = LOWER (?1)")
    List<User> findByEmail(String email);
```

Рисунок 1.3.13 – Методи пошуку користувача

Тепер ми маємо змогу реалізувати наш сервіс для основної бізнес логіки. Але все потрібно роботи на основі інтерфейсів тому, спочатку створюємо інтерфейс UserService з методами які були у нас в репозиторії, та імплементуємо цю реалізацію у класі DefaultUserService як зображено на рис 1.3.14:

```
@Override
public UserDTO getUser(String login) throws UserNotExistsException {
    return Optional.ofNullable(userRepository.findByLogin(login))
            .map(UserDTO::new)
            .orElseThrow(() -> new UserNotExistsException(login));
}
```

Рисунок 1.3.14 – Реалізація пошуку користувача

Після того як була створена основна логіка на сервісі та підключені усі біни, то необхідно реалізовувати контролер, для зв’язку сервера та клієнта. Для цього хорошу роль відіграє два основних фреймворки Spring Rest та Spring MVC.

Spring MVC - це веб-фреймворк Spring. Він дозволяє створювати веб-сайти або RESTful сервіси (наприклад, JSON / XML) і добре інтегрується в екосистему Spring, наприклад, він підтримує контролери та REST контролери в ваших Spring Boot додатках. При написанні веб-додатків на Java з використанням Spring або без нього (MVC / Boot) ви в основному маєте на увазі написання додатків, які повертають два різних формати даних:

- HTML → Ваше веб-додаток створює HTML-сторінки, які можна переглядати в браузері;
- JSON / XML → Ваше веб-додаток надає сервіси RESTful, які генерують JSON або XML. Сайти з великою кількістю Javascript або навіть інші веб-сервіси можуть потім використовувати дані, які надають ці послуги.
- Так, є і інші формати даних і варіанти використання, але поки ми їх ігноруємо.

RESTful сервіс повинен мати у собі контролер для перехоплення запиту з клієнта. Кожен контролер має свої точки доступу для розмови з сервером-клієнтом. Є різні точки для різного призначення:

1. Get – метод запиту використовується для отримання даних.
2. Post – метод запиту використовується для створення даних.
3. Put - метод запиту використовується для редагування даних.
4. Patch - метод запиту використовується для редагування/створення даних.
5. Delete - метод запиту використовується для видалення даних.

Реалізовуємо контролер який буде мати одну кінцеву точку с методом GET, для отримання даних користувача по його login. Для цього створюємо клас UserController , над шапкою класа встановлюємо дві основні анотації це RestController та RequestMapping. **RestController -** означає, що контролер надає послуги REST з типом відповіді JSON. **RequestMapping –** дає змогу назначити шлях доступу до цього контролера. Так як пошук буде здійснюватися за

допомогою login користувача потрібно додати параметр до метода через анотацію RequestParam.

Основна реалізація такого підходу буде базуватися на тому що усі запити будуть передаватися через протокол http/https, та відповідь буде відтворюватися у зручному форматі json. Json формат дуже легкий у використанні на стороні клієнта з використанням чистого javascript. Тому контролер потрібен для запиту від клієнта та відповіді від серверу, який контролює вхідні дані, шляхи до різних контролерів, та мінімалізує написання основної бізнес логіки. Схема роботи шаблону MVC повинна мати вигляд як вказано на рисунку нижче:

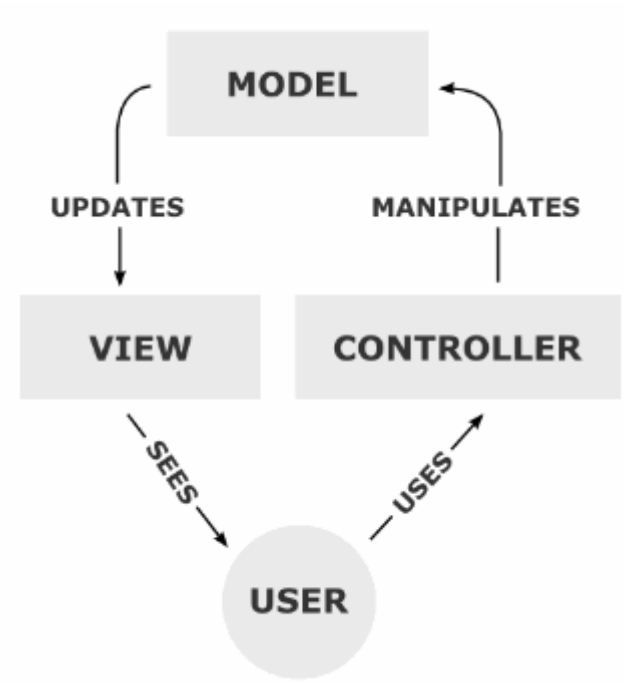


Рисунок 1.3.14 – Модель MVC

Так як усі компоненти створені налаштовані та підключені ми маємо змогу запустити наш сервер, та виконати перший запит для отримання даних о користувачі за допомогою **PostMan**. Завантажуємо програму через офіційного сайту, та вводимо там необхідні дані, для того щоб отримувати правильні обнови для цього продукту. Після того як завантаження та налаштування пройдуть нормально ми можемо запустити програму та будемо мати стартове вікно, як зображено нижче:

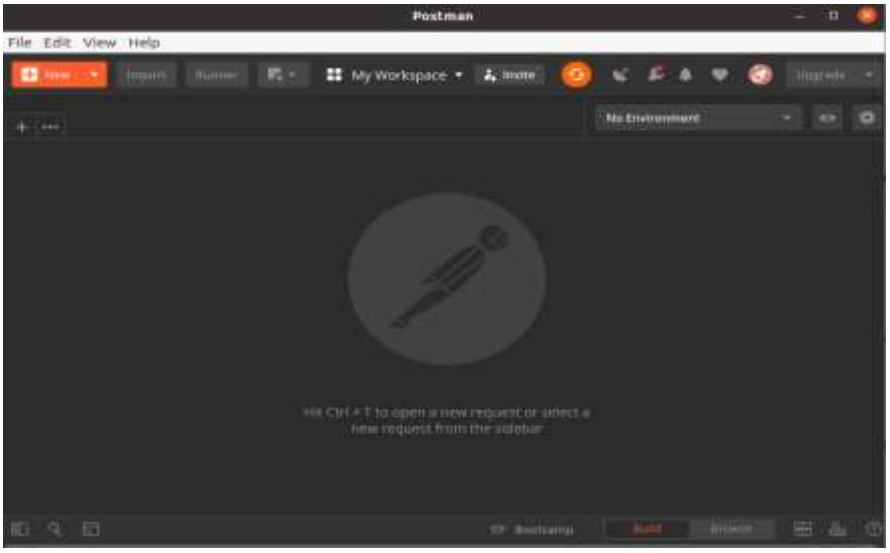
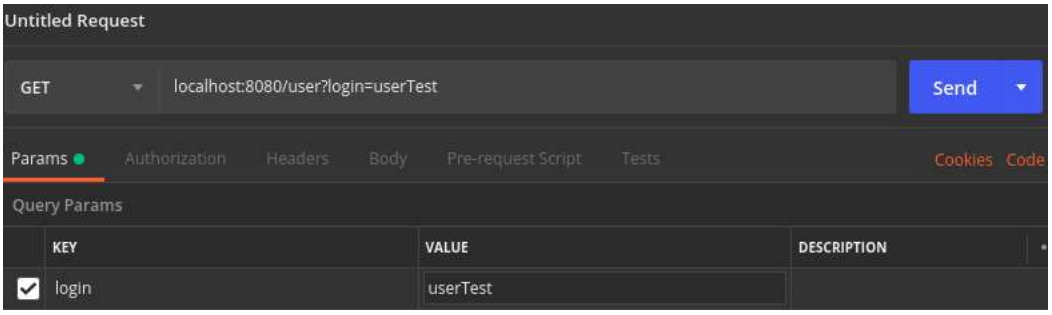


Рисунок 1.3.15 – Стартове вікно PostMan

Тепер ми маємо змогу налаштувати сам запит до сервера, для цього треба встановити метод запиту, у нашому випадку це GET, та прописати url до контролера з параметром та натиснути кнопку запит, це буде виглядати так:



1.3.16 – Запит до сервера

2 КОНЦЕПЦІЯ

2.1 Розробка концепції

Назва:

Програма має назву Imcms. На перший погляд вона здасться дуже не зрозуміло, але якщо почати розшифровувати, то можна побачити усю сутність назви. Перва частина це Im, що з англійської перекладається я є, та cms це і сам продукт. Тобто якщо перекласти на українську то назва має бути – «Я є система управління контентом».

Аудиторія:

Зазвичай такі системи використовують стартами, малі та середні бізнеси для швидкого розробки або покращення вже існуючого сайту. Система представляє багато різних вже готових шаблонів або дає можливість створювати свої. Але потрібно знати мінімально програмування для відтворення своїх шаблонів, тому основна аудиторія це не тільки просто користувачі, а і розробникі.

Маркетінг;

Проект рекламується різним блогам, презентаціям та від відгуків користувачів.

Маємо основний сайт де описується весь функціонал цієї програми. А тому не рідко він з’являється при пошуку в інтернеті, так як використовуються деякі посилання на нього.

Короткий опис програми:

Система розподілена та працює за різними ролями, у кожного користувача завжди є своя одна або декілька ролей у системі, яка дає змогу роботи різні маніпуляції в системі.

Опис роботи у системі як новий користувач:

- інтегруємо систему у наш проект, через залежності, вказуємо шлях та версію яку хочемо використовувати, або використовуємо вже веб версію;

- переходимо до реєстрації користувача, регіструємось;
- заходимо у систему під своїми даними;
- Якщо ми маємо дозволи на редагування, можемо створювати свої шаблони, або редагувати вже існуючі під свій смак;
- Затверджуємо свої зміни у шаблоні;
- Виходимо з системи;
- Бачимо результат нашого нового шаблона, для усіх користувачів.

Опис роботи у системі як користувач супер адміністратор:

- Переходимо до вікна входу у систему;
- Вводимо дані супер адміністратора;
- Робимо усі маніпуляції на сторінці(шаблоні), зміна тексту, картинок, и так далі;
- Супер адміністратор має можливість змінювати дозволи інших користувачів і повністю керувати системою;
- Маємо доступ до основної адміністративної панелі;
- І все що було написано у ролі простого користувача.

3 РОЗРОБКА АРХІТЕКТУРИ

3.1 Інструментальні засоби

IntelliJ IDEA - інтегроване середовище розробки Java додатків від компанії JetBrains. Її позиціонують як найрозумнішу і зручне середовище розробки для Java з підтримкою всіх останніх технологій і фреймворків. Хоч IntelliJ IDEA відома як середовище розробки для Java, в ній з коробки підтримуються кілька мов програмування.

Крім того, IntelliJ IDEA інтегрована з низкою сучасних фреймворків. У цю середу розробки вбудовані всі популярні системи контролю версій і системи збірки додатку. В IDEA реалізована підтримка багатьох серверів додатків. Починаючи з шостої версії, IntelliJ IDEA надає інтегрований інструментарій для розробки графічного інтерфейсу користувача. У цього середовища розробки є потужні аналітичні можливості. Завдяки їм ця IDE на льоту підказує розробникові кращі варіанти коду в поточному контексті. IDEA у своєму розпорядженні набір інструментів для рефакторінга існуючого коду і швидкого написання шаблонних конструкцій.

Перша версія IntelliJ IDEA з'явилася в січні 2001 року, і з тих пір компанія JetBrains доповнює своє дітище новими фичами і покращує існуючі. Починаючи з версії 9.0, IntelliJ IDEA доступна в двох варіантах:

- Community Edition;
- Ultimate Edition.

Community Edition - це вільна версія під ліцензією Apache 2.0. Вона призначена для JVM і Android розробки, а також додатків з GUI. Вона буде корисною як початківцям розробникам для освітніх цілей, так і професіоналам для комерційної розробки.

Ultimate Edition доступна під комерційною ліцензією, і в ній підтримується більше інструментів по сравнеію з Community Edition. Ця версія програми

призначена для enterprise і web розробки. Вона корисна для backend- і frontend-розробників.

IntelliJ IDEA поставляється для трьох платформ: Windows, macOS, Linux. Актуальну версію продукту можна завантажити з офіційного сайту компанії JetBrains.

Нижче наведено таблицю з відмінностями між версіями Ultimate Edition і Community Edition.

Таблиця 3.1.1 - Різниця середовищ розробок

	Підтримується тільки в Ultimate Edition	**Підтримується в Community Edition і Ultimate Edition**
Підтримка мов	• JavaScript • TypeScript • SQL • CSS, LESS, Sass, Stylus • CoffeeScript • ActionScript • XSL, XPath • Ruby, JRuby (через плагін) • PHP (через плагін) • Go (через плагін)	• Java • Groovy • Kotlin • Scala (через плагин) • Python, Jython (через плагин) • Dart (через плагин) • Erlang (через плагин) • XML, JSON, YAML • AsciiDoc, Markdown
Підтримка фреймворків	• Spring (Spring MVC, Spring Boot, Spring Integration, Spring Security and others) • Java EE (JSF, JAX-RS, CDI, JPA, etc) • Grails • GWT, Vaadin • Play (через плагин) • Thymeleaf,	• Android (включает функциональность Android Studio) • Swing (incl. UI Designer) • JavaFX

	Freemarker, Velocity, Tapestry • Struts, AspectJ, JBoss Seam, OSGI • React • AngularJS (через плагин) • Node.js (через плагин) • Apache Flex, Adobe AIR • Rails, Ruby Motion (через плагин) • Django, Flask, Pyramid (через плагин) • Drupal, Wordpress, Laravel (через плагин)	
Підтримка систем контролю версій:	• Team Foundation Server • Perforce	• Git, GitHub • Subversion • Mercurial • CVS
Підтримка інструментів розгортання:	• Tomcat • TomEE • Google App Engine and other clouds (через плагіни) • GlassFish • JBoss, WildFly • WebLogic • WebSphere, Liberty • Geronimo • Resin • Jetty • Virgo • Kubernetes (через плагін)	• Docker, Docker Compose
Підтримка систем збірки додатків:	• NPM (через плагін)	• Maven

	• Webpack • Gulp • Grunt	• Gradle • SBT • Ant • Gant • Ivy (через плагін)
