**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ДЗ «ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА»**

| Навчально-науковий інститут фізики, математики та інформаційних технологій |
| --- |
| (назва факультету, інституту) |
| Інформаційних технологій та систем |
| (назва кафедри) |

Пояснювальна записка

до кваліфікаційної роботи

за першим (бакалаврським) рівнем освіти

на тему: **РОЗРОБКА ТА ВПРОВАДЖЕННЯ CRM СИСТЕМИ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ РОБОТИ ЦЕНТРУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ УНІВЕРСИТЕТУ**

Виконав: здобувач вищої освіти 4 курсу

спеціальності

121 «Інженерія програмного забезпечення»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр і назва напряму підготовки, спеціальності)

Клопов Д.Р.

(прізвище та ініціали)

Керівник Семенов М.А.

(прізвище та ініціали)

Рецензент Козуб Ю.Г.

(прізвище та ініціали)

Полтава – 2023 року



**ЗМІСТ**

[**ЗМІСТ** 2](#_heading=h.2xcytpi)

[**ВСТУП** 4](#_heading=h.1ci93xb)

[**РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ** 9](#_heading=h.206ipza)

[**1.1**](#_heading=h.2zbgiuw) **Аналіз існуючих систем** 9

[**1.2**](#_heading=h.1d96cc0) **Аналіз основних вимог та потреб CRM системи** 12

[**1.3**](#_heading=h.j8sehv) **Аналіз перспектив розвитку сучасних CRM систем** 13

[**1.4**](#_heading=h.2eclud0) **Висновок за розділом** 21

[**РОЗДІЛ 2. ОБГРУНТУВАННЯ ВИБОРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ** 23](#_heading=h.3dhjn8m)

[**2.1**](#_heading=h.1smtxgf) **Вибір основної мови програмування** 23

[**2.2**](#_heading=h.1c1lvlb) **Основний фреймворк проєкту** 25

[**2.2.1** **Spring Boot** 26](#_heading=h.1pgrrkc)

[**2.2.2 Spring MVC** 27](#_heading=h.1h65qms)

**2.2.3 Spring Data**……………………………......…………..………………...26

[**2.2.4** **Spring Security** 29](#_heading=h.1maplo9)

[**2.2.5** **Spring Test** 30](#_heading=h.1e03kqp)

[**2.2.6** **Висновки по використанню Spring** 31](#_heading=h.2d51dmb)

[**2.3**](#_heading=h.24ufcor) **Шаблонізатор ThymeLeaf** 31

[**2.4**](#_heading=h.u8tczi) **Інтегроване середовище розробника** 33

[**2.5**](#_heading=h.2zlqixl) **Docker Desktop** 38

[**2.6**](#_heading=h.2apwg4x) **Системи контролю версій, репозиторії.** 40

[**2.8**](#_heading=h.217pygo) **Continuous Integration (CI) та Continuous Delivery/Deployment (CD)** 46

[**2.9**](#_heading=h.3r6zjac) **Огляд хостингів** 48

[**2.10**](#_heading=h.3kuv7i6) **Вибір бази данних** 51

[**2.11**](#_heading=h.3orulsy) **Висновок за розділом** 53

[**РОЗДІЛ 3. РОЗРОБКА CRM СИСТЕМИ УНІВЕРСИТЕТУ** 55](#_heading=h.2hcarzs)

[**3.1**](#_heading=h.whl27l) **Ініціалізація та перший запуск** 55

[**3.2**](#_heading=h.4kqq8sh) **Створення та підключення бази данних, налаштування міграцій.** 57

**3.3**  **Репозиторії, сервіси, контролери**…………………………………………57

[**3.4**](#_heading=h.1r8w8gf) **Налаштування Docker**……………………………………………...........61 

**3.5 Spring Security**…………………………………………………………………63

[**3.6** **Висновок за розділом** 68](#_heading=h.29slzgx)

[**ВИСНОВКИ** 70](#_heading=h.2n7rvfy)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 71](#_heading=h.1fs81ms)

[**ДОДАТОК А** 74](#_heading=h.44wffc2)

[**ДОДАТОК Б** 76](#_heading=h.z6zzro)

[**ДОДАТОК В** 78](#_heading=h.1ybxsna)

[**ДОДАТОК Г** 79](#_heading=h.2xgvliw)

[**ДОДАТОК Є** 80](#_heading=h.3wlteei)

[**ДОДАТОК Ж** 81](#_heading=h.qwdyu4)

[**ДОДАТОК З** 83](#_heading=h.1q1brpq)

**ВСТУП**

Реалії сьогодення такі, що існування підприємств або державних установ без централізованих систем документообігу неможливо назвати ефективним та впорядкованим.

Основна мета систем документообігу полягає в тому, щоб забезпечити організоване, ефективне і безпечне керування документами у цифровому форматі. У сучасному світі існує безліч інформації, яка постійно зростає і змінюється. Великі організації, компанії та установи зіштовхуються з потоком документів, даних та інформації, яку необхідно організовувати, керувати та зберігати.

В таких умовах системи документообігу стають незамінними інструментами, які забезпечують легкий доступ до інформації, зберігають її впорядкованою та допомагають управляти документами ефективно та безпечно.

Одна з головних переваг систем документообігу полягає у зручності та швидкості пошуку необхідної інформації. Замість того, щоб витрачати години на ручний пошук документів, системи документообігу дозволяють користувачам здійснювати пошук за ключовими словами, фільтрами або метаданими, що значно прискорює процес та дозволяє швидко знайти потрібну інформацію.

Крім того, системи документообігу полегшують співпрацю та комунікацію між співробітниками. Вони надають засоби для спільної роботи над документами, коментування, внесення змін та відстеження статусу завдань. Це забезпечує більш ефективну комунікацію, сприяє спільній роботі та покращує загальну продуктивність колективу.

Системи документообігу також гарантують безпеку та конфіденційність інформації. Вони дозволяють обмежувати доступ до документів за допомогою прав доступу та ролей користувачів. Це особливо важливо в сферах, де зберігання та обробка конфіденційної або регульованої інформації є необхідними.

Обіг документації в університеті є невід'ємною частиною його діяльності. Університети мають великий обсяг документів, які обробляються та передаються між різними відділами, факультетами, адміністративними підрозділами та працівниками.

Потік документації починається зі створення документів, які можуть бути академічними записами, науковими дослідженнями, звітами, фінансовими документами та багатьма іншими. Після створення документи можуть бути передані на рецензування, редагування або погодження до відповідних осіб або відділів. Завершивши ці процеси, документи можуть бути зареєстровані та надані в обіг.

У процесі переміщення документів в університеті використовуються різні методи передачі і обробки. Це може включати використання електронної пошти, внутрішніх систем обміну документами, паперових канцелярських документів та інших засобів комунікації. Під час обігу документів можуть використовуватися електронні підписи та системи контролю доступу для забезпечення безпеки та автентифікації.

Обробка документації в університеті також включає процеси зберігання та архівування документів. Деякі документи можуть бути збережені у паперовому форматі, в інших випадках вони можуть бути електронно збережені на серверах або хмарних системах зберігання. Документи можуть бути розподілені та доступні різним працівникам і студентам залежно від їх прав доступу.

Впровадження цифрової системи документообігу в університеті є необхідним кроком у сучасному світі. Така система дозволить замінити традиційний паперовий обіг документів на електронний, що принесе багато переваг.

По-перше, цифрова система документообігу забезпечить ефективну організацію та керування документами. Всі документи будуть зберігатися в централізованій базі даних, де їх легко знайти, редагувати та надсилати. Це зменшить час, потрібний для обробки документів, та спростить співпрацю між відділами та працівниками.

По-друге, цифрова система документообігу дозволить забезпечити швидкий та зручний доступ до документів. Завдяки пошуковим інструментам та категоризації документів, користувачі зможуть швидко знаходити необхідну інформацію. Це покращить продуктивність та пришвидшить процеси прийняття рішень в університеті.

По-третє, цифрова система документообігу забезпечить безпеку та конфіденційність документів. Захищена база даних та контроль доступу забезпечать, що лише авторизовані користувачі матимуть доступ до конфіденційної інформації. Крім того, резервне копіювання даних та захист від втрати забезпечать безпечне зберігання документів.

По-четверте, впровадження цифрової системи документообігу дозволить скоротити витрати на друк та зберігання паперових документів. Це призведе до економії коштів, а також сприятиме збереженню навколишнього середовища, зменшуючи використання паперу та інших матеріалів.

Загалом, впровадження цифрової системи документообігу в університеті має безліч переваг. Вона покращить ефективність, точність та безпеку обробки документів, сприятиме швидкому доступу до інформації та зменшить витрати. Це важливий крок у сучасному світі, де швидкість, ефективність та безпека є вирішальними факторами для успіху університету.

**Об’єкт дослідження:** сучасні системи документообігу, що посіли провідне місце у процедурі реєстрації та обміну документів між установами, залишивши позаду журнали вхідної документації.

**Предмет дослідження:**створення з врахуванням постійних вимог споживачів до нововведень, онлайн додатку, розробленого для підтримання процесу управління документами.

**Мета роботи** - розробка програмного забезпечення для документообігу університету.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Сформулювати основні завдання, модулі та функції, які буде вирішувати цифрова система документообігу;
2. Провести аналіз та обрати програмне забезпечення для реалізації поставленого технічного завдання;

3. Змоделювати, розробити та протестувати електронна система керування документами.

**Методи дослідження**

Дослідження теми електронної системи документообігу університету вимагає використання різних методів, що дозволяють ретельно дослідити та проаналізувати цей аспект університетської діяльності.

* Літературний аналіз — цей метод передбачає дослідження наявної літератури, наукових статей, книг, журналів та інших джерел, що стосуються електронних систем документообігу в університетському середовищі. Аналізуючи ці джерела, дослідник може отримати розуміння основних концепцій, принципів та технологій, пов'язаних з електронним документообігом.
* Аналіз даних — Зібрані дані можна проаналізувати за допомогою різних статистичних методів та програмного забезпечення для виявлення зв'язків, тенденцій та паттернів. Цей аналіз може допомогти виявити ефективність, виклики та можливості електронної системи документообігу в університеті.
* Порівняльний аналіз: Для отримання ширшого розуміння теми дослідження можна виконати порівняльний аналіз електронних систем документообігу, які використовуються в інших університетах або організаціях. Це дозволить виявити передові практики, успішні рішення та можливості для вдосконалення системи в університеті.
* Експертна оцінка: Важливим методом дослідження є залучення експертів, які мають досвід у галузі електронного документообігу або в університетській сфері. Експерти можуть надати цінні відгуки, поради та рекомендації щодо впровадження та оптимізації електронної системи документообігу в університеті.

Використання цих методів дослідження дозволить зібрати об'єктивні дані, провести аналіз та зробити висновки щодо ефективності та доцільності впровадження цифрової системи документообігу в університетському середовищі.

**РОЗДІЛ 1.   
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

Аналіз предметної області з теми електронної системи документообігу університету передбачає ретельне вивчення і розуміння ключових аспектів, принципів і процесів, пов'язаних з управлінням документами в університетському середовищі.

# **Аналіз існуючих систем**

Аналіз існуючих систем документообігу включає дослідження та оцінку різних рішень та платформ, які використовуються в організаціях для управління документами. Основні критерії для аналізу включають функціональність, надійність, безпеку, витрати та можливості розширення. Для вивчення основних можливостей, модулів та функцій існуючих цифрових систем документообігу нижче наведено приклади існуючих систем.

* **Microsoft SharePoint** є потужною платформою для спільної роботи та керування документами, яка допомагає організаціям ефективно співпрацювати та керувати своїми документами та контентом. Безпека даних та контроль доступу є важливою складовою SharePoint, де адміністратори можуть налаштовувати права доступу до документів і забезпечувати конфіденційність інформації. Крім того, SharePoint надає інструменти для організації завдань, календарів, списків контактів та інших ресурсів, що полегшує планування та керування процесами. Однією з найбільших переваг SharePoint є його інтеграція з іншими продуктами Microsoft, такими як Microsoft Office, Outlook, Teams та інші.



Рис. 1.1. Логотип програми Microsoft SharePoint

* **Alfresco** є відкритою платформою для керування документами та спільної роботи, яка надає широкий спектр функцій та можливостей для організацій.



Рис. 1.2. Логотип програми Alfresco

* **OpenText Documentum** є потужною та високопродуктивною системою керування документами, яка розроблена для ефективного управління та контролю над документами, спільної роботи та забезпечення безпеки інформації.



Рис. 1.3. Логотип програми OpenText Documentum

* **IBM FileNet** є потужною та інтегрованою платформою для керування контентом та документами, яка дозволяє організаціям ефективно управляти своїм цифровим контентом, забезпечуючи його безпеку, доступність та оптимізацію бізнес-процесів.