Інше	• Database Tools • Diagrams (UML, Dependencies, и т.д.) • Dependency Structure Matrix • Detecting Duplicates • Settings synchronization via JetBrains Account • REST Client	• Darcula (темная тема) • Debugger • Decompiler • Bytecode Viewer • Unit Tests Runner (JUnit, TestNG, Spock; Cucumber, ScalaTest, spec2, etc) • Інтеграція с баг-трекінговими системами (YouTrack, JIRA, GitHub, TFS, Lighthouse, Pivotal Tracker, Redmine, Trac, і т.д)
Підтримка користувачів	• Підтримка 24\7	• Баг-трекинговая система і форуми

Дана IDE допомагає максимізувати ефективність розробника. Турбота про ергономіку середовища розробки простежується в кожному аспекті. . Кожен інструмент можна швидко відобразити або приховати:

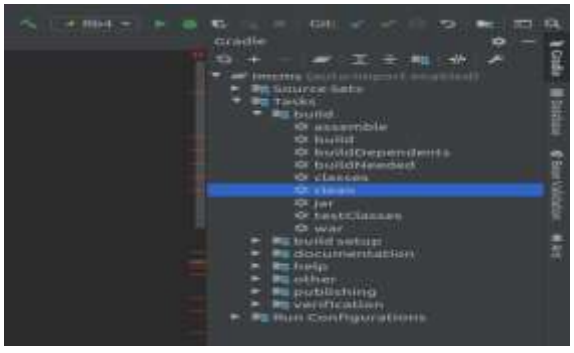


Рисунок 3.1.1 – Бокова панель збірки проекту

Інтерфейс середовища спроектований так, що більшу частину часу розробник бачить тільки редактор коду. Кнопки, що активують додаткові інструменти, розташовані на бічних і нижньої панелях екрану

У IntelliJ IDEA практично кожна дія можна виконати через певне поєднання клавіш. Розробник може сам призначати нові і змінювати старі поєднання клавіш для частих дій. В інтерфейсі IntelliJ IDEA в кожній структурі дерева, списку або спливаючому вікні, будь це дерево проекту або ж вікно налаштувань середовища розробки, є навігація і пошук. Досить сфокусуватися на потрібному місці і почати вводити текст для пошуку:

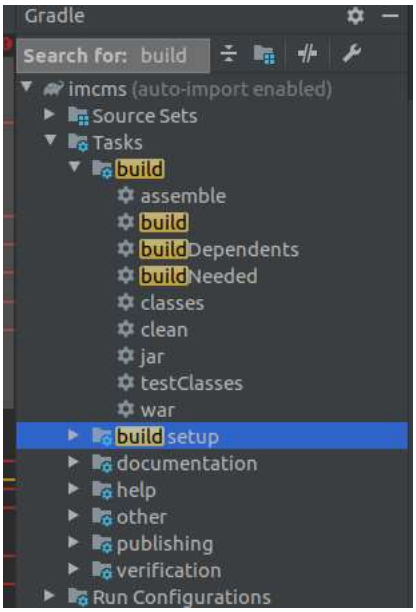


Рисунок 3.1.2 – Пошук по панелі

IntelliJ IDEA зручна при написанні коду і його налагодження. Дебагер IDEA показує значення змінних прямо в коді. І кожен раз, коли змінна змінює своє значення, вона підсвічується дебаггером:

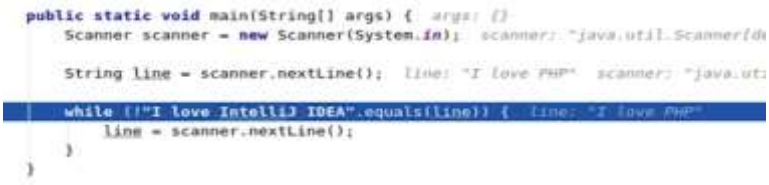


Рисунок 3.1.3 – Відстеження програмного коду

У IntelliJ IDEA є багато інструментів для роботи з кодом. Наведемо приклади деяких з них. Використовуючи інструмент Live Templates, розробник в рази скорочує час на написання часто використовуваних конструкцій коду.

Наприклад, для створення методу main досить набрати в редакторі psvm і натиснути клавішу tab або enter:

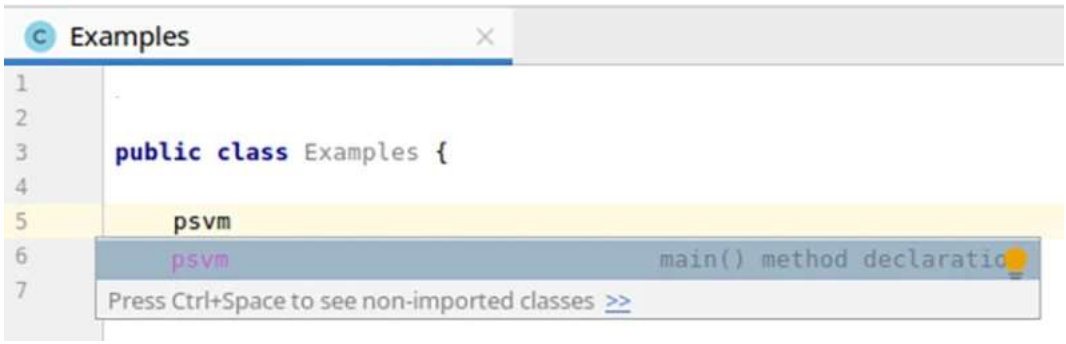


Рисунок 3.1.4 – Короткий запис даних

Нажимаємо tab або enter та бачимо:

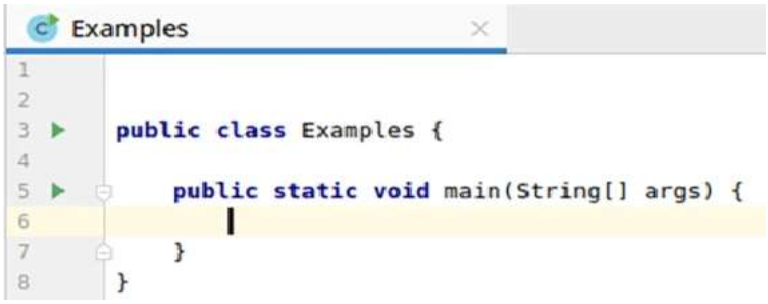


Рисунок 3.1.5 – Підтвердження короткого запису

IntelliJ IDEA індексує і аналізує весь проект, завдяки чому в будь-який час вона здатна моментально знаходити потрібні розробникові класи, методи і файли. В IDEA реалізовано кілька пошуків, серед яких є:

- Пошук класу на ім'я;
- Пошук файлу або директорії на ім'я;
- Пошук по проекту;
- Пошук по модулю;

- Пошук по директорії;
- Пошук по області, серед:
- файлів проекту;
- тестових файлів проекту;
- відкритих файлів;
- недавно переглянутих файлів;
- недавно змінених файлів.

Також завдяки індексації та аналізу всього проекту звичний для розробників автокомпліт стає на кілька рівнів розумніший.

Smart completion (Ctrl + Shift + Space) дає програмісту список найбільш релевантних варіантів коду, які можна застосувати до даного контексту:

Рисунок 3.1.6 – Підказки для реалізації

Chain Completion (Ctrl + Shift + Double Space) проводить глибший аналіз поточної ситуації та пропонує використовувати методи класів або змінних для поточного контексту:

Рисунок 3.1.7 – Розширені підказки

Функція Static Completion надає список статичних полів і методів можна застосувати в даному контексті.

Працюючи в IDEA, програмісту не потрібно думати про імпорт. Середовище розробки імпортує потрібні пакети і видаляє зі списку імпортованих пакетів непотрібні на льоту. Крім іншого, IntelliJ IDEA надає розробнику потужні інструменти для рефакторинга, щоб швидко реорганізовувати вихідний код програми. Все це - мала частина інструментів, які IntelliJ IDEA пропонує розробнику для роботи з кодом.

Та все це відноситься до плюсів IntelliJ IDEA. Однак, як і будь-який програмний продукт, у неї є і мінуси.

IntelliJ IDEA розробляється з 2001 року. У цього великого програмного продукту - велика кількість вихідного коду. Як наслідок, при роботі з IDEA можна наштовхнутися на баги.

IntelliJ IDEA вимоглива до ресурсів. За умовчанням вона виділяє до 512 Мб на x86 і до 768 Мб на x64. Але часом, наприклад, при великому рефакторінгу, навіть цього може бути недостатньо. Варто сказати, що ці значення можуть бути збільшені. Однак при цьому IDEA буде зжирає ще більше ресурсів системи. При роботі з великими файлами, наприклад, з класами в кілька тисяч рядків коду IDEA може помітно пригальмовувати. Компанія JetBrains регулярно випускає оновлення до IntelliJ IDEA. Дуже рідко, при оновленні IDEA, може щось зламатися.

На проекті також використовуються контейнер для клієнтської частини. Цей контейнер має назву **WebPack.** Webpack- це статичний модульний збирач для додатків на JavaScript.

Програми, написані на JavaScript, постійно ускладнюються, тому для збору модулів все частіше використовують спеціальний інструмент - Бандлер. Подібні інструменти дозволяють розробникам упаковувати, компілювати і в цілому організовувати всі ресурси, необхідні для проекту. Можна використовувати не тільки сторонні бібліотеки, а й власні файли.

Подібна модульна система дозволяє домогтися кращої організації проекту, так як він розбивається на невеликі модулі.Вебпак на даний момент є одним з найпотужніших подібних Бандлера, тобто модульний збирачів. Він має відкритий вихідний код і дозволяє вирішувати безліч завдань. Як і інші інструменти розробника, вебпак має свої плюси і мінуси. Принцип роботи вебпаку, зображено на рисунку нижче:

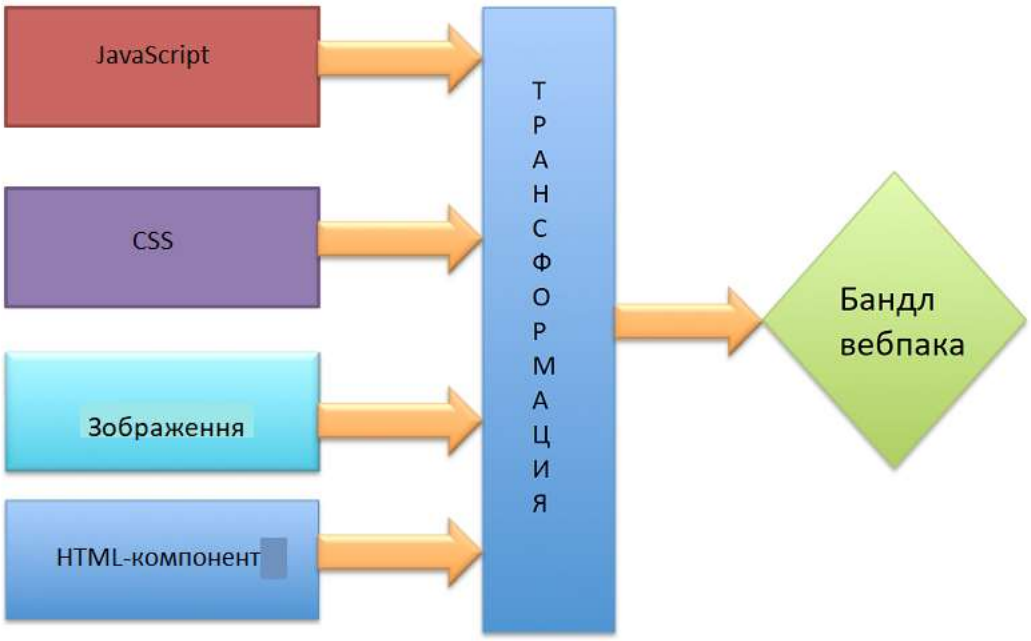


Рисунок 3.1.8 – Схема роботи WebPack

Почнемо з плюсів: він відмінно підходить для роботи з односторінкового додатками. Також вебпак може здійснювати просунуте поділ коду. Через ці та інші переваг він є одним з найбільш популярних інструментів JS-розробки на даний момент.

Мінуси: трохи складно розібратися в його роботі, частина документації застаріла через великої кількості змін в оновленнях.

Із бази даних використовуємо MySQL сервер. Це одна із популярних бази даних у світі. БД MySQL часто вибирають на проєти тому що вона:

- гнучка і нескладна у використанні. На створення і підтримку БД йде менше часу. Потрібно менший рівень компетенції для того, щоб повноцінно працювати з MySQL і реалізовувати весь її потенціал;
- має відкритий вихідний код, тому легко піддається модифікації, і за це не потрібно комусь платити;
- підтримується компанією Oracle і співтовариством розробників, які виступають за розвиток opensource-додатків;
- працює спритніше конкурентів. Внутрішня структура MySQL дозволяє їй розгрібати завали з таблиць і рядків за секунди. Незалежно від специфіки зв'язків між даними і їх кількості, сервер обробляє запити будь-якої складності швидше інших БД;
- стала загальною назвою і разом з цим певним стандартом в індустрії. Компанії шукають співробітників, які вміють працювати з MySQL, інтернет рясніє інструкціями по роботі якраз з MySQL-серверами;
- може похвалитися високим рівнем захисту даних завдяки системі видачі прав і просунутої системі управління користувачами. А ще тут є верифікація на базі хостингу і шифрування.

Захист даних забезпечується двома підсистемами: таблицею привілеїв і плагінами безпеки.

Перша потрібна, щоб захистити частину даних від певної групи користувачів. Вона змушує клієнтів, які роблять запити, авторизуватися в системі, щоб та могла переконатися в праві клієнта на отримання запитуваної інформації. Тобто управляти базою в повній мірі може обмежене число осіб. За бажанням можна продовжити певному колу осіб можливість вносити в таблиці будь-які зміни або видаляти з них дані, але при цьому залишити за ними можливість додавати нові одиниці даних в існуючу базу.

Модулі безпеки розширюють базові механізми захисту цілісності даних. Наприклад, створюють на сервері більш сувору політику створення паролів або додаткове сховище для конфіденційної інформації.

Недоліки MySQL має 4 основних у цій БД:

- MySQL не завжди поводиться стабільно. За даними популярного хостингу Digital Ocean, шведська СУБД зовсім не така надійна, як про неї говорять. Частина поширених завдань нерідко завершуються помилкою;
- вище я писав, що MySQL - продуктивна. Так це так. Навіть при роботі з великим об'ємом даних. Але не з великим об'ємом одночасно виконуваних завдань. При їх збільшенні спостерігаються помітні простої і уповільнення. Розробники відзначають, що СУБД поводиться куди більш слухняним і передбачувано в невеликих масштабах і при роботі з мінімальною кількістю операцій типу «запис / читання»;
- розвиток MySQL сповільнилося з тих пір, як її купила Oracle. Компанія не витрачає час і ресурси на розвиток придбаного продукту. При цьому патчі, пропоновані незалежними розробниками, відкидає;
- легкість системи в цілому досягається за рахунок мінімізації доступних за замовчуванням функцій. І навіть базові функції залежні від сторонніх розробок. Доводиться «наздоганяти» за рахунок установки розширень.

3.2 Основна архітектурна модель проекту

Найважливіший момент для початку створення програми є правильно побудувати архітектуру проекту. Тому що це відіграє велику роль у модернізації коду, та зменшує появлення різних багів. Потрібно притримуватися принципів ООП та SOLID. А тому весь проект повинен бути побудований на абстракціях, та бути максимально абстрагований до реалізації конкретних класів.

На першому етапі розробки почнемо з сторінки авторизації та реєстрації. А для цього потрібно мати налаштований WebPack, для подальшої легкої реалізації клієнтської сторони.

Після завантаження WebPack потрібно налаштувати вхідні та вихідні дані які будуть находитися у контейнері. Webpack - це збирач модулів JavaScript з відкритим вихідним кодом. Він створений в першу чергу для JavaScript, але може перетворювати зовнішні ресурси, такі як HTML, CSS і зображення, якщо включені відповідні завантажувачі. Програми, написані на JavaScript, постійно ускладнюються, тому для збору модулів все частіше використовують спеціальний інструмент - Бандлер. Подібні інструменти дозволяють розробникам упаковувати, компілювати і в цілому організовувати всі ресурси, необхідні для проекту. Для цього знаходимо файл webpack.config.js у проекті, та вводимо налаштування під проект як вказано нижче:

```
const merge = require('webpack-merge');
const baseConfig = require('./base.config.js');
const path = require('path');
const CleanWebpackPlugin = require('clean-webpack-

module.exports = merge(baseConfig, {
    mode: 'development',
    watch: true,
    output: {
        path: path.resolve(__dirname, './../../../
    },
    plugins: [
        new CleanWebpackPlugin(['./../../../build/
    ],
});
```

Рисунок 3.2.1 – Конфігурація WebPack

Після завершення налаштування вебпаку, необхідно створити шаблон для сторінок. Тому створюємо таку ієрархію папок, та де будуть зберігатися усі наші шаблони активності - /src/webapp/WEB-INF/jsp/imcms/view/. Під цим шляхом створюємо файл Login.jsp. Також необхідно підключити jstl теги для комфортної роботи на стороні клієнту, та для того щоб зменшити використання оригінального джава коду на сторінці клієнта. Заходимо в файл залежностей та підключаємо бібліотеку, через шлях до віддаленого репозиторію, як вказано нижче

Рисунок 3.2.2 – Підключення бібліотек

Як тільки ми підключаємо якусь залежність то відразу завантажується до проекту усі бібліотеки з методами у форматі jar. Тепер створюємо шаблон для аутентіфікація користувача до системи. Додаємо до шаблону два компонента текстового вводу для логина/пароля, та під ними розташовуємо дві кнопки. Перша кнопка з зеленим кольором буде підтверджувати авторизацію, друга синя кнопка буде для регістрації нового користувача у систему.

Але також система має різний мовний інтерфейс, тому необхідно додати кукі інформацію, для цієї сторінки. Тобто, система буде визначати на якій мові показувати сторінки. Тому створюємо два ресурсних файла, де будемо мати інформацію з різними мовами, файли з назвами imcms_en.properities/imcms_sv.properties. Розглянемо більш детально як працює авторизація кожного користувача на сторонні сервера:

- користувач вводить свої данні (логин та пароль);
- запит заходить у головний фільтр, де перевіряємо мову кукі даних, якщо немає то назначаємо дефолту мову яка вказано у системі;
- заходимо до сервлету, де перевіряємо отриманні дані з БД;
- проводиться перевірка є користувач закритий для цієї системи (якщо так, то переходимо до сторінки 403 помилки);
- сервер перевіряє чи нульовий час повторного вводу даних (якщо ні, то буде вивід на вікні авторизації текст з часом, скільки потрібно чекати на повторний вхід у систему);
- перевіряємо дозволи на сторінку;
- система назначає нову сесію для користувача та робить його авторизованим у системі.

Як тільки була створена сторінка авторизації, потрібно мати стартову сторінку, для переходу до неї при старті програми та при авторизації користувача

у систему. Тому необхідно створити новий шлях до основних текстових сторінок, а тобто шаблонів цього проекту. Файл буде мати назву demo.jsp та знаходитися у /src/webapp/WEB-INF/templates/text. Цей шлях буде основним шляхом де повинні зберігатися усі шаблони сторінок для системи.

Від тепер ми маємо стартову сторінку та авторизацію користувача в системі. Тепер необхідно зробити ролі для кожного користувача, їх дозволи по системі, та відтворити головну адміністративну панель де зможемо використовувати моди редагування, публікації змінення контенту, та перегляд змінень сторінки до публікації.

Створюємо клас RoleJPA та абстрактний клас Role. У проекті будуть використовуватися префікси як JPA та DTO. Кожний з них має свою роль між архітектурною взаємодію проекту. JPA класи їх ще називають сутностями, використовуються переважно у архітектурі між репозиторієм та сервісом, а dto класи використовуються у сервісах та контролерах. Тобто ці класи не завжди однакові тому що, деякі поля , або якась логіка не потрібні у БД, тому їх виносять від головної сутності та відображають вже на кінцевому рівні через контролер з використанням dto класів.

Приклад розглянемо пізніше , а тому переходимо до створення RoleJPA, RoleDTO класів. Для взаємодії основної моделі класів створюємо абстракцію ролі та сереалізуємо її для передачі даних по мережі. Також створюємо абстрактні гетери та сетери для наших полів, і створюємо стандартний конструктор для мапінгу об’єкту, основний абстрактний клас Role має вигляд як нижче:

Рисунок 3.2.3 – Модель абстрактної ролі

Тепер створюємо основну модель jpa класу. Додаємо над шапкою класу анотації для валідації та створюємо поля для цього класу, як id, name, RolePernissionsJPA.

RolePermissions необхідний для визначення дозволів для визначеної ролі. Ми будемо мати можливість створювати свою роль зі своїми дозволами, тому клас RoleJPA буду використовуватися у цій логіці. Роль сутність має вигляд:

```
@Entity
@Table(name = "roles")
@Data
@NoArgsConstructor
@EqualsAndHashCode(callSuper = false, exclude = "permissions")
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class RoleJPA extends Role {

    private static final long serialVersionUID = -602664810401741991

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "role_id")
    private Integer id;

    @Column(name = "role_name", nullable = false, unique = true)
    private String name;
```

Рисунок 3.2.4 – Модель сутності ролі

Також необхідно створити RoleDTO, для того щоб працювати з цим класом на підрівні з основною логікою та контролером. Тому створюємо папку dto, переходимо туди, та створюємо клас, з ідентичними полям як у сутності з виглядом нижче:

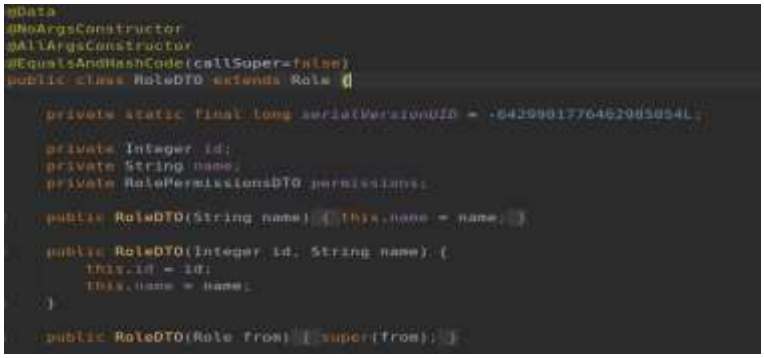


Рисунок 3.2.5 – Модель трансфер ролі

Зараз ми маємо роль і можемо назначити цю роль для кожного користувача, але ця роль ні де не зберігається тому необхідно створити таблицю ролей в БД.

Створюємо sql файл та добавляємо до файлу такий код:

```
DROP TABLE IF EXISTS `roles`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `roles` (
  `role_id` int(11) NOT NULL AUTO_INCREMENT,
  `role_name` varchar(60) NOT NULL,
  `permissions` int(11) NOT NULL DEFAULT '0',
  `admin_role` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`role_id`),
  UNIQUE KEY `UQ_roles__role_name` (`role_name`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;
```

Рисунок 3.2.6 – Скрипт створення таблиці ролей

3.2.1 Модель конфігурації

Для того щоб мати змогу працювати з кожнім компонентом спрінга, налаштовувати БД та взаємодію сутностей, а також мати можливість створювати свої beans, необхідно створити проектні конфігурації, які будуть знаходитися під папкою configs.

Нам потрібно вирішити питання с мігруванням кожного добавленого скріпту до різних проектів. Тому необхідно створити клас конфігурації, де будемо створювати та модернізовувати конфігурації роботи с БД. Створюємо папку configs/ та клас DBConfig. Додаємо до шапки класу анотації:

- @Configuration;
- @EnableTransactionManagement;
- @EnableJpaRepositories(basePackages={
  "com.imcode.imcms.persistence.repository"
  "com.imcode.imcms.mapping.jpa"}).

Анотація Configuration означає що клас являє конфиграціонима тому при старті спріг першу чергу заходить то конфігурацій та створює чи виконує

необхідні маніпуляції які вказані у цьому класі. Друга анотація означає що ми будемо використовувати на проекті транзакцію.

Транзакція забезпечує максимально цілісність даних при появлені помилок з базою або у програмі. Вона допомагає відкотити дані до початкового показу, якщо при маніпуляції з'явиться помилка. И третя анотація створена для уточнення спрінгу де знаходяться наші репозиторії та їх методи роботи с базою даних. Конфігураційний клас буде мати вигляд:

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(basePackages = {
        "com.imcode.imcms.persistence.repository",
        "com.imcode.imcms.mapping.jpa"
})
class DBConfig {
```

Рисунок 3.2.1.1 – Модель конфігурації БД

Під класом ми можемо створювати свої додаткові компоненти класу, та зв'язувати інші компоненти у цьому класі. В першу чергу створюємо bean який буде працювати с підключенням бази даних за допомогою серверних властивостей та налаштовуємо кеш конфігурацію. Бін має вигляд:

```
@Bean
public DataSource dataSource() {
    HikariConfig config = new HikariConfig();
    setImcmsDataSourceProperties(config);
    return new HikariDataSource(config);
}

@Bean
public DataSource dataSourceWithAutoCommit() {
    HikariConfig config = new HikariConfig();
    setImcmsDataSourceProperties(config);
    config.setAutoCommit(true);
    return new HikariDataSource(config);
}
```

Рисунок 3.2.1.2 – Біни для підключення до БД

Створюємо бін для генерування фабрики управління мігрування скірптів до БД, та версіями БД. А також налаштовуємо сканування різних папок де знаходяться усі сутності, які працюють з БД, та налаштовуємо Hibernate, кешування запитів, кешування сутностей, підключення діалекту та іншого, як зображено знизу:

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource dataSource) throws IOException {
    runSqlDiffs(dataSource);
    final LocalContainerEntityManagerFactoryBean entityManagerFactory = new LocalContainerEntityManagerFactoryBean();
    entityManagerFactory.setDataSource(dataSource);
    entityManagerFactory.setJpaVendorAdapter(new HibernateJpaVendorAdapter());
    entityManagerFactory.setJpaDialect(new HibernateJpaDialect());
    entityManagerFactory.setPackagesToScan("com.imcode.imcms.mapping.jpa", "com.imcode.imcms.persistence.entity");
    entityManagerFactory.setPersistenceUnitName("com.imcode.imcms");
    entityManagerFactory.setJpaPropertyMap(createHibernateJpaProperties());
    return entityManagerFactory;
}
```

Рисунок 3.2.1.3 – Бін менеджер налаштувань Hibernate

Також якщо ми використовуємо транзакцію, необхідно створити бін для цього, щоб можна було програмно мати управління транзакціями. Це у рази збільшує гнучкість управління та відкриває багато можливосте роботи з нею.

```
@Bean
public JpaTransactionManager transactionManager(EntityManagerFactory emf) {
    final JpaTransactionManager jpaTransactionManager = new JpaTransactionManager();
    jpaTransactionManager.setEntityManagerFactory(emf);
    return jpaTransactionManager;
}
```

Рисунок 3.2.1.4 – Тразакційний менеджер

Конфігурація БД через клас завершена, тепер необхідно створити конфігурацію для перетворення одного типу класу в інший. Назначаємо ім’я класу MappingConfig. Над шапкою класу ставимо анотацію конфігурації та додаємо логуваня помилок. Це дає можливість відстежити де відпала програма. Тепер маємо можливість створювати біни для перетворення, але це буде пізніше.

Необхідно підключити та налаштувати Web-MVC, тому створюємо конфігурацію під назвою WebConfig. Над шапкою класу додаємо анотації:

- Configuration;
- EnableWebMvc;
- EnableAspectAutoProxy;
- ComponentScan({…}).

Анотація EnableAspectAutoProxy включає аспектне програмування. Аспектно-орієнтоване програмування (АОП) - це парадигма програмування є подальшим розвитком процедурного та об'єктно-орієнтованого програмування (ООП). Ідея АОП полягає у виділенні так званої наскрізний функціональності. Схема роботи АОП виглядає так:

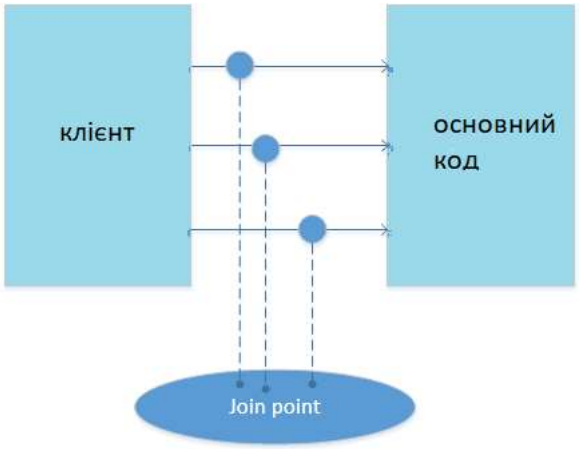


Рисунок 3.2.1.5 – Схема роботи аспектів

Основний клієнт робить запит до сервера, аспект перехоплює цей запит робить свої маніпуляції які прописані на сервері, та вже після всього допускає до основної бізнес логіки серверу. Такий принцип дуже схожий на фільтри. У веб конфігурації необхідно мати бін для показу всіх текстових шаблонів. Тому створюємо бін як показано нижче:

```
@Bean
public ViewResolver templateViewResolver() {
    InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
    viewResolver.setViewClass(JstlView.class);
    viewResolver.setPrefix("/WEB-INF/templates/text/");
    viewResolver.setSuffix(".jsp");
    viewResolver.setOrder(1);
    viewResolver.setExposedContextBeanNames("loopService", "imageService", "menuService", "textService");
    return viewResolver;
}
```

Рисунок 3.2.1.6 – Налаштування показу шаблонів

Необхідно мати основну конфігурацію проекту, де ми зможемо вказати , де знаходяться наші властивості, провалідіровать властивості з серверного файлу, та створити основні біни для роботи з сервесами. Тому переходимо до папки configs та створюємо там класс MainConfig. Над шапкою класу додаємо анотації:

- Configuration;
- PropertySource(value={/WEB-INF/conf/server.properties, classpath:test.server.properties});
- Import({DbConfig, ApplicationConfig, MappingConfig, WebConfig});
- ComponentScan({…}).

Кінцевий вигляд конфігурації повинен мати такий вигляд:

```
@Configuration
@PropertySource(value = {
        "/WEB-INF/conf/server.properties",
        "classpath:test.server.properties"},
        name = "imcms.properties", ignoreResourceNotFound = true)
@Import({
        DBConfig.class,
        ApplicationConfig.class,
        MappingConfig.class,
        WebConfig.class,
        CachingConfig.class
})
@ComponentScan({
        "com.imcode.imcms.domain",
        "com.imcode.imcms.mapping",
        "com.imcode.imcms.api",
        "imcode.util",
        "imcode.server",
        "com.imcode.imcms.components",
        "com.imcode.imcms.db",
})
public class MainConfig {
```

Рисунок 3.2.1.7 – Основна конфігурація проекту

Головною метою головної конфігурації є те що , необхідно об’єднати усі конфігурації в одне ціле, та мати контроль над ними. У анотації сканування ми вказуємо усі шляхи де знаходяться наші біни або компоненти які використовує спрінг. Та імпортуємо усі конфігурації до нашого осиного конфігу для маніпуляції над проектом.

Створюємо бін для головних властивостей нашого проекту, та робимо його вигляд як вказано на рис. 3.2.1.8:

```
@Bean
public Properties imcmsProperties(StandardEnvironment env,
                                  Validator<Properties> propertiesValidator,
                                  @Value("WEB-INF/solr") File defaultSolrFolder) {

    final Properties imcmsProperties = (Properties) env.getPropertySources().get("imcms.properties").getSource();
    final String solrHome = defaultSolrFolder.getAbsolutePath();
    imcmsProperties.setProperty("SolrHome", solrHome);

    propertiesValidator.validate(imcmsProperties);

    return imcmsProperties;
}
```

Рисунок 3.2.1.8 – Перевірка валідності властивостей

3.2.2 Архітектурна модель бізнес логіки серверу

Для того щоб створювати шаблони сторінок, необхідно мати теги контексту, такі як меню, картинки, тексти і т.д. Щоб була можливість додавати та створювати унікальні шаблони проекту під свій смак. Для цього необхідно створити папку web-inf/tags де буде створювати адміністративні теги. Створюємо файл під назвою menu.tag. та добавляємо основну логіку до відображення тексту з серверу. Для цього необхідно мати параметри для отримання даних з БД, це є індекс, документ (сторінка) на якому знаходиться тег. Також додамо додаткові властивості атрибутів та збільшемо функціональність нашого тегу. Меню тег буде додаткові атрибути label, showlabel, pre, post, attributes, treeKey, wrap. Label – це додаткова інформація над шапкою меню-редагування. Та showlabel має значення true/false для підтвердження показу додаткової інформації. Pre, Post атрибут має можливість загорнути наш редактор у написаний html-код. Та останні атрибути як

attributes, treeKey, wrap будуть грати важливу роль у автоматичній генерації html-коду з серверу, яка буде мати структуру списку. Атрибути необхідні та становляться у тезі з основним ресурсом, кожен атрибут буде відігравати роль зміни вихідних даних, який ми будемо получати з севреру. Кожен атрибут має вигляд у фалі таким чином:

```
<%@ attribute name="index" required="true" type="java.lang.Integer" %>
<%@ attribute name="document" required="false" type="java.lang.String" %>
<%@ attribute name="pre" required="false" type="java.lang.String" %>
<%@ attribute name="post" required="false" type="java.lang.String" %>
<%@ attribute name="attributes" required="false" type="java.lang.String" %>
<%@ attribute name="treeKey" required="false" type="java.lang.Integer" %>
<%@ attribute name="wrap" required="false" type="java.lang.String" %>
<%@ attribute name="label" required="false" %>
<%@ attribute name="showlabel" required="false" type="java.lang.Boolean" %>
```

Рисунок 3.2.2.1 – Атрибути меню тега

Для роботи з кодом серверу необхідно підключити його до файлу, тому додаємо властивості:

```
<%--@elvariable id="currentDocument" type="imcode.server.document.textdocument.TextDocumentDomainObject"--
<%--@elvariable id="isEditMode" type="boolean"--%>
<%--@elvariable id="isPreviewMode" type="boolean"--%>
<%--@elvariable id="targetDocId" type="java.lang.Integer"--%>
<%--@elvariable id="menuService" type="com.imcode.imcms.domain.service.MenuService"--%>
<%--@elvariable id="language" type="java.lang.String"--%>
<%--@elvariable id="editOptions" type="com.imcode.imcms.domain.dto.RestrictedPermissionDTO"--%>
<%--@elvariable id="isDocNew" type="boolean"--%>
<%--@elvariable id="disableExternal" type="java.lang.Boolean"--%>
```

Рисунок 3.2.2.2 – Властивості меню-тегу

Тепер ми маємо можливість вписувати бізнес логіку меню тегу та починати маніпуляцію с редактором. Додаємо умову коли тег буде використовувати автоматичну генерацію меню елементів як список, а коли буде створювати нашу стандарту схему. З отриманих меню елементів з серверу починаємо загортати в атрибути, які були раніше вказані, якщо їх немає продовжуємо працювати. Перевіряємо, на якій мові у нас поточний користувач та встановлюємо

локалізаційний текст. Вихідні дані готові для встановлення їх до основного класу меню редактора. Перевіряємо, в якому моді ми знаходимось та згідно з поточним модом відображаємо дані.

Для основного класу необхідно додати свої стилі, для того щоб мати унікальний вигляд контенту, потрібно додати у вигляді як вказано нижче:

```
.imcms-editor-area,
.imcms-editor-area * {
    border-radius: 0;
}

.imcms-editor-area {
    background-color: white;
    -webkit-box-shadow: 0 0 18px 1px rgba(0, 0, 0, 0.07);
    -moz-box-shadow: 0 0 18px 1px rgba(0, 0, 0, 0.07);
    box-shadow: 0 0 18px 1px rgba(0, 0, 0, 0.07);
    display: inline-block;
    padding: 10px 25px 10px 10px;
    position: relative;
    width: 100%;
}
```

Рисунок 3.2.2.3 - Стилі редактору меню

Основний меню тег готовий для використання. Але нам потрібно мати можливість і редагувати, додавати та видаляти меню елементи з редактора тому необхідно підключити js файли, та почати створювати основні маніпуляції там. Створюємо папки /js/buildres/windows/editors та створюємо декілька файлів js:

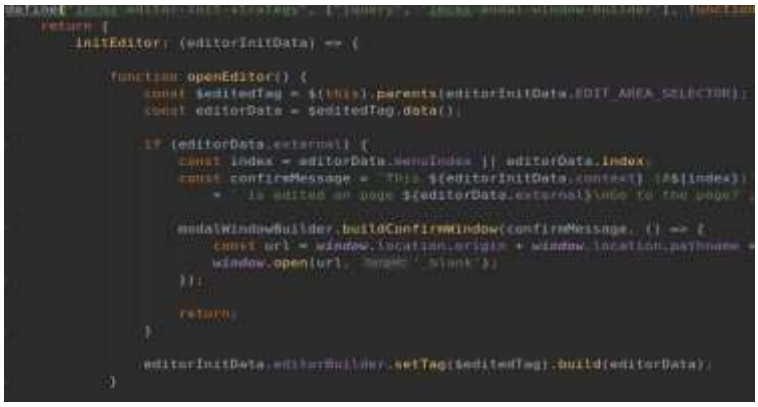


Рисунок 3.2.2.4 – Ініціалізація редактору

Імлементуємо основний код до одного компоненту та добавляємо вхідні параметри:

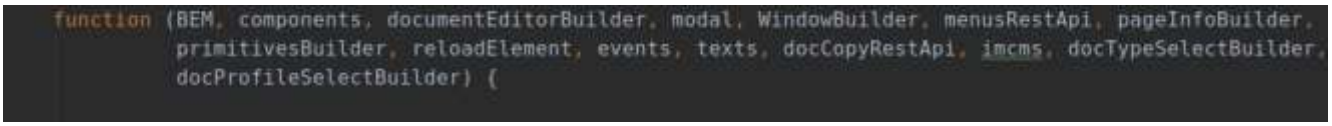


Рисунок 3.2.2.5 – Параметри редактору у функції

Тепер коли ми маємо стратегію імплементації, ми можемо створювати основний маніпуляційний файл для редагування вхідних даних у самому редакторі. Створюємо файл menu-editor-builder.js та підключаємо усі необхідні компоненти. Функція готова, тепер маємо змогу створювати маніпуляцію над елементами. Для того щоб отримати об’єкти з серверу необхідно зробити запит до серверу, для цього будемо використовувати ajax запити. Створюємо параметри для отримання даних з серверу та вставляємо їх до запиту, та з вихідними даними, заповнюємо або відобразимо їх у редакторі. Код повинен мати вигляд наступним чином:

```
function loadMenuEditorContent(opts) {
    addHeadData(opts);
    menusRestApi.read(opts)
        .done(menu => {
            opts.inMenu = true;
            opts.typeSort = menu.typeSort
            fillEditorContent(menu.menuItems, opts);
        })
        .fail(() => modal.buildErrorWindow(texts.error.loadFailed));
}
```

Рисунок 3.2.2.6 – Загрузка даних з сервера

Також додаємо рішення що до відображення помилок, тобто якщо з серверу відповідь не прийшла, то необхідно показати клієнту текстову пояснювальний текст у модальному вікні. Відображення даних та будова елементів у редакторі:

Рисунок 3.2.2.7 – Відбудова елементів у редакторі

Кожний елемент у меню редакторі повинен мати свої унікальні та необхідні дані. Ми повинні знати, чи має поточний текстовий документ нову версію для публікації, коли документ був створений, ким був створений, індетифікатор цього документа та інше. Тому прийнято рішення створити також сортування по цім полям, у редакторі. Спершу треба додати колонки та відобразити ці дані на клієнті, тому виконуємо код таким чином:

```
const $idColumnHead = $("<div>", {
    class: "imcms-grid-col-1",
    text: texts.id
});
$idColumnHead.modifiers = ["id"];

const $titleColumnHead = $("<div>", {
    class: "imcms-flex--flex-1",
    text: texts.docTitle
});

const $publishedDateHead = $("<div>", {
    class: "imcms-grid-col-3",
    text: texts.publishDate
});
$publishedDateHead.modifiers = ["date"];