Рис. 2.4. Логотип IBM FileNet

*Таблиця 1.1*

**Таблиця основних відмінностей сучасних систем документообігу**

| **Основні відмінності** | **Microsoft SharePoint** | **Alfresco** | **OpenText Documentum** | **IBM FileNet** |
| --- | --- | --- | --- | --- |
| **Тип системи** | Комерційна | Відкрита | Комерційна | Комерційна |
| **Ліцензування** | Платне | Вільне | Платне | Платне |
| **Архітектура** | Клієнт-сервер | Клієнт-сервер | Клієнт-сервер | Клієнт-сервер |
| **Масштабованість** | Велика | Середня | Мала | Велика |
| **Функціональні можливості** | Широкий спектр | Середній спектр | Широкий спектр | Малий спектр |
| **Наявність розширень** | Багато | Обмежено | Багато | Мало |
| **Підтримка стандартів** | Широкий спектр | Середній спектр | Широкий спектр | Середній спектр |

* 1. **Аналіз основних вимог та потреб CRM системи**

Основні вимоги до системи документообігу університету включають:

* Централізоване зберігання: Система повинна забезпечувати централізоване зберігання всіх документів та матеріалів, пов'язаних з університетською діяльністю. Це дозволить зручний доступ до інформації та уникнення втрати або неправильного розташування документів.
* Ефективне управління документами: Система повинна забезпечувати можливості ефективного управління документами, включаючи створення, редагування, перегляд, видалення та контроль версій. Вона повинна підтримувати організацію документів у логічній структурі та забезпечувати швидкий пошук та навігацію.
* Безпека та конфіденційність: Система повинна мати вбудовані механізми для забезпечення безпеки та конфіденційності документів. Це включає рівні доступу до інформації, автентифікацію користувачів, шифрування даних та аудит активності.
* Автоматизація бізнес-процесів: Система повинна підтримувати автоматизацію бізнес-процесів, пов'язаних з документообігом, які виникають університетським середовищем. Це можуть бути процеси схвалення документів, розсилки, архівування та інші дії, які можуть бути автоматизовані для підвищення ефективності та продуктивності.
* Інтеграція з існуючими системами: Система повинна бути здатною інтегруватися з існуючими університетськими системами, такими як системи управління студентськими записами, фінансові системи та інші. Це дозволить забезпечити обмін даними та взаємодію між різними системами, спрощуючи роботу користувачам.
* Мобільний доступ: З урахуванням зростаючої мобільності користувачів, система повинна надавати можливість доступу до документів та функцій системи з різних мобільних пристроїв. Це дозволить користувачам отримувати доступ до необхідної інформації з будь-якого місця та в будь-який час.
* Масштабованість: Система повинна бути масштабованою і готовою впоратися з ростом обсягів документації та користувачів. Вона повинна мати потужність для обробки великої кількості документів і забезпечувати швидкий доступ до них, незалежно від обсягу.
* Зручний інтерфейс користувача: Система повинна мати зручний та інтуїтивно зрозумілий інтерфейс користувача, який дозволяє легко навігувати, швидко знаходити необхідну інформацію та виконувати різні дії з документами.

Ці вимоги є ключовими для ефективного функціонування системи документообігу університету, що дозволить забезпечити ефективне управління документами, зручний доступ до інформації, безпеку та забезпечити автоматизацію бізнес-процесів.

* 1. **Аналіз перспектив розвитку сучасних CRM систем**

Аналіз перспектив розвитку сучасних систем документообігу включає розгляд тенденцій та напрямків, які впливають на розвиток цієї галузі. Розвиток сучасних систем документообігу є вкрай доцільним у сучасному світі, оскільки вони пропонують низку переваг та вирішують ряд проблем, пов'язаних з обробкою документів із великим обсягом інформації. Одна з головних переваг сучасних систем документообігу полягає в автоматизації рутинних процесів. Замість ручного оброблення та переміщення фізичних документів, цифрові системи документообігу дозволяють автоматизувати процеси створення, збереження, редагування та поширення документів. Це зменшує ризик помилок, прискорює час обробки документів та покращує продуктивність робочих процесів. Крім того, розвиток сучасних систем документообігу дозволяє забезпечити зручний та безпечний доступ до документів. Цифрові системи дозволяють зберігати документи в електронному форматі та забезпечувати контроль доступу до них. Це покращує конфіденційність та безпеку інформації, а також дає можливість користувачам знаходити, переглядати та редагувати документи з будь-якого місця та в будь-який час, що забезпечує більшу гнучкість та ефективність роботи. Інтеграція з іншими системами також є важливим аспектом розвитку сучасних систем документообігу. Це дозволяє забезпечити обмін даними між різними системами, такими як системи управління взаємовідносинами з клієнтами (CRM), системи управління проектами (PM), електронні поштові сервіси та багато інших. Інтеграція спрощує обмін інформацією, покращує комунікацію та співпрацю між різними департаментами та структурними підрозділами організації.

Крім того, розвиток сучасних систем документообігу сприяє збереженню та архівуванню документів. Цифрові системи дозволяють зручно та безпечно зберігати документи, забезпечуючи контроль доступу, можливість резервного копіювання та відновлення даних, а також можливість створення електронних архівів для збереження документів на довгострокову перспективу. Це сприяє збереженню історичної документації, забезпечує дотримання правових та регуляторних вимог щодо збереження документів та спрощує пошук та відновлення інформації.

На зараз вже простежуються основні напрями, за якими в майбутньому будуть створюватися більш сучасні та технологічні електронні системи документообігу:

**Цифрова трансформація електронних систем документообігу** відіграє важливу роль у сучасному світі, перетворюючи спосіб, яким ми працюємо з документами і інформацією. Цей процес передбачає впровадження новітніх технологій та цифрових інструментів для поліпшення всіх етапів життєвого циклу документів. Однією з ключових переваг цифрової трансформації є зниження обсягу паперової документації.

Замість традиційного паперового документообігу, електронні системи дозволяють зберігати та обробляти документи в цифровому форматі. Це зменшує необхідність у фізичному зберіганні, транспортуванні та пошуку документів, що поліпшує ефективність робочих процесів та зменшує витрати на документообіг. Ще однією перевагою цифрової трансформації є покращена доступність та швидкість обробки документів. Електронні системи документообігу дозволяють легко зберігати, організовувати та шукати документи за допомогою ключових слів або фільтрів. Це спрощує доступ до необхідної інформації, скорочує час пошуку та обробки документів, що поліпшує продуктивність роботи та сприяє швидкому прийняттю рішень.

Крім того, цифрова трансформація електронних систем документообігу сприяє підвищенню безпеки та конфіденційності даних. Електронні системи дозволяють контролювати доступ до документів, встановлювати права користувачів та застосовувати шифрування для захисту від несанкціонованого доступу. Це допомагає забезпечити конфіденційність документів, запобігти втрату або пошкодженню даних та забезпечити відповідність вимогам щодо захисту інформації. Однак, цифрова трансформація електронних систем документообігу також супроводжується викликами та недоліками. Перехід до цифрових систем вимагає великих інвестицій у технології, інфраструктуру та навчання персоналу.

Також, можуть виникати проблеми зі сумісністю різних систем, потреба у додатковій кібербезпеці та захисті від кібератак. У підсумку, цифрова трансформація електронних систем документообігу є необхідною у сучасному світі, оскільки вона забезпечує покращення ефективності роботи, зниження витрат та підвищення безпеки документів. Проте, необхідно ретельно планувати та управляти процесом трансформації, враховуючи виклики та ризики, що виникають при впровадженні нових технологій.

**Використання хмарних технологій в процесі розвитку систем документообігу** є важливою та перспективною стратегією. Хмарні технології дозволяють зберігати, обробляти та обмінювати документи у віртуальному середовищі, що принесе численні переваги для університету. Однією з головних переваг використання хмарних технологій є легкість доступу та спільної роботи над документами.

Завдяки хмарним системам, користувачі можуть отримати доступ до своїх документів з будь-якого пристрою та місця, де є Інтернет-підключення. Це сприяє зручності та ефективності роботи, особливо для спільних проектів або роботи з великим обсягом документації. Хмарні технології також забезпечують масштабованість та гнучкість системи документообігу. Університети можуть легко розширювати або зменшувати потужності зберігання даних та обробки документів залежно від своїх потреб. Це дозволяє ефективно управляти ресурсами та забезпечувати безперебійну роботу системи навіть при змінних навантаженнях.

Безпека є ще однією вагомою перевагою хмарних технологій. Багато хмарних платформ надають рівень захисту даних, шифрування та механізми автентифікації, що дозволяє забезпечити конфіденційність та цілісність документів. Крім того, резервне копіювання даних та можливість відновлення в разі аварії забезпечують надійність та невтратуваність документації. Використання хмарних технологій також сприяє економії витрат. Замість інвестування власних серверних інфраструктур та обладнання, університети можуть використовувати інфраструктуру хмарного провайдера. Це дозволяє зменшити витрати на придбання, установку та підтримку апаратного забезпечення.

У підсумку, використання хмарних технологій в розвитку систем документообігу університету є обґрунтованою та перспективною стратегією. Це сприяє зручності доступу до документів, масштабованості, безпеці та економії витрат. Зростаюча популярність хмарних технологій серед університетів свідчить про їх значний потенціал у покращенні процесів документообігу та сприянні розвитку освітньої сфери.

**Інтеграція з іншими системами** є ключовим аспектом вдосконалення систем документообігу. В сучасному світі, де комп'ютеризація та автоматизація процесів є невід'ємною частиною бізнес-середовища, забезпечення безперебійної і ефективної взаємодії між різними системами є критично важливим завданням. Інтеграція систем документообігу з іншими системами дозволяє забезпечити автоматичний обмін даними та документами між різними програмними рішеннями. Наприклад, інтеграція з системою управління відносинами з клієнтами (CRM) дозволяє автоматично зберігати документи, пов'язані з клієнтами, відразу відповідно до встановлених правил та процедур. Це сприяє покращенню ефективності роботи, скороченню часу на обробку документів та зменшенню ризиків помилок. Інтеграція також дозволяє підключати систему документообігу до інших важливих інформаційних джерел, таких як електронна пошта, бази даних, файлові сховища та інші. Це дозволяє отримувати та обробляти дані з різних джерел в одній централізованій системі, спрощуючи роботу з документами та покращуючи доступність і точність інформації. Однією з переваг інтеграції є також автоматизація рутинних процесів. Завдяки інтеграції, можна налаштувати автоматичні потоки роботи, які будуть виконувати певні дії з документами на основі заданих умов. Наприклад, автоматичне маркування, розподіл або нагадування про терміни здійснюються без необхідності вручну керувати цими процесами. Загалом, інтеграція з іншими системами дозволяє створити потужну та гнучку інформаційну інфраструктуру, яка забезпечує безперебійний потік документів та даних між різними системами. Це полегшує роботу співробітників, забезпечує швидкий доступ до необхідної інформації та сприяє підвищенню ефективності та продуктивності університету.

**Мобільність має значний вплив на розвиток систем документообігу**, забезпечуючи нові можливості та перетворюючи спосіб, яким люди працюють з документами. З поширенням сучасних мобільних пристроїв, таких як смартфони і планшети, співробітники можуть отримати доступ до документів та виконувати роботу навіть поза офісом, що сприяє зручності та ефективності. Мобільна документообігова система дозволяє співробітникам отримувати, переглядати, редагувати та обмінюватися документами з будь-якого місця, де є доступ до Інтернету. Це дозволяє працювати в режимі реального часу, без затримок і обмежень, що особливо корисно для команд, які працюють в розподілених місцях або виїжджають в командировки. Мобільна документообігова система також сприяє зменшенню паперової документації та упрощує процеси зберігання і пошуку документів. Завдяки цифровому формату, документи можуть бути легко організовані, класифіковані та індексовані, що полегшує їх пошук та забезпечує швидкий доступ до потрібної інформації. Крім того, мобільна документообігова система дозволяє забезпечити безпеку та конфіденційність документів. Захист даних, шифрування та контроль доступу дозволяють зберігати документи в безпечному середовищі, зменшуючи ризик втрати або несанкціонованого доступу до конфіденційної інформації. Мобільність також сприяє покращенню комунікації та співпраці. Завдяки можливості швидко обмінюватися документами, коментарями та зворотними зв'язками, команди можуть працювати синхронно та ефективно співпрацювати над спільними проектами та завданнями. Узагалі, мобільність впливає на розвиток систем документообігу, надаючи більш гнучкі, зручні та продуктивні рішення для співробітників. Вона дозволяє пристосуватися до швидкотемпованого темпу роботи, забезпечує доступ до інформації в будь-який час та з будь-якого місця, що сприяє підвищенню ефективності та конкурентоспроможності організації.