const $modifiedDateHead = $("<div>", {
    class: "imcms-grid-col-3",
    text: texts.modifiedDate
});
$modifiedDateHead.modifiers = ["date"];
```

Рисунок 3.2.2.8 – Поля елементів меню

Так як меню має змогу сортуватися як по принципу дерева, то необхідно мати функціонал перетаскування елементів до інших елементів. Тому, потрібно створити початкові координати для кожного елемента, x, y. Добавляємо картинку та вішаємо на неї дію, що під кожне затискання кнопки на мишці ми можемо перетягувати весь об’єкт до іншого розташування. Все зроблено за принципом:

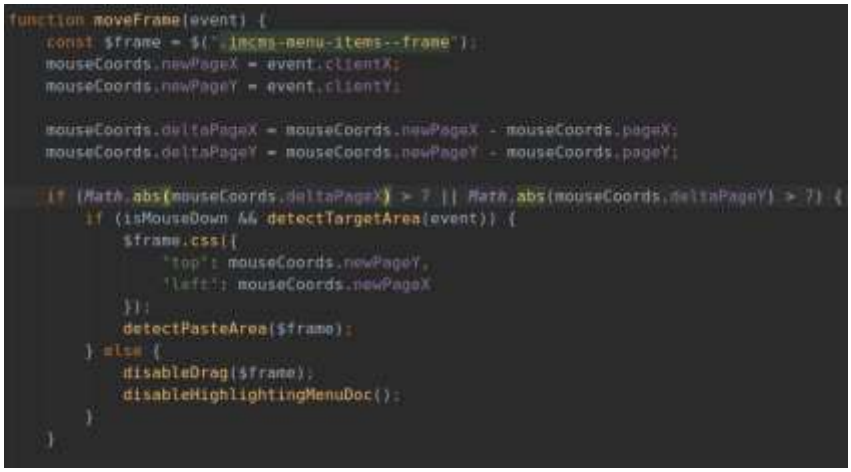


Рисунок 3.2.2.9 – Умови перетягування елементу

Після перетягування елемента, потрібно його розташувати у поставленому місці. Для цього зчитуються координати редактора та самого елемента, та після відпускання кнопки мишки, здійснюється паст до вказаної точки. Знаходимо клас який перетягуємо та починаємо считувати його координати, визначаємо є цей документ у списку меню та списку документів , які розподілені на два під розділі як було сказано вище. Тепер можемо додати до класу об’єкту нові класи для додавання стилів, та розпізнання цього об’єкту клієнтом. Визначаємо сортування, Визначаємо найвищу точку об’єкту, та меню редактора, змінюємо підрівень елементу, та додаємо цей елемент до або після другого елементу якщо такий існує, код має вигляд як нижче:

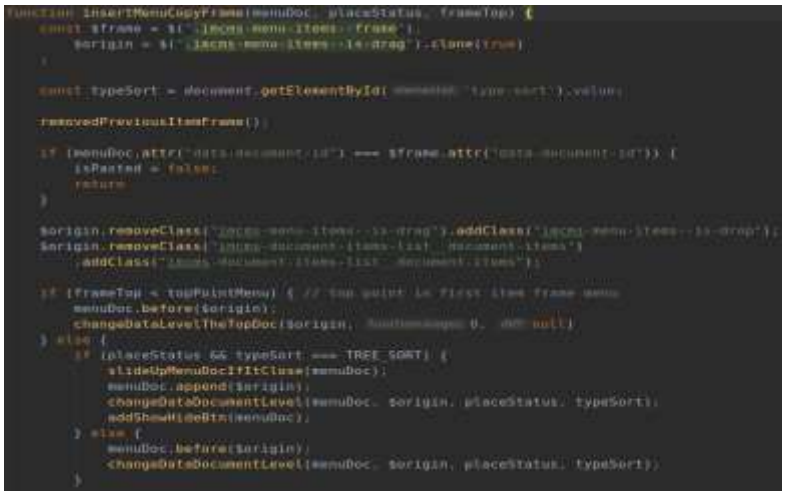


Рисунок 3.2.2.10 – Умови встановлення елементу у редакторі

Основної задачею меню редактора це маніпулювання та створення списку документів на сторінці. Для того щоб мати у собі список документів, необхідно додати до меню редактора ще одне вікно зі списком усіх створених документів. Розділяємо редактор на два незалежних вікна, та додаємо маніпуляцію над списками документів. До меню редактора також приєднуємо список сортування елементів редактора, та створюємо видалення елементів, по штучно або комплексним видаленням. По за вершенню клієнтського коду, маємо змогу перейти до основної бізнес логіки серверу. Для початку необхідно відтворити модель яка буде працювати с базою даних, як меню редактор, який будемо отримувати на сторінці у моді редагування, та його елементи. А тому сутність меню, буде мати у собі елементи з прив’язкою OneToMany. Меню сутность повинна мати у собі поле індетифікатора, колекцію з меню-елементів, та тип сортування. В той час також створюємо сутність для меню-елемента, та додаємо поля id, номер документа для якого воно створюється та число сортування. Тепер ми маємо модель для роботи с БД, і переходимо до реалізації методів репозиторію. Створюємо під папкою repositories клас MenuRepository, та додаємо методи що будуть отримувати дані з основого класу меню сутності. Реалізація має вигляд так:

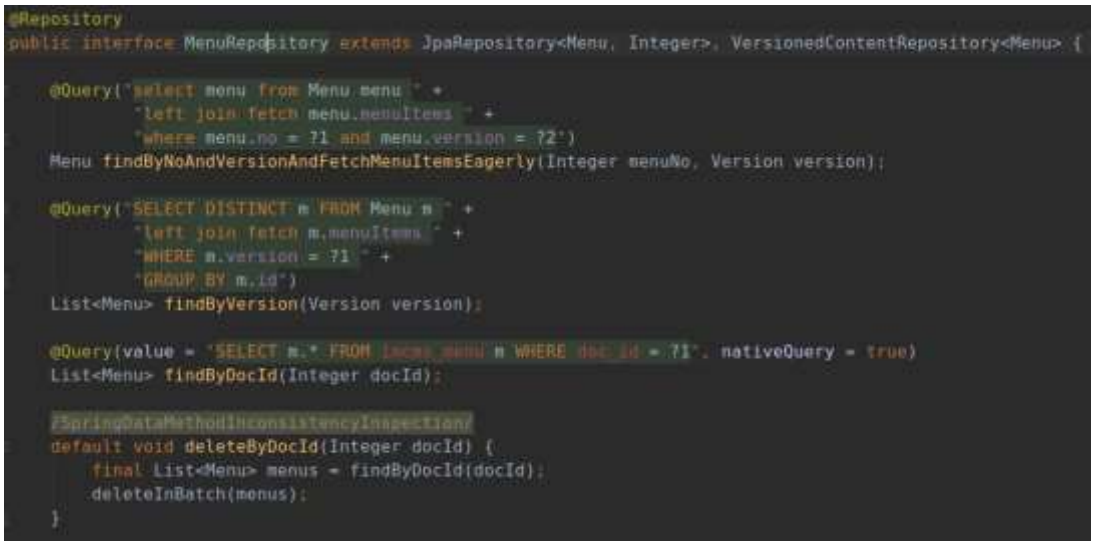


Рисунок 3.2.2.11 – Репозиторій меню запитів

Основна реалізація створена для роботи з БД, тепер необхідно відтворити сервіси, де буде проходити основа бізнес логіка. На сервісі, ми будемо отримувати дані з контролера та маніпулювати їм через різні методи і віддавати вихідні дані на клієнт. Реалізація меню інтерфейсу має вигляд як вказано нижче:

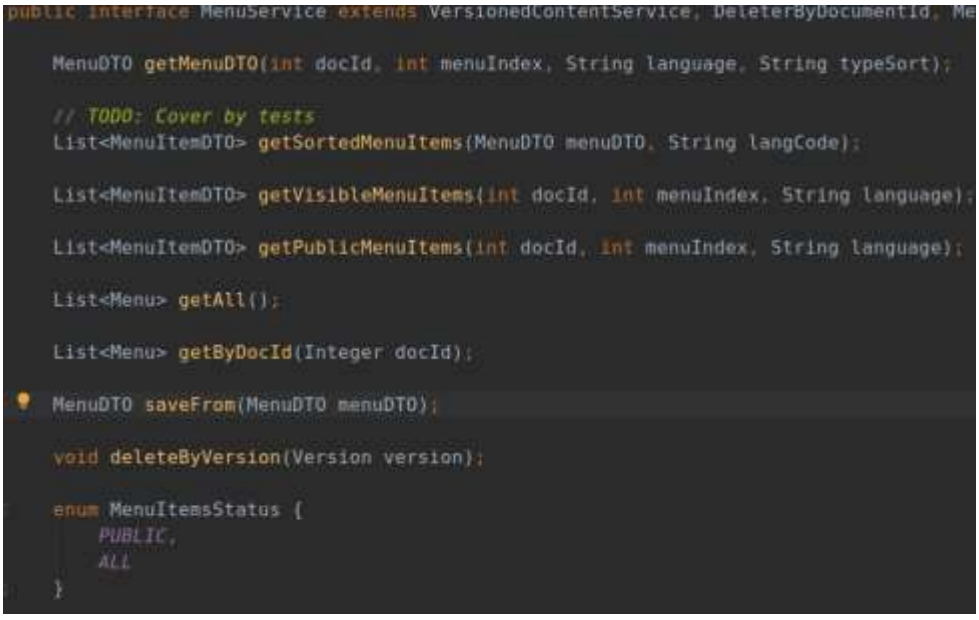


Рисунок 3.2.2.12 – Сервесні методи меню API

Після завершення реалізації сервісу, переходимо на створення контролера з кінцевими пунктами запиту, як get, post, put, виглядає це таким чином:

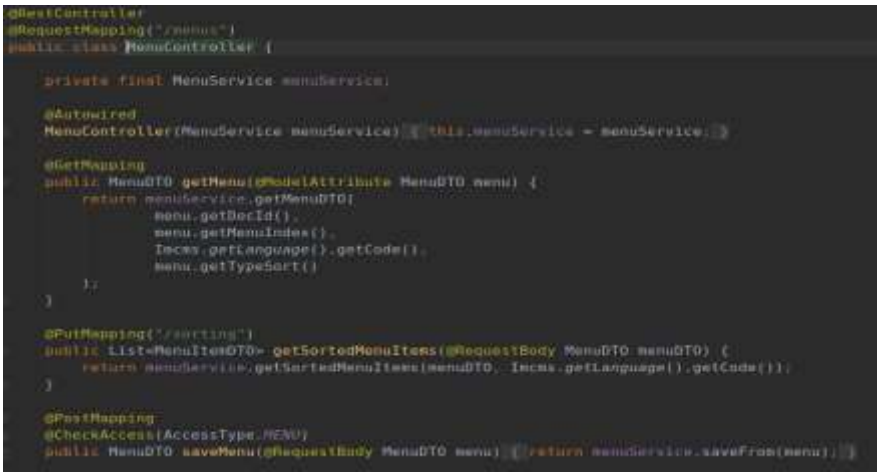


Рисунок 3.2.2.13 – Контролер запитів меню редактора

3.3 Графічний інтерфейс

Для створення унікального шаблону ми використовуємо теги нашого проекту, або маємо використовувати API, та напряму оформлювати собі сторінку. Але для цього необхідно повністю розуміти як побудована програма, та як працює сервер., тому рекомендовано використовувати вже зроблені теги на сервері. Заходимо у систему та вводимо свої дані у вказаних полях логін\пароль:

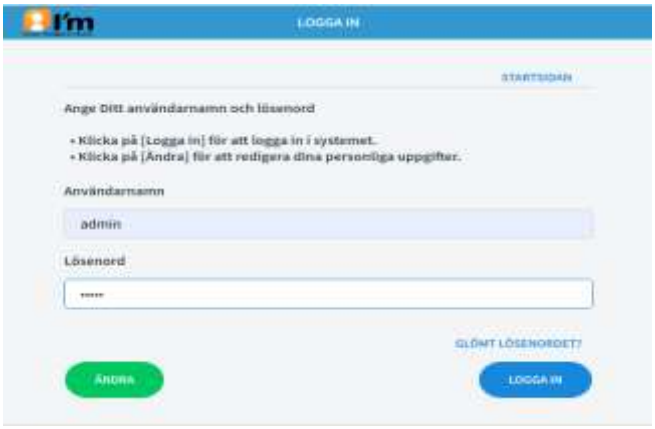


Рисунок 3.3.1 – Сторінка для входу у систему

Початкова сторінка шаблону після створення має пусте вікно без ніякого контексту, додаємо перші редактори. Додамо декілька картинок редактора з різним функціоналом для відображення можливостей редактору, також редактор меню та текстовий редактор. Після входу до системи ми переходимо автоматично до стартової сторінки. Якщо ми маємо права адміністратора, то ми можемо побачити адміністративну панель з різними модами та режимами, як зображено на рисунку 3.3.2:

Рисунок 3.3.2 – Адміністративна панель

Кожне вікно має свої особливості:

- Public – дозволяє бачити користувачам кінцевий результат публічного контенту;
- Edit – мод для редагування контенту на сторінці;
- Preview – дозволяє бачити користувачам який буде результат публічного контенту;
- Publish – опубліковуємо контент;
- Page-Info – дає інформацію о документі;
- Documents – показує список всіх документів;
- Admin – пере направлення до адміністративної сторінки;
- Logout – кнопка виходу з системи;
- Set-up* - кнопка для налаштування показу адміністративної панелі.

Тепер переходимо до edit моду та бачимо усі активні редактори на нашому текстовому документі. Приведемо приклад роботи з кожним редактором.

3.3.1 Меню редактор

Знаходимось на головній сторінці, та бачимо ікону у редакторі меню з права. Нажимаємо цю іконку, та переходимо до редактора меню. Редактор розподіленний на дві частини, на головну частину меню редактора, як зображено на рисунку 3.3.1.2:

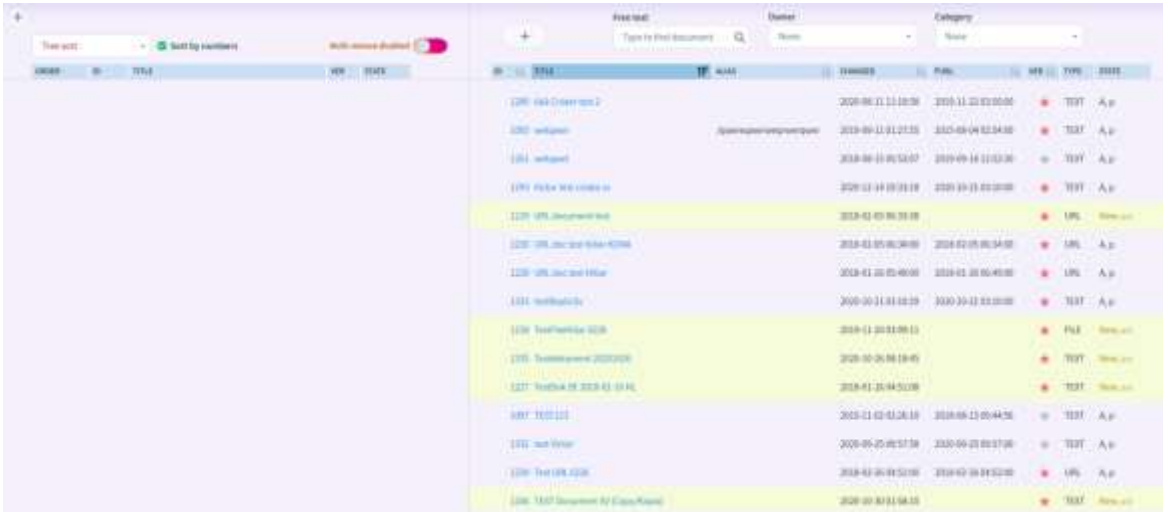


Рисунок 3.3.1.2 – Редактор меню

Кожний документ с права має іконку для перетаскування, затискаємо мишку та перетягуємо документи до меню. Тепер коли документ находиться в меню, ми бачимо що документ має затемнений колір, це означає що документ вже знаходиться у полі меню. Меню також має список сортування для елементів по їх полям. Меню підтримує сортування як:

- Tree Sort – дозволяє сортувати у вкладеному списку;
- Manual – дозволяє сортувати тільки в простому списку без вкладених рівнів;
- Alphabet (a-Z) – сортує по алфавіту від першої літери до останньої;
- Alphabet (Z-a) – сортує по алфавіту від останньої літери до першої;
- Published (new first) – сортує по першій упублікованій даті;
- Published (old first) – сортує по останній упублікованій даті;

- Changed (new first) – сортує по першій даті редагування документу;
- Changed (old first) - сортує по останній даті редагування документу.

У списку виділяється червоним кольором останній збережене сортування меню. Після вибору сортування як Tree Sort , біля кожного документа в меню ми маємо поля де потрібно вказувати порядковий номер сортування, при кожній помилці, редактор автоматично справить ці помилки. Ми також можемо перетискувати як було вказано вище, то кожен елемент буде знаходитись у вищій точці поля меню. Якщо ми хочемо видалити елемент з меню, ми можемо використати виделення яке зображення на кожному елементі як хрестик. Або скористатися мулті-видалення яке зображено у верхній частині редактору. При натисканні на копку цього видалення, активується видалення кожного елементу у меню. Тепер вибираємо кожний елемент який хочемо позбутися у редакторі. Результат буде зображений як на рисунку 3.3.1.3:

Рисунок 3.3.1.3 – Видалення усіх елементів з редактора

Зберігаємо, кінцевий результат меню та бачимо цей результат у моді редагування, як зображено нижче :

Рисунок 3.3.1.4 – Кінцевий вигляд меню

3.3.2 Редактор картинок

Нижче у документі ми можемо побачити редактор картинок з іконкою з права який має вигляд , як показано нижче:

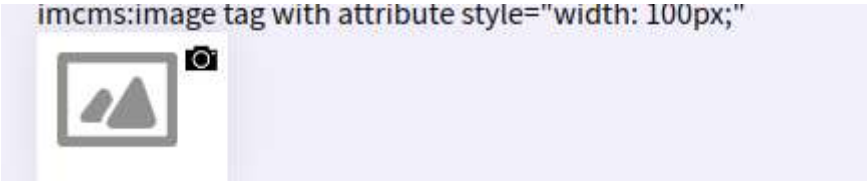


Рисунок 3.3.2.1 – Редактор картинки на сторінці

Натискаємо іконку та переходимо до редактору картинок, там маємо пусте поле редактора, та з права інформацію картинок, та кнопку бібліотеку збереження зображень, основний вид редактор маємо як зображено внизу:

Рисунок 3.3.2.2 – Редактор картинки всередині

Натискаємо на кнопку library та переходимо до сторінки де зберігаються усі зображення платформи. Ми також можемо створювати, свої папки де будемо зберігати картинки, або вибирати їх до редактора. Вибираємо картинку до редактора, та переходимо до основного редактору. Редактор картинок має свої функціональності, які можуть робити різні маніпуляції з картинок:

- Кнопка вкл/викл пропорцій зображення;
- Кнопки збільшення/зменшення зуму картинки;
- Відсоткове співвідношення до оригіналу;

- Активація редактору для перевертання картинок;
- Активація редактору для підрізання картинок;
- Кнопка повернення розмірів до оригіналу.

Також редактор має вікно з зображенням оригінальної картинки та його розміром. Тепер перейдемо до правої частини редактору де зображено додаткові функціонали, як показано внизу:

Рисунок 3.3.2.3 – Основна інформація картинки в редакторі

Перший синій блок показує інформацію о картинці, тобто її путь де вона знаходиться, скільки важить та оригінальні розміри. Редактор також може мати і другий блок , жовтий, він означає що редактор має назначені стилі для цієї картинки, або має якісь обмеження до розмірів. Останній блок який активується після натискання кнопки advanced показує нам можливість зберігати картинку у

різних форматах, та також дає змогу налаштувати на якому рівні у текстовому редакторі буде зображена картинка.

Сам блок має вигляд як нижче:

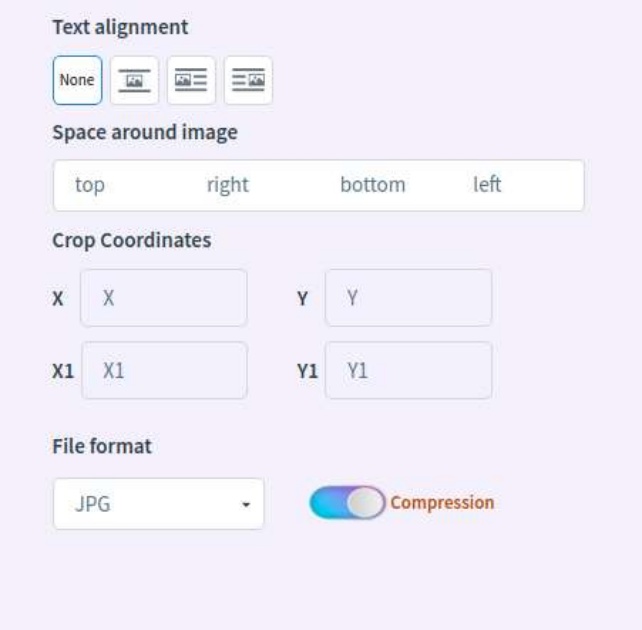


Рисунок 3.3.2.4 – Додаткові налаштування картинки

Зберігаємо та закриваємо редактор картинок, тепер бачимо цю картинку у текстовому документі:

Рисунок 3.3.2.4 – Кінцевий вигляд збереженої картинки

3.3.3 Редактор текстів

Документ містить у собі декілька текстових редакторів, але має різницю лише у функціональності кожного. Розглянемо редактор який найбільше функціональності з усіх. Кожен редактор має свій індетифікатор за допомогою якого програма впізнає оригінальність об’єкту на документі. Пустий редактор на сторінці має вигляд як зображено внизу:

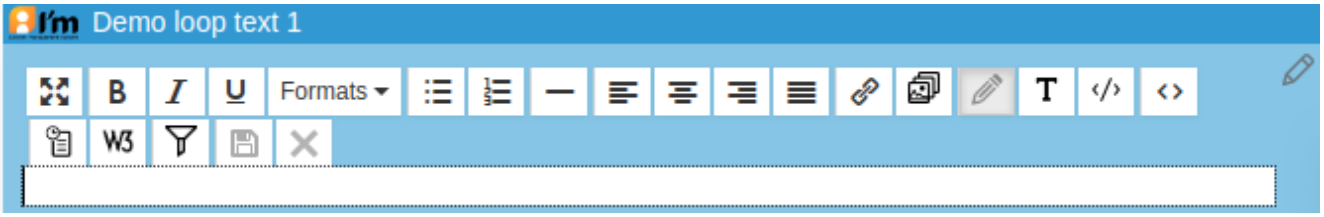


Рисунок 3.3.3.1 – Редактор текстів

Багато функціональний редактор має у особі багато дій а саме:

- Розгортання редактору до повного масштабу екрану;
- Позначення тексту за різним форматом (жирний, курсив);
- Налаштування формату заголовків;
- Додавання різних списків;
- Редагування вирівнювання за різними боками;
- Додавання посилання;
- Додавання картинки до тексту;