**У цифровому віці, де обробка документів стає все більш електронною**, забезпечення безпеки та конфіденційності є надзвичайно важливим аспектом. Розвиток технологій вимагає ефективних заходів забезпечення захисту даних та документів в цифрових системах обробки документів. Нижче розглянуто декілька важливих аспектів, які сприяють покращенню безпеки та конфіденційності в цифрових системах обробки документів:

* Автентифікація та авторизація: Цифрові системи обробки документів вимагають суворої ідентифікації та авторизації користувачів. Використання сильних паролів, біометричних методів автентифікації та двофакторної автентифікації допомагає забезпечити доступ тільки авторизованим особам.
* Шифрування даних: Шифрування використовується для захисту даних в цифрових системах обробки документів. Важливі дані можуть бути зашифровані під час передачі та зберігання, що робить їх незрозумілими для несанкціонованих осіб, які мають доступ до цих даних.
* Захист від несанкціонованого доступу: Розробники систем документообігу мають впроваджувати заходи безпеки, такі як файрволи, антивірусні програми та системи виявлення вторгнень, для захисту від несанкціонованого доступу до документів та даних.
* Аудит та моніторинг: Системи документообігу можуть включати функції аудиту та моніторингу, які дозволяють вести журнали дій користувачів, виявляти незвичайну активність та потенційні загрози безпеці. Це допомагає вчасно виявляти та вирішувати проблеми безпеки.
* Контроль доступу та рівні привілеїв: Цифрові системи документообігу можуть мати гнучкі налаштування контролю доступу та рівнів привілеїв. Це дозволяє обмежувати доступ користувачів до конфіденційних документів та інформації лише до необхідного рівня.
* Захист від зловмисних програм та вразливостей: Постійне оновлення програмного забезпечення, використання антивірусних програм та заходи для захисту від зловмисних програм допомагають запобігти вразливостям та атакам на системи документообігу.
* Захист від втрати даних: Регулярне резервне копіювання даних та використання систем відновлення даних дозволяють уникнути втрати важливої інформації в разі аварій, витоку даних чи інших непередбачуваних ситуацій.

Ці аспекти сприяють підвищенню безпеки та конфіденційності в цифрових системах обробки документів. Важливо враховувати ці фактори при розробці та впровадженні систем документообігу, щоб забезпечити захист важливих документів та конфіденційність даних.

**Інтелектуальний аналіз даних** є важливою та перспективною складовою сучасних систем документообігу. Він використовує передові алгоритми та технології для автоматичного виявлення, інтерпретації та аналізу даних, що містяться в документах. Інтелектуальний аналіз даних дозволяє отримувати цінні інсайти та інформацію з великого обсягу документації, що допомагає приймати кращі управлінські рішення та покращувати ефективність роботи. За допомогою інтелектуального аналізу даних, системи документообігу можуть автоматично класифікувати документи за типом, виявляти ключові терміни та поняття, розпізнавати структуру документів та виявляти зв'язки між ними. Наприклад, система може автоматично виділяти інформацію зі звітів, контрактів, листування та інших документів і створювати структуровану базу даних, що спрощує пошук та доступ до важливої інформації. З часом інтелектуальний аналіз даних у системах документообігу може розвиватись і стати ще більш потужним та розширеним. Завдяки постійному розвитку штучного інтелекту, машинного навчання та обробки природних мов, системи зможуть виявляти не тільки базову інформацію, але і виявляти складні зв'язки між документами, прогнозувати тенденції та робити аналітичні висновки. Наприклад, інтелектуальний аналіз даних може виявляти тенденції в змісті документів, передбачати можливі ризики або пропонувати оптимальні шляхи оптимізації бізнес-процесів на основі аналізу великого обсягу документації. Використання штучного інтелекту та машинного навчання дозволить системам документообігу стати ще більш автоматизованими, інтелектуальними та адаптивними до потреб користувачів. Розвиток інтелектуального аналізу даних у системах документообігу має потенціал значно поліпшити управління документами, забезпечити швидкий доступ до необхідної інформації та підвищити продуктивність. Однак, важливо враховувати етичні та конфіденційність аспекти при роботі з даними та забезпечувати належний рівень захисту та приватності документів та інформації.

У підсумку, **розвиток сучасних систем документообігу** є необхідним у сучасному світі, оскільки вони допомагають організаціям забезпечити ефективне та безпечне управління документами, поліпшити продуктивність робочих процесів, підвищити доступність та надійність інформації, а також забезпечити відповідність правовим та регуляторним вимогам.

* 1. **Висновок за розділом**

Аналізуючи предметну область систем електронного документообігу, можна зробити наступні висновки. Системи електронного документообігу є необхідною складовою сучасного управління документами в організаціях. Вони сприяють ефективному обробленню, зберіганню, пошуку та обміну документами, спрощують робочі процеси та забезпечують швидкий доступ до необхідної інформації. Аналізуючи існуючі системи електронного документообігу, можна зазначити, що вони мають ряд переваг, таких як автоматизація рутинних завдань, покращення зв'язку та співпраці між співробітниками, забезпечення централізованого зберігання документів та покращення безпеки і конфіденційності інформації. Однак, при впровадженні систем електронного документообігу також потрібно враховувати деякі виклики та обмеження. Це можуть бути складність інтеграції з існуючими системами, необхідність навчання та підтримки персоналу, а також питання щодо захисту даних та забезпечення відповідності зв'язаних з ними правил та нормативів. У майбутньому розвиток систем електронного документообігу може передбачати впровадження передових технологій, таких як штучний інтелект, машинне навчання та розподілена обробка даних. Це дозволить автоматизувати більшу кількість завдань, забезпечити більш точний та швидкий аналіз документів, а також покращити розуміння та використання контекстуальної інформації.

Загалом, системи електронного документообігу відіграють важливу роль у сучасному управлінні документами, сприяючи ефективності, прозорості та безпеки робочих процесів. Продовження розвитку та впровадження передових технологій у ці системи може принести значну користь організаціям у подальшому.

**РОЗДІЛ 2.   
ОБГРУНТУВАННЯ ВИБОРУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ**

* 1. **Вибір основної мови програмування**

Java є однією з найпопулярніших мов програмування в світі, і це не випадково. Використання мови програмування Java має безліч переваг, які сприяють розробці програмного забезпечення та підвищують ефективність роботи розробників. Ось декілька ключових переваг використання Java:

* Переносимість: Java є платформонезалежною мовою, що означає, що програми, написані на Java, можуть запускатися на будь-якій платформі, яка підтримує відповідний віртуальний Java-машину (JVM). Це дозволяє розробникам створювати програмне забезпечення, яке працює на різних операційних системах без необхідності змінювати код. Об'єктно-орієнтований підхід:
* Java базується на об'єктно-орієнтованому підході, що сприяє більшій модульності, повторному використанню коду та забезпечує більшу організованість проекту. Розробники можуть створювати класи, об'єкти, успадкування та інші об'єктно-орієнтовані конструкції, що спрощують розробку та підтримку програмного забезпечення.
* Велика екосистема: Java має велику спільноту розробників та багатоцільову екосистему, яка включає в себе бібліотеки, фреймворки, інструменти розробки та інші ресурси. Це дозволяє розробникам ефективно використовувати готові рішення, прискорюючи процес розробки та підвищуючи якість програмного забезпечення. Висока продуктивність:
* Java відома своєю високою продуктивністю завдяки вбудованій оптимізації, що забезпечує швидке виконання програм. Вона також має вбудовану систему збору сміття, яка автоматично вивільняє пам'ять від невикористаних об'єктів, спрощуючи управління пам'яттю для розробників. Безпека: Java має вбудовану систему безпеки, яка дозволяє розробникам створювати безпечні програми. Вона має механізми для управління доступом, перевірки цілісності та обмежень, що допомагають запобігати вразливостям та забезпечують захист даних.
* Широке застосування: Java використовується в багатьох сферах, включаючи веб-розробку, мобільний розробку, розробку вбудованих систем, штучний інтелект, фінанси та інше. Це дозволяє розробникам мати багато перспективних можливостей для кар'єрного росту.

Звичайно, як і будь яка мова програмування, Java має свої недоліки, які необхідно розуміти та враховувати під час розробки програмного забезпечення, аби уникнути подальших проблем при використанні програмного продукту і його вдосконаленні на подальших етапах розвитку. **Основними недоліками вибраної мови програмування є:**

* **Висока витрата пам'яті:** Java відома своєю високою витратою пам'яті. Це може стати проблемою, особливо при розробці програм для обмежених пристроїв або систем з обмеженими ресурсами. Розмір виконуваного коду та вимоги до пам'яті можуть бути значними, що може впливати на продуктивність програми.
* **У порівнянні з більш швидшкими мовами програмування**, такими як C++ або C#, Java може бути менш ефективною в плані швидкодії. Це пов'язано з додатковими прошарками абстракції та вбудованою системою збору сміття, що може призводити до затримок в роботі програми.
* **Обмежені можливості низькорівневого програмування:** Оскільки Java є високорівневою мовою програмування, вона не надає такого рівня контролю над апаратними ресурсами, як низькорівневі мови. Це може бути проблемою для деяких задач, де потрібна пряма маніпуляція з пам'яттю або реєстрами процесора.
* **Обмеження на розробку під деякі платформи:** Хоча Java є платформонезалежною мовою, є випадки, коли вона може бути обмежена у підтримці певних платформ або пристроїв. Деякі функції можуть бути недоступними або працювати не настільки ефективно на певних платформах.
* **Великий об'єм коду**: Завдяки своїй об'єктно-орієнтованій природі та широкій бібліотеці, програми на Java можуть мати великий обсяг коду. Це може зробити код складним для розуміння та підтримки, особливо для початківців або для проектів з обмеженими ресурсами.



Рис. 2.1. Логотип мови програмування Java

Враховуючи ці недоліки, важливо ретельно обдумати вибір мови програмування для своїх проектів. Використання мови Java має свої плюси, але також повинно бути зважено на її недоліки та потреби проекту.

* 1. **Основний фреймворк проєкту**

Для мови програмування Java за весь час розвитку було написано багато фреймворків та додаткових інструментів, які полегшують роботу програміста. Одним з найбільших та найрозвиненіших є фрейморк Spring.



Рис. 2.2. Логотип фреймворку Spring

Spring - це потужний фреймворк для розробки додатків на мові Java. Він надає розширений набір інструментів та функціональних можливостей, що допомагають розробникам швидко створювати надійні та ефективні програми. Однією з головних переваг Spring є його легкість використання та гнучкість. Він пропонує інтуїтивно зрозумілу модель програмування і спрощені конфігураційні параметри, що полегшують розробку. Ще однією сильною стороною Spring є його розширюваність. Фреймворк надає багато модулів та інструментів, які дозволяють легко інтегрувати його з іншими технологіями та фреймворками.

* + 1. **Spring Boot**

Spring Boot є фреймворком, який дозволяє швидко розробляти стійкі, масштабовані та ефективні додатки. При написанні електронної системи документообігу Spring Boot може бути надзвичайно корисним. Ось декілька ключових причин, чому варто використовувати Spring Boot:

* Швидка розробка: Spring Boot забезпечує конфігурацію за замовчуванням, що дозволяє розпочати розробку швидше. Він автоматично керує багатьма аспектами додатку, такими як завантаження залежностей, налаштування середовища та стартові параметри, що робить процес розробки більш простим і ефективним.
* Вбудований сервер: Spring Boot постачається з вбудованими серверами, такими як Tomcat або Jetty, що дозволяє вам запускати вашу систему документообігу безпосередньо з фреймворку. Це спрощує розгортання та тестування вашого додатку.
* Автоконфігурація: Spring Boot надає автоматичну конфігурацію, яка дозволяє визначати поведінку додатку за допомогою анотацій та налаштувань. Це робить процес налаштування додатку більш простим і зменшує кількість необхідного коду.
* Управління залежностями: Spring Boot має вбудований менеджер залежностей Maven або Gradle, який дозволяє легко керувати бібліотеками та залежностями вашого проєкту. Ви можете швидко додавати та оновлювати залежності без необхідності вручну налаштовувати проєкт.
* Моніторинг та управління: Spring Boot надає вбудований інструментарій для моніторингу, відладки та управління додатком. Ви можете легко настроїти інтеграцію з інструментами моніторингу, такими як Actuator, щоб отримувати метрики та статус вашого додатку.

Використання Spring Boot у розробці електронної системи документообігу дозволить прискорити процес розробки, спростити налаштування, забезпечити швидке розгортання та зменшити кількість необхідного коду. Цей фреймворк має потужну екосистему і надає багато інструментів, які допоможуть вам побудувати стійку та ефективну систему електронного документообігу.

* + 1. **Spring MVC**

Spring MVC є одним з ключових компонентів фреймворку Spring, який використовується для розробки веб-додатків. При написанні електронної системи документообігу використання є вкрай необхідним компонентом. Декілька способів, які можна використовувати Spring MVC в процесі розробки системи документообігу:

* Модель-представлення-контролер (MVC) архітектура: Spring MVC заснований на архітектурному шаблоні MVC, що розділяє логіку програми на модель (дані), представлення (користувацький інтерфейс) та контролер (управління запитами та відповідями). Це дозволяє ефективно організувати код і полегшує розробку та підтримку системи.
* Обробка HTTP запитів: Spring MVC надає потужний механізм для обробки HTTP запитів. Ви можете визначити контролери, які відповідають на різні маршрути та запити. Контролери обробляють запити, виконують необхідні дії та повертають відповіді на клієнт.
* В’язкість з моделями даних: За допомогою Spring MVC ви можете легко використовувати моделі даних для передачі та обробки даних у системі документообігу. Ви можете визначити моделі, які представляють об’єкти документів, користувачів, метадані тощо. Це допомагає організувати дані та забезпечує їх легку маніпуляцію.

Використання Spring MVC у розробці електронної системи документообігу дозволяє ефективно організувати код, забезпечити обробку HTTP запитів, управляти моделями даних, налаштувати маршрути та перенаправлення, валідувати дані, інтегруватися з шаблонами представлення та обробляти помилки.

* + 1. **Spring Data**

Spring Data є компонентом фреймворку Spring, який спрощує взаємодію з базами даних у програмних проектах. Під час написання системи документообігу використання Spring Data може бути надзвичайно корисним. Основні можливості компоненту, які будуть використані під час розробки:

* Легке налаштування доступу до бази даних: Spring Data надає абстракцію бази даних, що дозволяє легко налаштовувати підключення до різних типів баз даних, таких як реляційні та нереляційні бази даних.
* Автоматичне створення запитів: Spring Data надає можливість автоматично створювати запити до бази даних на основі методів, визначених у репозиторіях. Ви можете використовувати конструкції іменованих методів або навіть використовувати мову запитів, таку як JPQL або SQL, для виразного визначення запитів.
* Підтримка CRUD-операцій: Spring Data надає стандартні методи для створення (Create), читання (Read), оновлення (Update) та видалення (Delete) об'єктів в базі даних. Ви можете легко використовувати ці методи для маніпуляції з даними системи документообігу.
* Підтримка пошуку та сортування: Spring Data дозволяє виконувати пошук та сортування даних на основі критеріїв та умов. Ви можете використовувати методи з ключовими словами, анотації або навіть створювати складні запити з використанням спеціальних інтерфейсів.

Використання Spring Data у системі документообігу дозволяє спростити доступ до бази даних, автоматизувати створення запитів, підтримувати CRUD-операції, виконувати пошук та сортування даних, кешувати дані та управляти транзакціями. Це допомагає покращити продуктивність, надійність та розширюваність системи електронного документообігу.

* + 1. **Spring Security**

Spring Security відіграє важливу роль у системі документообігу університету, забезпечуючи безпеку та контроль доступу до різних ресурсів системи. Основна мета Spring Security полягає в захисті конфіденційної інформації, запобіганні несанкціонованому доступу та забезпеченні автентифікації та авторизації користувачів.

У системі документообігу університету компонент Spring Security буде виконувати наступні функції:

* Аутентифікація користувачів: Spring Security надає механізми для перевірки ідентифікаційних даних користувачів, таких як логін та пароль. Він забезпечує захищену аутентифікацію, включаючи можливість налаштування різних стратегій аутентифікації, включаючи базу даних, LDAP, OAuth і багато інших.
* Авторизація доступу: Spring Security дозволяє визначати рівні доступу та ролі користувачів, а також налаштовувати права доступу до різних ресурсів системи. Ви можете використовувати анотації або конфігураційні файли для встановлення правил авторизації, що дозволяє обмежувати доступ до функціональності системи на основі ролей користувачів.
* Захист від атак: Spring Security надає захист від різних типів атак, таких як перехоплення сесій, XSS (міжсайтовий скриптинг), CSRF (міжсайтова подделка запиту) та інші. Він забезпечує механізми фільтрації та перевірки вхідних даних, що дозволяє зменшити ризик зловживання атаками та зберігання надійності даних.
* Журналювання та аудит: Spring Security надає можливості для журналювання подій, пов'язаних з безпекою, і аудиту дій користувачів. Це дозволяє вести реєстрацію дій користувачів, що спрощує виявлення та розслідування подій безпеки.

Загалом, роль Spring Security у системі документообігу університету полягає в забезпеченні захисту, контролі доступу та безпеки системи. Використання Spring Security дозволяє зменшити ризики безпеки, забезпечити автентифікацію та авторизацію користувачів та забезпечити конфіденційність та цілісність даних.

* + 1. **Spring Test**

Тестування є невід'ємною частиною розробки електронної системи документообігу університету, оскільки допомагає перевірити правильність роботи системи, виявити помилки та забезпечити високу якість програмного продукту. У цьому процесі засоби Spring Test стають незамінним помічником. Spring Test - це модуль фреймворку Spring, який надає набір інструментів для тестування програмних компонентів, реалізованих з використанням Spring. Він спрощує створення тестів, забезпечує розширені можливості для покриття різних аспектів системи тестами і допомагає забезпечити надійність та стабільність електронної системи документообігу. Використання засобів Spring Test у написанні електронної системи документообігу дозволяє забезпечити якість програмного продукту, виявити та усунути помилки та забезпечити надійність роботи системи. Вони допомагають створювати комплексні тести для різних аспектів системи, забезпечують зручний інтерфейс для виконання тестів та забезпечують повноту та точність перевірок.

* + 1. **Висновки по використанню Spring**

У висновку можна підкреслити, що фреймворк Spring відіграє ключову роль у розробці системи документообігу університету. Його використання забезпечує багато переваг і допомагає зробити процес розробки ефективним та зручним. Основні висновки про роль фреймворку Spring у розробці системи документообігу включають наступне:

1.Спрощена розробка

2. Інверсія керування

3. Інтеграція з іншими технологіями

4. Керування залежностями

5. Тестування та підтримка

Загалом кажучи, фреймворк Spring є потужним інструментом для розробки системи документообігу університету. Він допомагає створити ефективну, гнучку та легко розширювану систему, яка відповідає потребам університету. Застосування Spring дозволяє зосередитися на бізнес-логіці та функціональності системи, спрощує розробку, забезпечує високу якість та надійність, а також сприяє швидкому розвитку та змінам в майбутньому.

* 1. **Шаблонізатор ThymeLeaf**

Thymeleaf є потужним інструментом, який знаходить широке застосування під час розробки електронної системи документообігу.



Рис. 2.3. Логотип шаблонізатору Thymeleaf

Основні цілі використання Thymeleaf включають:

1. Динамічна генерація веб-сторінок: Thymeleaf дозволяє розробникам створювати динамічні веб-сторінки, які можуть відображати змінні дані з бази даних або інших джерел. Використання Thymeleaf дозволяє вставляти дані у шаблони сторінок та виконувати логіку обробки даних, що дозволяє створювати багатофункціональний та інтерактивний веб-інтерфейс.
2. Вбудовані можливості шаблонування: Thymeleaf надає широкі можливості для створення шаблонів веб-сторінок, що дозволяє використовувати повторювані елементи, включати фрагменти коду та легко організовувати структуру сторінок. Це полегшує розробку та підтримку веб-інтерфейсу системи документообігу.
3. Підтримка міжнародизації: Thymeleaf має вбудовану підтримку міжнародизації, що дозволяє легко локалізувати веб-сторінки та використовувати різні мови. Це особливо корисно для електронної системи документообігу, яка може мати користувачів з різних країн та культур.
4. Інтеграція з іншими технологіями: Thymeleaf легко інтегрується з іншими технологіями та фреймворками, такими як Spring, для створення повноцінного веб-додатка. Використання Thymeleaf разом з іншими інструментами дозволяє створювати потужні та ефективні системи документообігу.

*Таблиця 2.1*

**Таблиця переваг та недоліків використання шаблонізатору Thymeleaf**

| Переваги | Недоліки |
| --- | --- |
| Чистий та природний синтаксис | Потребує певного часу для ознайомлення |
| Вбудована підтримка міжнародизації | Обмежені можливості для статичних сторінок |
| Легка інтеграція з фреймворками та технологіями | Порівняно повільніше виконання |
| Велика спільнота розробників та підтримка | Вимагає знання синтаксису та методології роботи |
| Можливість генерації HTML, XML, JSON та ін. | Складність в управлінні шаблонами |
| Підтримка шаблонів з виразами та умовами | Залежність від розробників |
| Перевірка семантики та безпеки даних |

Отже, вміле застосування шаблонізатора у проєкті електронної системи документообігу має значні переваги і загалом саме Thymleaf є вигідним вибором.

* 1. **Інтегроване середовище розробника**

Інтегроване середовище розробника (Integrated Development Environment, IDE) - це програмний комплекс, який надає розробникам зручну робочу область для написання, відлагодження та впровадження програмного коду. Для мови програмування Java існує кілька популярних інтегрованих середовищ, які надають розширені функціональні можливості та підтримують весь процес розробки. Ось декілька таких інтегрованих середовищ для мови Java:

**1. Eclipse** є одним з найпопулярніших інтегрованих середовищ для розробки на мові Java. Воно надає широкий набір інструментів, включаючи редактор коду, відлагоджувач, систему керування версіями і підтримку розширень за допомогою плагінів.



Рис. 2.4 Логотип IDE Eclipse

1. **IntelliJ IDEA:** від компанії JetBrains є потужним інтегрованим середовищем розробки для мови Java. Воно пропонує широкі можливості автоматичного завершення коду, відлагодження, аналізу коду та підтримку фреймворків.



Рис. 2.5 Логотип IDE IntelliJ IDEA

1. **NetBeans:** це вільно розповсюджуване інтегроване середовище розробки, яке надає широкі можливості для розробки на мові Java. Воно підтримує розробку як стандартних додатків на платформі Java SE, так і веб-додатків на базі Java EE.



Рис. 2.6. Логотип IDE NetBeans

1. **Visual Studio Code** - це легкий, але потужний текстовий редактор, який може бути розширений для розробки на мові Java за допомогою розширень. Він надає функції підсвічування синтаксису, автоматичного завершення коду та відлагодження.



Рис. 2.7. Логотип IDE Visual Studio Code

1. **BlueJ** є спеціалізованим інтегрованим середовищем розробки для вивчення мови програмування Java. Воно пропонує простий інтерфейс та інструменти, спрямовані на полегшення вивчення програмування.



Рис 2.8 Логотип IDE BlueJ

*Таблиця 2.2*

**Основні переваги та недоліки найпопулярніших IDE для Java**

| **Інтегроване середовище** | **Переваги** | **Недоліки** |
| --- | --- | --- |
| **Eclipse** | Широкий вибір плагінів та розширень | Швидкість роботи може бути повільнішою у порівнянні з іншими IDE |
|  | Потужна система підтримки керування версіями | Складніше налаштування та конфігурація проєкту |
|  | Велика спільнота розробників та активна підтримка | Інтерфейс може виглядати застарілим порівняно з деякими сучасними IDE |
|  | Інструменти для відлагодження та профілювання коду |  |
| **IntelliJ IDEA** | Висока продуктивність та швидкодія | Платна версія IntelliJ IDEA Ultimate має обмеження для некомерційного використання |
|  | Розширені можливості автоматичного завершення коду, рефакторингу та аналізу коду | Вимагає більш потужний комп'ютер для оптимальної роботи |
|  | Велика кількість інтегрованих інструментів для розробки Java, включаючи фреймворки та технології | Великий обсяг функціональності може призвести до складнощів для новачків в програмуванні |
| **NetBeans** | Легке налаштування та встановлення | Менш популярне середовище порівняно з Eclipse та IntelliJ IDEA |
|  | Вбудована підтримка фреймворків Java EE та веб-технологій | Інтерфейс може виглядати менш сучасним та забарвлення може викликати деякі проблеми для користувачів зі зниженим зором |
|  | Широкі можливості розробки, включаючи редактор коду, відлагоджувач, систему керування версіями та інші | Менша спільнота розробників порівняно з Eclipse та IntelliJ IDEA |
| **Visual Studio Code** | Безкоштовна та відкрита платформа | Вимагає додаткових налаштувань та встановлення плагінів для повноцінної підтримки розробки Java |
|  | Велика кількість плагінів та розширень, що розширюють функціональність редактора | Не має таких розширених можливостей специфічно для Java, як у вищезгаданих IDE |
|  | Широкий спектр інструментів для роботи з Git, включаючи інтеграцію з GitHub |  |

Ці інтегровані середовища розробника надають зручні інструменти для програмістів, які працюють з мовою програмування Java. Вони допомагають підвищити продуктивність, спростити відлагодження та забезпечити широкі можливості для розробки програмного забезпечення. Кожне з них має свої особливості та переваги, тому вибір інтегрованого середовища залежить від потреб розробників та характеру проєкту.

* 1. **Docker Desktop**

Docker - це відкрите програмне забезпечення, яке надає зручний та ефективний спосіб розгортання, управління та запуску програмних додатків у контейнерах. Контейнери Docker є легкими та автономними середовищами, що включають всі необхідні залежності для виконання додатків.

Основною ідеєю Docker є упакування додатків разом з усіма їх залежностями в контейнери, які можуть бути розгорнуті на будь-якому комп'ютері або сервері, що підтримує Docker. Контейнери забезпечують ізольоване середовище для додатків, що дозволяє їм працювати незалежно від інших додатків чи системних налаштувань. Кожен контейнер має свої власні процеси, файли, мережеві ресурси та інші компоненти, і вони можуть бути керовані окремо.



Рис. 2.9 Логотип програмного забезпечення Docker

*Таблиця 2.3*

**Сценарії використання докеру в проєкті**

| Варіант використання | Опис |
| --- | --- |
| Розгортання додатків | Дозволяє легко розгорнути додатки у контейнерах на різних середовищах, забезпечуючи консистентність та переносимість. |
| Мікросервісна архітектура | Docker може бути використаний для розгортання та керування окремими компонентами мікросервісної архітектури, забезпечуючи масштабованість та гнучкість |
| Тестування | Надає ізольоване середовище для тестування додатків, що дозволяє швидко та ефективно виконувати тестові сценарії. |
| Управління залежностями | Дозволяє контролювати та керувати залежностями додатків, забезпечуючи відокремленість та незалежність компонентів. |
| Контінеризація баз даних | Docker може використовуватись для розгортання та управління базами даних у контейнерах, спрощуючи процес розгортання та масштабування. |
| Керування інфраструктурою | Може бути використаний для автоматизації керування інфраструктурою, такою як розгортання серверів та мережеві налаштування. |
| Continuous Integration | Docker може бути використаний для створення ізольованих тестових середовищ для процесу постійної інтеграції (CI), спрощуючи тестування та розгортання. |

Docker є потужним інструментом для розробки, розгортання та управління додатками, що дозволяє створювати переносимі, ефективні та швидкі середовища. Використання Docker може спростити процес розробки та покращити швидкість та надійність додатків.

* 1. **Системи контролю версій, репозиторії.**

Системи контролю версій (СКВ) є важливим інструментом для керування версіями файлів і спільної роботи над проектами. Основною метою СКВ є зберігання, відстеження і керування змінами, які вносяться до файлів під час розробки програмного забезпечення або інших проектів.

Найпопулярнішою та найрозвинутішою на данний момент є система контролю версій Git. Це розподілена система керування версіями, що використовується для відстеження змін в програмному коді та спільної роботи над проектами. Вона забезпечує зручний та ефективний спосіб керування версіями файлів, дозволяючи розробникам відстежувати, реєструвати та об'єднувати зміни в коді.

Git зберігає повну історію змін файлів та дозволяє повертатися до попередніх версій проекту. Завдяки розподіленій архітектурі, кожен розробник може мати повну копію репозиторію, що дозволяє працювати над проектом навіть у відсутність мережевого з'єднання. Крім того, Git дозволяє створювати гілки розробки, що дозволяє розробникам відокремлювати різні функціональності та експериментувати без впливу на головну гілку проекту.

Система контролю версій Git широко використовується в галузі програмного забезпечення для спільної роботи над проектами, розробки нових функцій та виправлення помилок. Вона дозволяє розробникам ефективно співпрацювати, відстежувати зміни, вирішувати конфлікти, повертатися до попередніх версій коду та зберігати цінну історію проекту. Git забезпечує стабільність, надійність та гнучкість у керуванні версіями, що робить його незамінним інструментом для розробників у всьому світі.

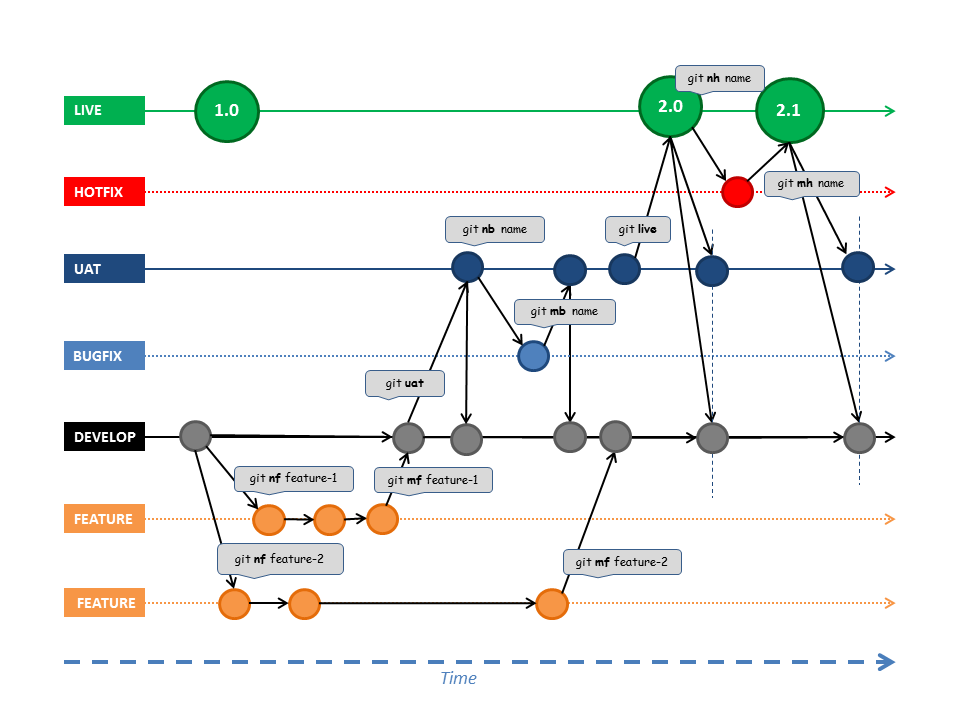


Рис. 2.10. Схема робочого процесу системи контроля версій Git

**Віддалені репозиторі**ї є важливою складовою систем контролю версій (СКВ) і використовуються для зберігання та спільної роботи над проектами. Ось кілька причин, для чого потрібні віддалені репозиторії:

* Збереження даних: Віддалені репозиторії служать як централізовані сховища для зберігання всіх версій файлів проекту. Вони дозволяють зберігати всю історію змін, гілки розробки і внесені покращення.
* Резервне копіювання: Віддалені репозиторії забезпечують резервне копіювання даних. Вони зберігають проект на віддаленому сервері, що забезпечує безпеку та відновлення даних у випадку втрати або пошкодження локальних копій.
* Відстеження змін: Віддалені репозиторії зберігають повну історію змін, що дозволяє відстежувати, коли і які зміни були внесені до проекту. Це надає змогу переглядати, порівнювати та відновлювати стан проекту у будь-який момент часу.
* Робота з різними пристроями: Віддалені репозиторії дозволяють розробникам працювати з різних пристроїв і місць. Вони забезпечують доступ до проекту з будь-якого місця з підключенням до Інтернету, що дає змогу зручно працювати в команді або над проектом з віддаленого місця.
* Контроль версій і версіонування: Віддалені репозиторії дозволяють контролювати версії проекту і внесені зміни. Вони забезпечують можливість відновлення попередніх версій файлів, відкату змін і вирішення конфліктів.

На момент написання роботи існують два найпопулярніших сервіси, які дозволяють зберігати віддалено проєкти. Це GitHub та GitLab.

1). **GitHub** - це веб-платформа для зберігання, спільної роботи та керування проектами розробки програмного забезпечення з використанням системи контролю версій Git. Вона надає можливість розробникам зберігати свій код, спілкуватися, спільно працювати над проектами, вносити зміни та контролювати версії файлів. GitHub також надає функціонал для перегляду, обговорення та перегляду змін коду, а також використовується як платформа для спільного внесення внесків до відкритих проектів. Він широко використовується спільнотою розробників для співпраці та спільної роботи над проектами будь-якого розміру.

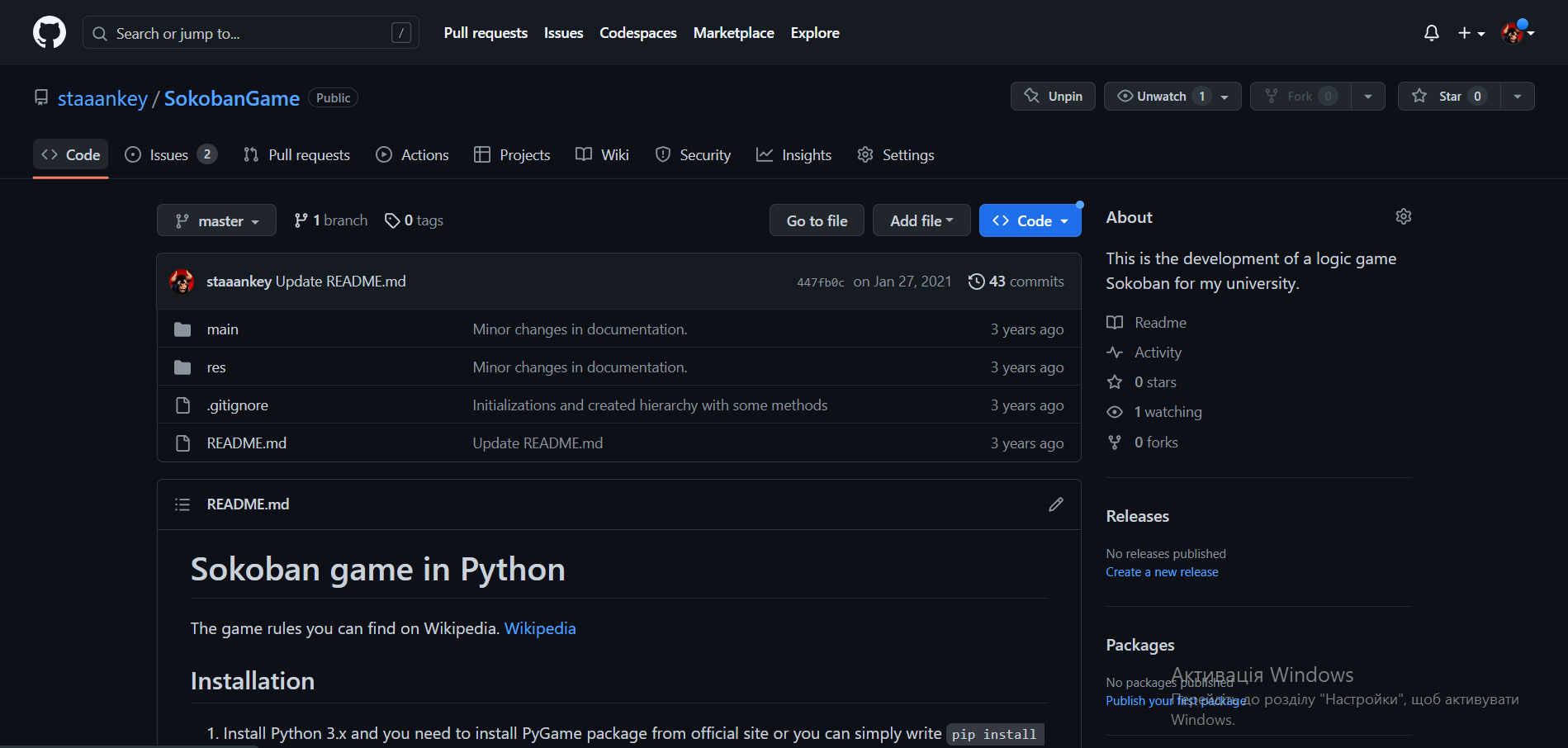


Рис. 2.11. Приклад інтерфейсу платформи GitHub

2). **GitLab** - це веб-платформа для керування проектами розробки програмного забезпечення, яка забезпечує зберігання коду, контроль версій і спільну роботу команди. GitLab пропонує функціонал системи контролю версій Git, а також включає інші інструменти, такі як інтеграція CI/CD, управління задачами, спринтами та інші функції, які допомагають ефективно керувати проектами. Він може бути встановлений як локальний сервер або використовуватися на хмарній платформі GitLab.com. GitLab є відкритим програмним забезпеченням, що надає можливість самостійно використовувати його для керування розробкою програмного забезпечення власними командами.

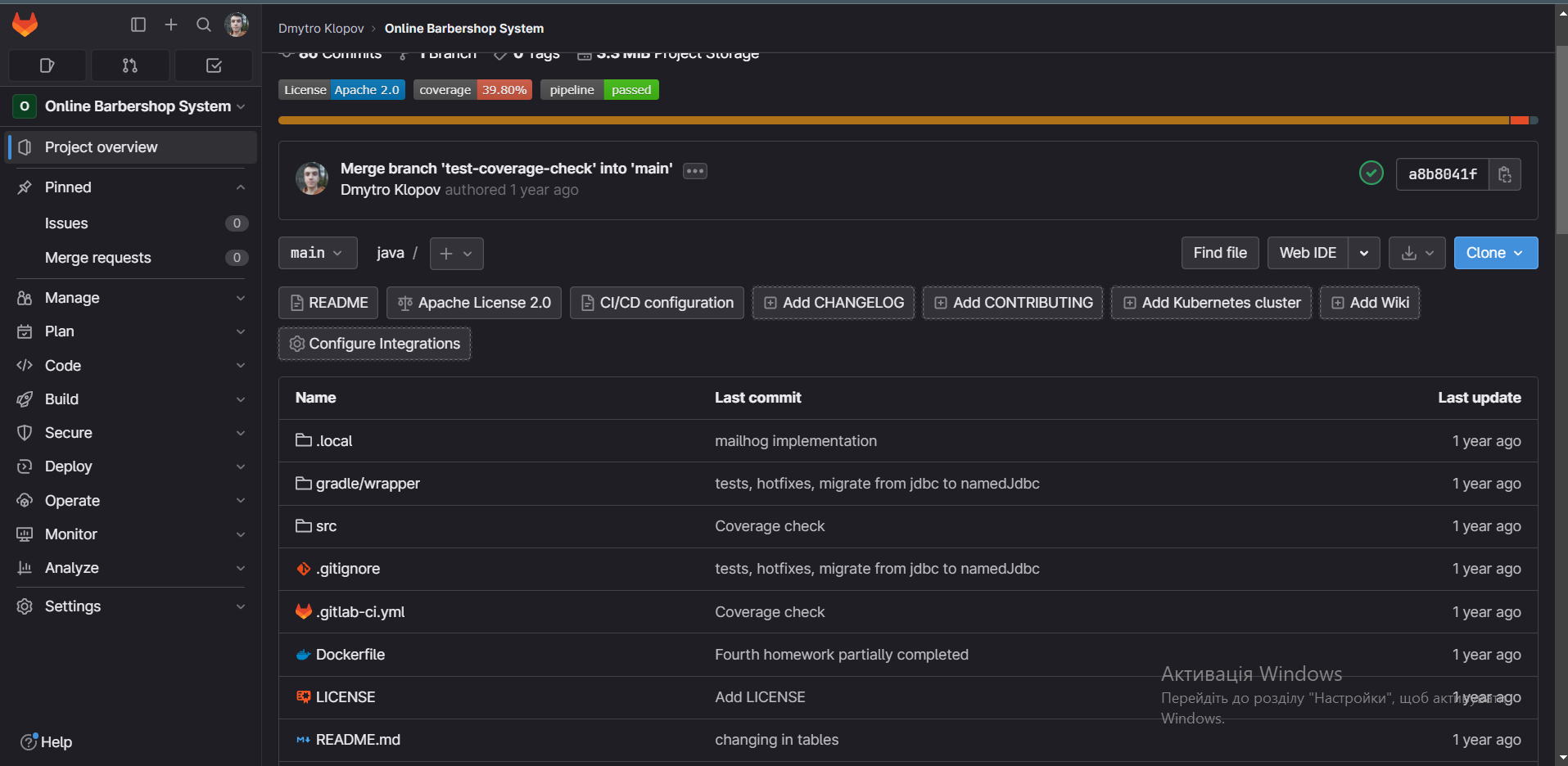


Рис. 2.12. Приклад інтерфейсу платформи GitLab

*Таблиця 2.4*

**Таблиця порівння можливостей сервісів GitHub та GitLab**

|  | GitHub | GitLab |
| --- | --- | --- |
| Хостинг | Хмарний сервіс | Може бути самостійно розгорнутий або хмарний |
| Ціноутворення | Безкоштовні та платні плани | Безкоштовні та платні плани |
| Співпраця | Підтримка публічних та приватних репозиторіїв | Підтримка публічних та приватних репозиторіїв |
| Відстеження проблем | Вбудована система відстеження проблем | Вбудована система відстеження проблем |
| Інтеграція CI/CD | Інтеграція з різними інструментами CI/CD | Інтеграція з CI/CD-пайплайнами |
| Варіанти розгортання | Підтримка GitHub Pages для статичних веб-сайтів | Вбудовані можливості розгортання |
| Спільнота та екосистема | Велика спільнота користувачів та проекти з відкритим кодом | Ростуча спільнота користувачів та проекти з відкритим кодом |
| Розширюваність | Підтримка широкого спектру інтеграцій | Підтримка широкого спектру інтеграцій |
| Інтерфейс користувача | Сучасний та зручний інтерфейс | Функціональний та налаштовуваний інтерфейс |

Після порівняння GitHub та GitLab можна зробити такі висновки:

* Хостинг: Обидва сервіси надають можливість розміщення репозиторіїв в хмарі. Однак GitLab також може бути самостійно розгорнутий на власному сервері або хмарному середовищі.
* Ціноутворення: Обидва сервіси мають безкоштовні та платні плани, що дозволяє вибрати підходящий варіант залежно від потреб проєкту.
* Співпраця: GitHub та GitLab підтримують як публічні, так і приватні репозиторії, що дозволяє працювати як над відкритими проектами, так і в команді.
* Відстеження проблем: Обидва сервіси мають вбудовані системи відстеження проблем, що спрощує управління задачами та співпрацю в команді.
* Інтеграція CI/CD: GitHub та GitLab підтримують інтеграцію з різними інструментами для безперервної інтеграції та постачання програмного забезпечення.
* Варіанти розгортання: GitHub підтримує GitHub Pages для розгортання статичних веб-сайтів, тоді як GitLab має вбудовані можливості розгортання, що дозволяють запускати веб-додатки.
* Спільнота та екосистема: Обидва сервіси мають велику спільноту користувачів та активно розвиваються проекти з відкритим кодом.
* Розширюваність: Як GitHub, так і GitLab підтримують широкий спектр інтеграцій з іншими інструментами та послугами.
* Інтерфейс користувача: GitHub та GitLab пропонують зручний та функціональний інтерфейс, проте GitLab також надає можливість налаштовування інтерфейсу залежно від потреб користувача.
* Керування доступом: Обидва сервіси дозволяють налаштовувати рівні доступу до репозиторіїв та керувати правами користувачів.

Хоча GitHub та GitLab мають багато спільного, вони також мають деякі відмінності, які можуть залежати від конкретних потреб і уподобань проєкту.

* 1. **Continuous Integration (CI) та Continuous Delivery/Deployment (CD)**

Пайплайни є невід'ємною частиною сучасного розробницького процесу. Вони використовуються для автоматизації та прискорення процесу розробки, тестування, постачання та розгортання програмного забезпечення. Основна мета CI/CD пайплайнів полягає в забезпеченні швидкої та надійної доставки якісного програмного забезпечення до кінцевих користувачів.

Основні причини використання CI/CD пайплайнів:

* **Автоматизація:** CI/CD пайплайни дозволяють автоматизувати процеси, що раніше вимагали ручного втручання, такі як збірка, тестування та розгортання програмного забезпечення. Це дозволяє зменшити ризики помилок та забезпечує стабільність розробки.
* **Інтеграція коду:** CI/CD пайплайни дозволяють командам розробників інтегрувати свій код на регулярній основі. Це сприяє виявленню конфліктів та проблем злиття коду, що допомагає забезпечити високу якість та цілісність програмного забезпечення.
* **Швидка збірка та тестування:** CI/CD пайплайни дозволяють швидко збирати та тестувати код. Це дозволяє виявляти проблеми в ранніх етапах розробки та швидко вносити необхідні виправлення.
* **Постачання та розгортання:** CI/CD пайплайни дозволяють автоматично постачати та розгортати програмне забезпечення на виробничі середовища. Це дозволяє прискорити час від ідеї до випуску продукту та забезпечує безперервну доставку нових функцій та виправлень користувачам.
* **Тестування якості:** CI/CD пайплайни дозволяють виконувати різноманітні тести, такі як функціональні тести, тести безпеки, навантажувальні тести та інші. Це допомагає забезпечити якість програмного забезпечення та попередити появу помилок на продуктивних середовищах.
* **Відстеження змін:** CI/CD пайплайни зберігають історію змін коду та результати виконання тестів. Це дозволяє команді відстежувати та аналізувати зміни, розуміти причини помилок та шукати способи поліпшення процесу розробки.

Використання CI/CD пайплайнів дозволяє підвищити ефективність, швидкість та надійність розробки програмного забезпечення. Вони стають невід'ємною складовою сучасного розробницького процесу та сприяють постійному вдосконаленню проектів.

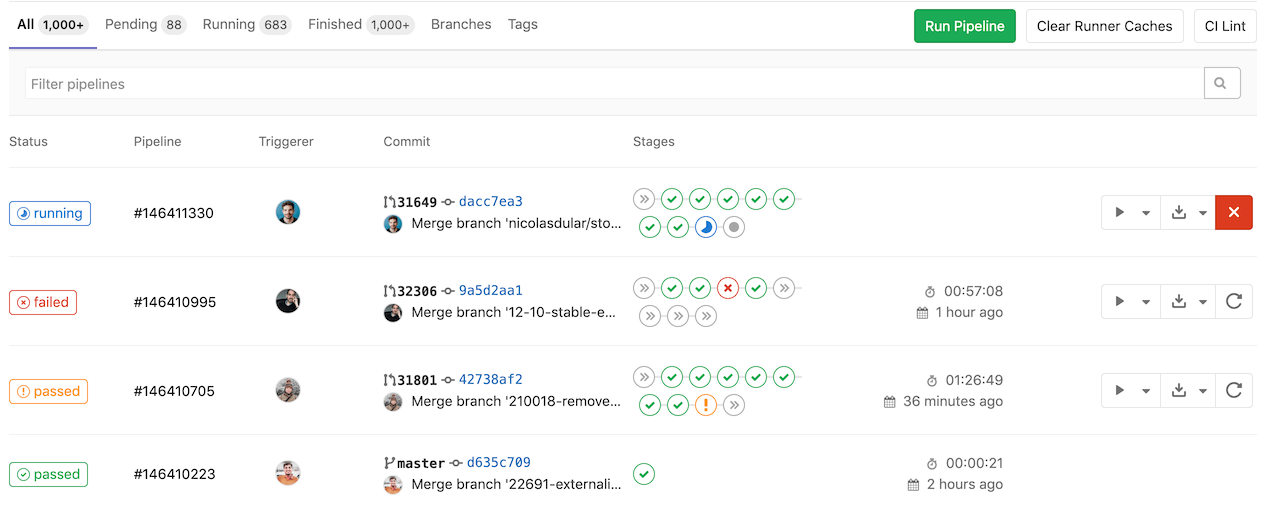


Рис. 2.13. Приклад роботи пайплайнів у середовищі GitLab

* 1. **Огляд хостингів**

Хостинг є невід'ємною складовою сучасного Інтернету і виконує важливу роль у розміщенні та доступі до веб-сайтів, додатків та інших онлайн-ресурсів. Основна мета хостингу - забезпечити інтернет-користувачам можливість завантажувати, зберігати і отримувати доступ до своїх веб-сторінок, файлів і додатків через Інтернет.

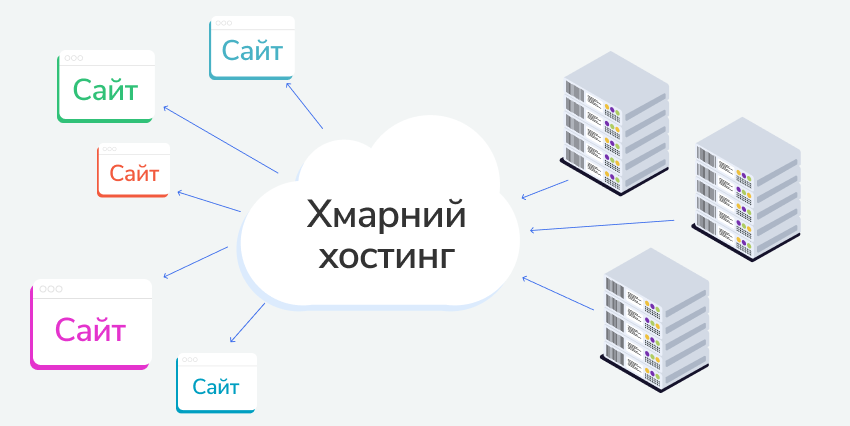


Рис. 2.14. Принцип роботи хмарного хостингу

**Основновними функціями хостингу** є:

* **Зберігання веб-сайтів**: Хостинг надає фізичне місце для зберігання файлів, даних та вмісту веб-сайту. Він забезпечує безпечне та надійне зберігання всіх ресурсів, що потрібні для роботи веб-сайту.
* **Публікація веб-сайту:** Хостинг дозволяє розмістити веб-сайт на сервері, зробити його доступним для користувачів Інтернету і публікувати його адресу (URL).
* **Надання ресурсів:** Хостинг забезпечує необхідні ресурси, такі як обчислювальна потужність, пам'ять, мережевий доступ та пропускна здатність, для роботи веб-сайту та його взаємодії з користувачами.
* **Доступність:** Хороший хостинг гарантує високу доступність веб-сайту, щоб користувачі могли зайти на сайт у будь-який час і без перебоїв.
* **Безпека:** Хостинг забезпечує захист веб-сайту від шкідливих програм, хакерських атак та несанкціонованого доступу. Він надає захист інформації, збереженої на сервері, і забезпечує резервне копіювання даних.
* **Технічна підтримка:** Багато хостинг-провайдерів надають технічну підтримку для вирішення проблем та відповіді на запитання користувачів.
* **Масштабованість:** Хостинг дозволяє розширювати ресурси та масштабувати веб-сайт залежно від потреб користувача. Він може підтримувати зростання відвідуваності та обсягу даних.

*Табл. 2.5*

**Порівняльна характеристика найбільших в світі постачальників хостингових послуг**

| Характеристика | AWS | Google Cloud Platform | Microsoft Azure |
| --- | --- | --- | --- |
| Гнучкість та масштабованість | Підтримка широкого спектру послуг і можливість масштабування | Різноманітні сервіси та гнучкість масштабування | Широкий спектр послуг та гнучкість масштабування |
| Ціна | Різноманітність планів та опцій, гнучка система ціноутворення | Конкурентні ціни та можливість заощадження | Гнучка система ціноутворення та можливість заощадження |
| Доступність | Велика географічна присутність та висока доступність | Глобальна інфраструктура та висока доступність | Глобальна мережа та висока доступність |
| Безпека | Високий рівень безпеки та широкі можливості контролю доступу | Можливості безпеки та контролю доступу | Широкі можливості безпеки та контролю доступу |
| Компанійна підтримка | Широка мережа партнерів та професійна підтримка | Професійна підтримка та консультації | Широка мережа партнерів та професійна підтримка |
| Інтеграція | Широкий спектр сервісів і можливість інтеграції з іншими рішеннями | Можливості інтеграції з іншими продуктами Google | Широкі можливості інтеграції з іншими продуктами Microsoft |

* 1. **Вибір бази данних**

Система електронного документообігу є складним і багатофункціональним інструментом, який дозволяє управляти та обробляти документи в електронному форматі. Ця система зберігає велику кількість даних, включаючи документи, метадані, інформацію про користувачів та їх права доступу.

Для ефективної роботи з документами і забезпечення безпеки та надійності, система електронного документообігу потребує бази даних. База даних виступає як центральне сховище для зберігання всіх даних, пов'язаних з документами.

Основна роль бази даних у системі електронного документообігу включає:

* Зберігання документів: База даних забезпечує місце для зберігання самого тексту документів, включаючи файли, метадані та іншу важливу інформацію.
* Керування версіями: База даних дозволяє відстежувати і керувати різними версіями документів, що дозволяє користувачам відновлювати попередні версії, вносити зміни та виконувати інші операції пов'язані з версіонуванням.
* Керування доступом: База даних зберігає інформацію про права доступу користувачів до документів. Вона контролює, які користувачі мають право переглядати, редагувати або видаляти документи, забезпечуючи безпеку та обмеження доступу до конфіденційної інформації.
* Пошук та фільтрація: База даних дозволяє виконувати розширений пошук та фільтрацію документів за різними параметрами, що допомагає знаходити потрібні документи швидко та ефективно.
* Аналітика та звітність: Збережені дані в базі даних можуть бути використані для створення різноманітних звітів та аналітичних даних, що надають корисну інформацію про стан та ефективність системи електронного документообігу.

*Таблиця 2.6*

**Порівняльна характеристика найрозповсюдженіших баз данних**

| **Назва бази даних** | **Тип бази даних** | **Мова запитів** | **Масштабованість** | **Можливості реплікації** | **Підтримка транзакцій** | **Відкритий код** |
| --- | --- | --- | --- | --- | --- | --- |
| MySQL | Реляційна | SQL | Добра | Є | Є | Так |
| PostgreSQL | Реляційна | SQL | Добра | Є | Є | Так |
| Oracle | Реляційна | SQL | Висока | Є | Є | Ні |
| MongoDB | Документна | MongoDB Query Language (MQL) | Добра | Ні | Ні | Так |
| Redis | Ключ-значення | Redis Commands | Добра | Ні | Ні | Так |
| Cassandra | Колоночна | CQL | Висока | Є | Ні | Так |
| SQLite | Реляційна | SQL | Низька | Ні | Є | Так |

З огляду на порівняльні характеристики, вибір PostgreSQL для проекту може бути обґрунтований його відкритістю, безкоштовністю, розширеною функціональністю, масштабованістю, надійністю та підтримкою спільноти. Він є потужним інструментом для зберігання, опрацювання та управління даними проєкту.

* 1. **Висновок за розділом**

В процесі обґрунтування вибору програмного забезпечення для розробки було враховано кілька факторів, що впливають на ефективність, продуктивність та якість роботи команди розробників. Після аналізу доступних варіантів було прийнято рішення вибрати конкретні інструменти та технології, що найкраще задовольняють вимоги проекту.

В результаті дослідження було обрано фреймворк Spring для розробки системи документообігу університету. Цей вибір зумовлений рядом переваг, які надає Spring, таких як широкі можливості управління залежностями, інтеграція з різноманітними технологіями та добре встановлений стандарт у галузі розробки на базі Java. Крім того, Spring пропонує модульну архітектуру, що дозволяє легко розширювати функціональність системи та забезпечує зручний інструментарій для реалізації різних бізнес-логік.

Також, в процесі вибору програмного забезпечення для розробки, було вирішено використовувати базу даних PostgreSQL. Цей вибір зумовлений високою продуктивністю, розширеними можливостями та надійністю, які надає PostgreSQL. Вона підтримує розширення SQL, має потужні механізми оптимізації запитів та забезпечує надійність та цілісність даних.

Додатково, вибір інтегрованого середовища розробки (IDE) також був обґрунтований. У даному випадку, було вирішено використовувати IntelliJ IDEA, оскільки воно надає потужний набір інструментів для розробки на Java, має зручний інтерфейс та велику спільноту користувачів, що забезпечує підтримку та поради в разі потреби.

В цілому, обґрунтування вибору програмного забезпечення для розробки було проведено на основі аналізу потреб проекту, порівняння доступних варіантів та врахування ключових факторів, таких як функціональність, продуктивність, надійність та спільнота користувачів. Цей вибір має вплив на подальший успіх та якість розробки системи документообігу університету.

**РОЗДІЛ 3.   
РОЗРОБКА CRM СИСТЕМИ УНІВЕРСИТЕТУ**

* 1. **Ініціалізація та перший запуск**

Для первинного створення проєкту на базі фреймворку Spring у програмному середовищі IntelliJ IDEA існує компонент Spring Initializr, який налаштовує первинні зв’язки та підключає основні залежності для першого запуску проєкту. Spring Initializr у IntelliJ IDEA дозволяє швидко створювати нові проекти на основі Spring з відповідними налаштуваннями та залежностями. Це спрощує процес початку роботи з фреймворком Spring та дозволяє швидко налаштувати проект залежно від вашої потреби.

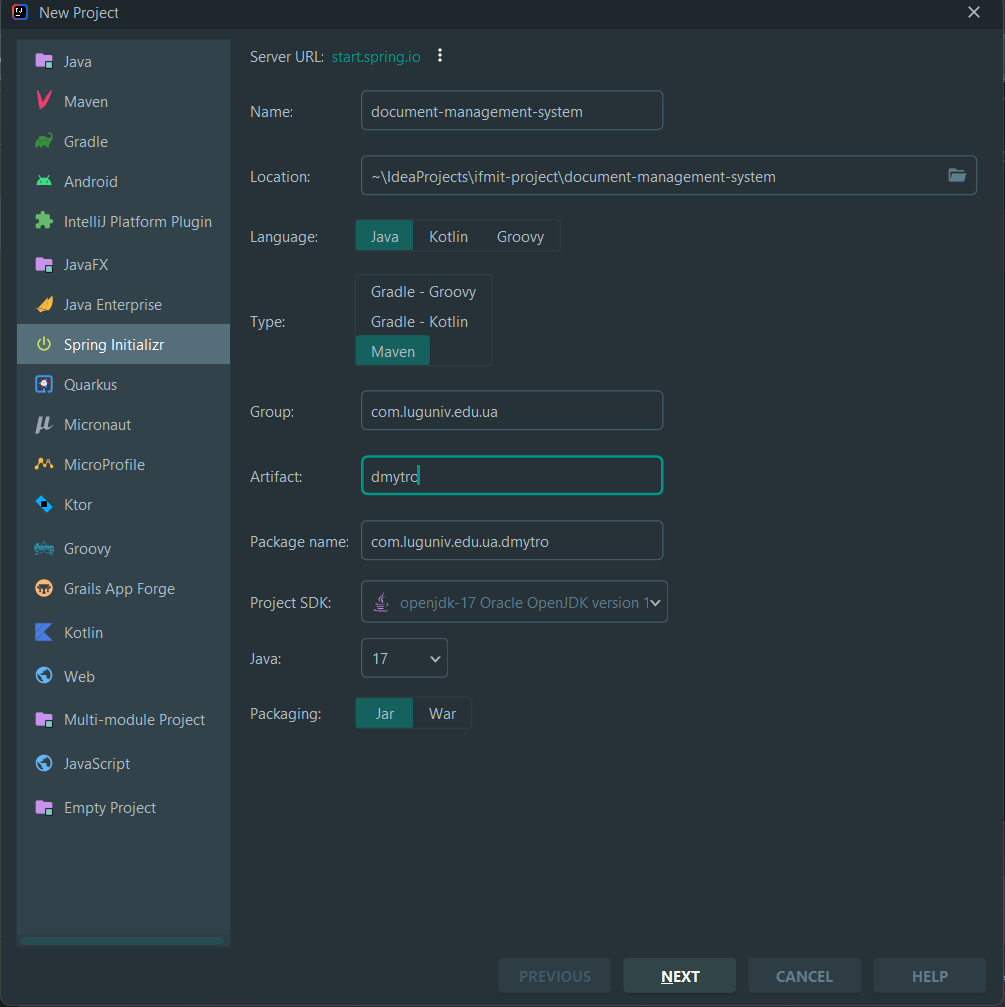


Рис. 3.1. Використання компоненту Spring Initializr

Після створення проєкту необхідно підключити модуль Spring Web для того, щоб запустити локальний Web-сервер Apache Tomcat та перевірити роботу додатку в браузері. Щоб підключити модуль Spring Web до проекту Gradle, слід дотриматися наступних кроків:

1.Відкрити файл build.gradle вашого проекту.

2. Знайти секцію dependencies у файлі build.gradle.

3. Додати залежність до модулю Spring Web, використовуючи наступний код:

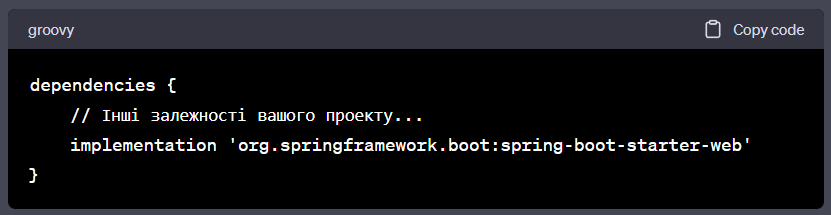


Рис. 3.2. Підключення залежності модулю Spring Web



Рис. 3.3 Контролер привітання з користувачем

Цей код також використовує анотацію @Controller для позначення класу як компонента Spring, що обробляє HTTP-запити. Метод greeting використовує анотацію @GetMapping для визначення шляху /greeting та HTTP-методу GET для обробки запиту. Він приймає параметр name, який передається через URL-параметр. За допомогою об'єкта Model додається атрибут name зі значенням параметра name, і метод повертає рядок "greeting", який буде використовуватися як ім'я шаблону для відображення сторінки.

В результаті проведених дій, якщо звернутися до адресної строки браузера зі шляхом localhost:8080 та параметром сервер відповідає HTML-сторінкою привітання користувача з введеним параметром.

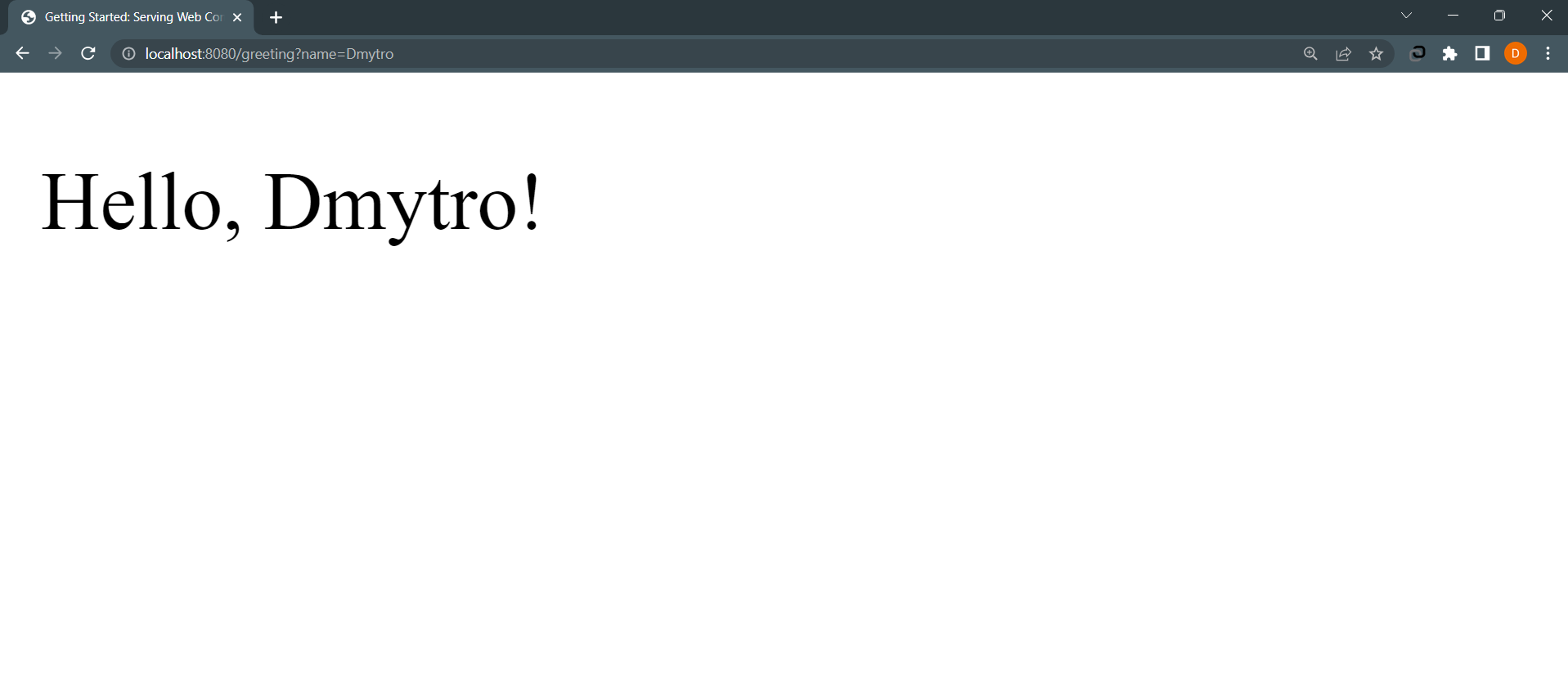


Рис. 3.4. Відповідь серверу

* 1. **Створення та підключення бази данних, налаштування міграцій.**

Оскільки під час аналізу баз даних для проєкту була обрана був обраний PostgreSQL необхідно використати сервер, на якому ця база даних буде запущена. Щоб створити базу даних на сервері PgAdmin 4 та додати її до IntelliJ IDEA, слід дотримуватися наступних кроків:

1. Відкрити PgAdmin 4 і підключитись до вашого сервера баз даних PostgreSQL.

2. В лівій панелі PgAdmin 4 вибрати сервер баз даних та натиснути правою кнопкою миші на розділ "Databases". Обрати "Create" та "Database".

3. У вікні створення бази даних введсти ім'я бази даних, наприклад, "document-managemnt system", та налаштувати потрібні параметри, такі як кодування та шаблон.

4. Натиснути кнопку "Save”, щоб створити базу даних.

Тепер, щоб додати цю базу даних до проекту в IntelliJ IDEA, необіхдно наступні кроки:

5.Відкрите проект в IntelliJ IDEA та відкрити файл "application.properties"

6. Додати наступні рядки коду для налаштування підключення до бази даних

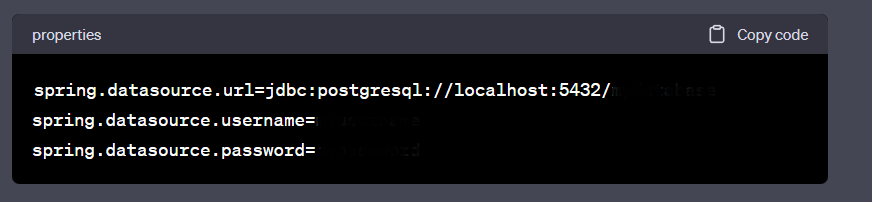


Рис. 3.5. Властивості для підключення бази даних

* **spring.datasource.url:** Цей параметр визначає URL-адресу бази даних PostgreSQL. У даному випадку, jdbc:postgresql://localhost:5432/ означає, що база даних розташована на локальному сервері з портом 5432.
* **spring.datasource.username:** Цей параметр визначає ім'я користувача бази даних, до якого буде встановлено з'єднання.
* **spring.datasource.password:** Цей параметр визначає пароль користувача бази даних, до якого буде встановлено з'єднання.

Міграції баз даних є процесом керування версіями структури бази даних. Вони дозволяють контролювати та автоматизувати зміни в структурі бази даних з плином часу. Основна ідея міграцій полягає в тому, щоб зберігати зміни в структурі бази даних у вигляді коду та застосовувати їх покроково при розгортанні або оновленні додатка.

Flyway - це один з популярних інструментів для керування міграціями баз даних. Він дозволяє вам створювати та застосовувати міграції з використанням SQL-скриптів або Java-коду. Це дозволяє зручно та безпечно оновлювати структуру бази даних.

Для підключення Flyway потрібно виконати наступні кроки:

* Додати залежність Flyway до файлу конфігурації проекту (наприклад, pom.xml для Maven або build.gradle для Gradle).
* Створити директорію для міграцій баз даних у проекті (зазвичай названу db/migration).
* Створити SQL-скрипти або Java-код для кожної міграції бази даних. Скрипти повинні бути названі згідно з певною конвенцією, щоб Flyway міг визначити порядок їх застосування.
* Налаштувати конфігурацію Flyway, вказавши деталі з'єднання до бази даних та шлях до директорії міграцій.

При розгортанні або оновленні додатка, Flyway автоматично зчитує міграції з директорії та застосовує їх до бази даних в потрібному порядку. Він відстежує використані міграції та зберігає історію в базі даних, щоб не застосовувати їх повторно.

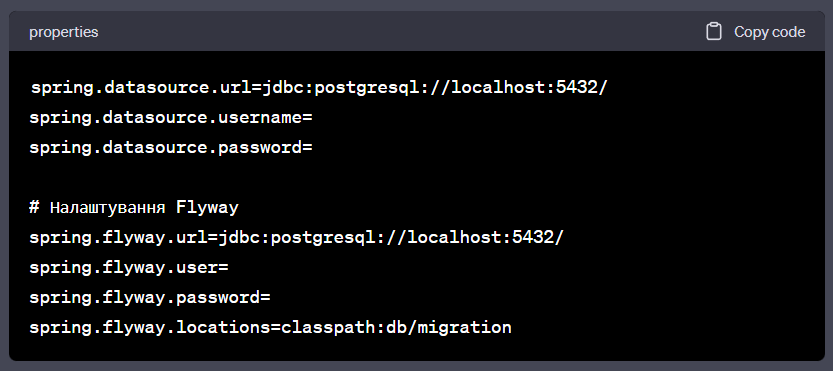


Рис. 3.6. Налаштування міграцій Flyway

* **spring.flyway.url:** Встановлює URL-адресу бази даних для Flyway. Це повинно бути таке саме значення, як у spring.datasource.url.
* **spring.flyway.user:** Встановлює ім'я користувача бази даних для Flyway. Це повинно бути таке саме значення, як у spring.datasource.username.
* **spring.flyway.password:** Встановлює пароль користувача бази даних для Flyway. Це повинно бути таке саме значення, як у spring.datasource.password.
* **spring.flyway.locations:** Встановлює шлях до директорії міграцій. У даному випадку, міграції знаходяться у classpath:db/migration.

Після встановлення цих параметрів, ваш проект буде підключено до бази даних PostgreSQL та буде використовувати Flyway для керування міграціями бази даних.

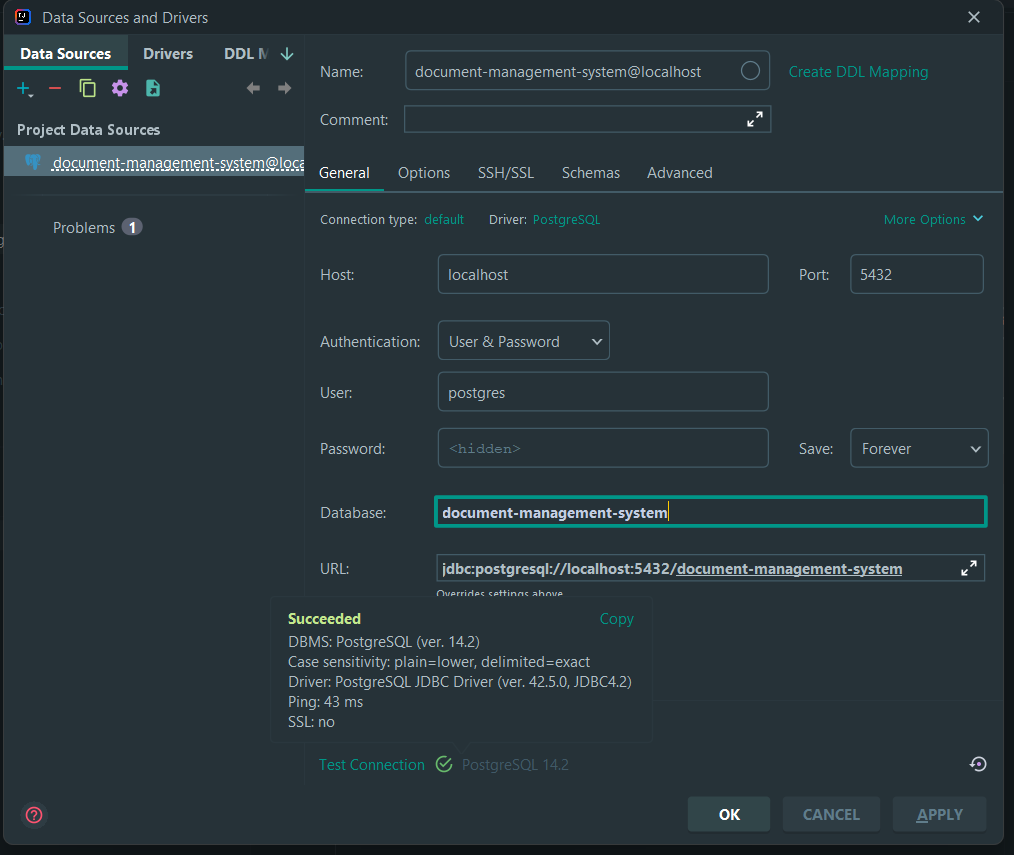


Рис. 3.7. Створення Data Source

* 1. **Репозиторії, сервіси, контролери**

У розробці програмного забезпечення застосовуються певні архітектурні підходи, такі як MVC (Model-View-Controller) або подібні, для організації логіки та функціональності додатку. У цьому контексті репозиторії, сервіси та контролери виконують важливі ролі.

* Репозиторії використовуються для доступу до даних і взаємодії з базою даних. Вони забезпечують абстракцію над роботою з базою даних і надають методи для створення, збереження, оновлення та видалення об'єктів даних. Репозиторії забезпечують розділення логіки доступу до даних від решти додатку.
* Сервіси містять бізнес-логіку додатку. Вони виконують операції над даними, оброблюють бізнес-правила і взаємодіють з репозиторіями для отримання необхідних даних. Сервіси забезпечують гнучкість і модульність додатку, оскільки вони концентруються на конкретних функціональних аспектах і можуть використовуватися в різних контекстах.
* Контролери призначені для обробки запитів і відправки відповідей. Вони приймають HTTP-запити від клієнта, взаємодіють з сервісами для обробки бізнес-логіки і повертають відповіді у вигляді даних або представлень. Контролери визначають маршрутизацію і логіку відображення даних на стороні сервера.

Для функції реєстрації документа у електронній базі необхідно використати усі компоненти архітектурного підходу **MVC.** Для реєстрації документу в ЕСД необхідний репозиторій, який буде відповідати за процес додавання документу до бази данних.

@Override  
public Integer save(Document document) {  
 KeyHolder keyHolder = new GeneratedKeyHolder();  
 final String SQL\_INSERT = "insert into documents (type, body\_id, created\_at, created\_by) values (:type