- Пошук текстової збереженої історії редактору.

Переходимо до редактора та додаємо туди текст, зберігаємо його за допомогою панелі зверху, та бачимо змінення на документі, як показано нижче:

Рисунок 3.3.3.2 – Запис у редакторі

Також редактор має попереджувальне вікно о зміненні тексту редактора:

Рисунок 3.3.3.3 – Вікно підтвердження збереження змін

Якщо ми заходимо до редактора, натискаємо там щось або змінюємо текст, та хочемо перейти до іншого контенту без збереження змін, то побачимо попередження з підтвердженням зберіганням змін. Тепер ми маємо сторінку з нашим редагуванням контенту, і тому можемо це зберегти та упубліковати наші зміни, для користувачів.

3.3.4 Адміністративна сторінка

Кожний адміністратор має змогу керувати платформою. Вони маю доступ до головної адміністративної сторінки.

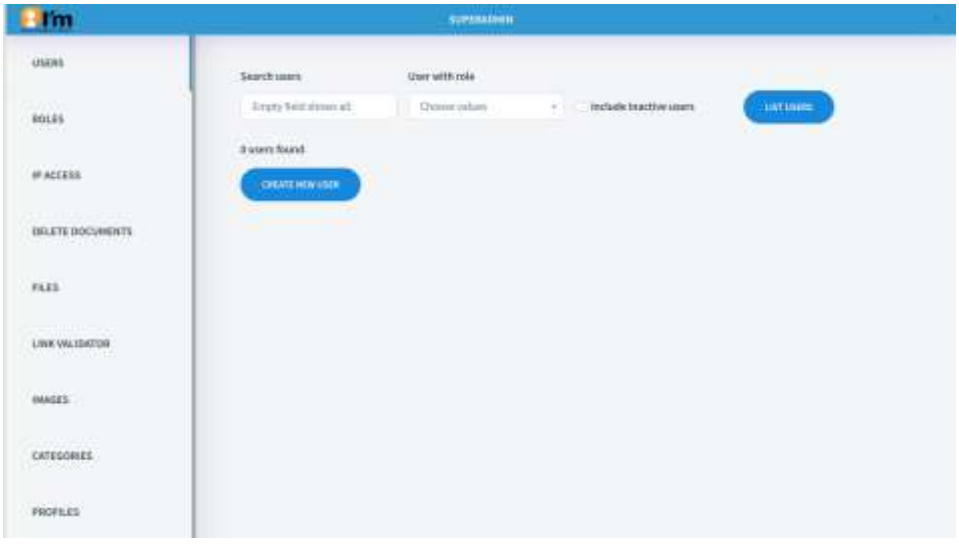


Рисунок 3.3.4.1 – Головна адміністративна сторінка

Ця вкладка доступна лише головним адміністраторам, якщо користувач не є таким, то ми бачимо вікно помилки , з текстом, немає доступу до сторінки. Так як ми зараз головний адміністратор то одразу ж перейшли до адміністративної панелі.

Бачимо різні вкладки, з різними функціями, які будуть описані нижче:

- Users – у цій вкладці мамо змогу бачити всіх зарегестрованих користувачів, та також можемо редагувати кожного або створювати нового;
- Roles – дає можливість список усіх ролей у системі, та створювати нові ролі;
- IP ACCESS – можна створювати обмеження доступу до системи по ip;
- Delete documents – дає змогу видаляти документи по індетифікатору;
- Files – можна створювати шаблони, та робити різні маніпуляції з файлами проекту;
- Link Validator – дає можливість показати стан кожного посилання на сторінках;
- Images – посилання на бібліотеку де знаходяться усі картинки;
- Categories – дає можливість створювати категорії та типи категорій, для документів;
- Profiles - дає можливість проглядати, або створювати нові профілі у системі.

ВИСНОВОК

Під час створення програмного продукту веб-додатку контролю контентом, було пройдено етапи, а саме:

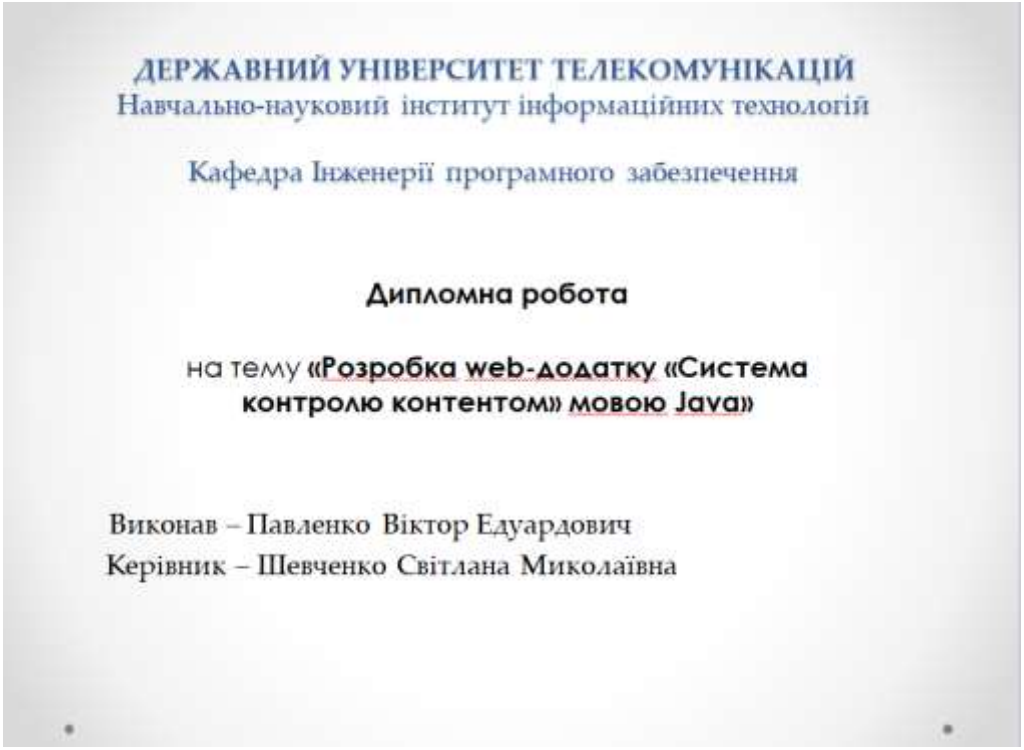
1. Було проаналізовано та охарактеризовано предметну область проекту та існуючі аналоги світового ринку. Видалено їх переваги та недоліки.
2. На етапі проектування програмного забезпечення був зроблений аналіз середи розробки, на якому робився проект, чому використовується саме вона, та її переваги.
3. Показано детальну конфігурацію проекту, та його можливості, як використовувати на проекті, з чи зв’язані ці конфігурації, та для чого вони.
4. Етап проектування розбитий на програмний код, та графічний інтерфейс, де роз’яснюється кожен контент редактор окремо.
5. На завершальному етапі було створено адміністративну сторінку з поясненням кожної вкладки.

Програмний веб-додаток управлінням контентом, дозволяє створювати іншим користувача свої сайти або шаблони для сайту, з використанням API продукту, та дозволяє використовувати вже готові шаблони для своїх цілей. З можливостями редагування та управління показу контенту для особливих користувачів. Також система має дуже гнучке управління користувачами, сторінками та файлами підсистеми, до якої доступ має лише особливий користувач з роллю як супер адміністратор.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

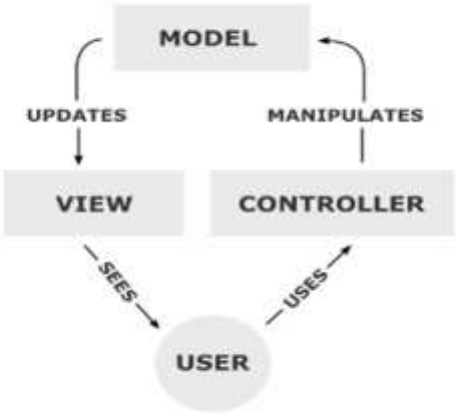
1. Spring annotations - [Електронний ресурс] – Режим доступу до ресурсу: https://www.baeldung.com/spring-bean-annotations.
2. Hibernate - [Електронний ресурс] – Режим доступу до ресурсу: https://proselyte.net/tutorials/hibernate-tutorial/.
3. Steam API Java 8 - [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/en/company/luxoft/blog/270383/.
4. Кей Хорстманн «Java 8.Тонкості програмування. Том 2 »Київ, 2017 р.
5. Патрік Ноутон «Java 8. Найбільш повне керівництво» Київ, 2017 р.
6. Програмування. Практикум / Укл.: Семенюк А.Д., Сопронюк Ф.О. – Чернівці: Рута, 2011.– 143 с.
7. Java EE - [Електронний ресурс]. — Режим доступу до ресурсу: https://goo.gl/qF4YDj.
8. Основные преимущества СУБД MySQL [Електронний ресурс]. — Режим доступу до ресурсу: https://goo.gl/M1vnGh.
9. SQL Server [Електронний ресурс]. — Режим доступу до ресурсу: https://goo.gl/5YOKZh.
10. Модель-вид-контролер [Електронний ресурс]. — Режим доступу до ресурсу: https://goo.gl/CVoK9U.

Додаток А


ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут інформаційних технологій
Кафедра Інженерії програмного забезпечення
Дипломна робота
на тему «Розробка web-додатку «Система контролю контентом» мовою Java»
Виконав – Павленко Віктор Едуардович
Керівник – Шевченко Світлана Миколаївна


Основні використанні програмні засоби
HIBERNATE
spring
Tomcat
MySQL
IJ
Java
EHCACHE
webpack

MVC Архітектура проекту



АОП. Схема аспектне орієнтованого програмування

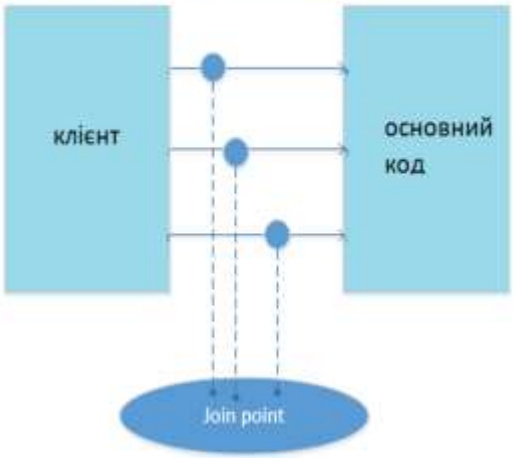


Схема взаємодії класів сутностей Hibernate та БД

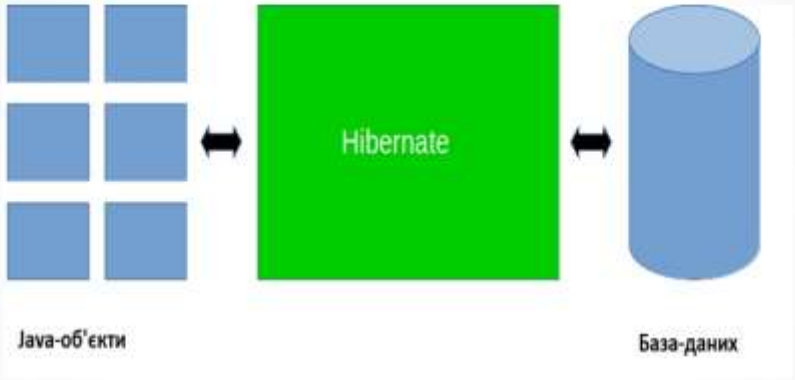
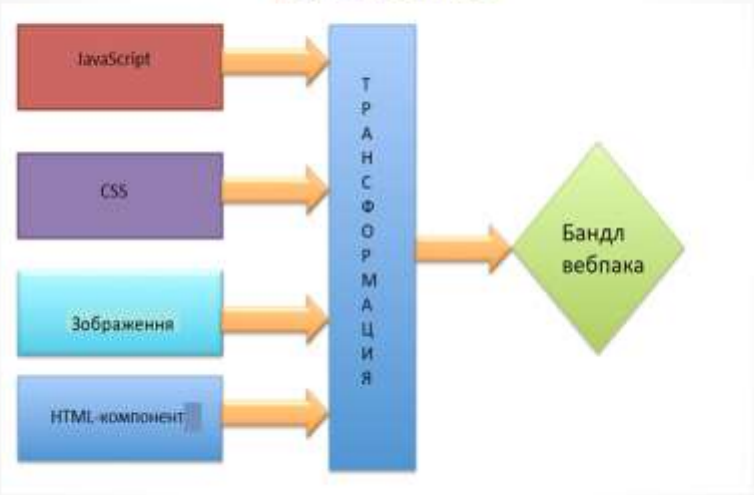


Схема роботи webpack та взаємодія клієнтських компонентів



Тестування API через Junit5 and Mockito

API

↓

JUnit 5

↓

5 + mockito

Завершення

Основний вигляд адміністративних панель платформи: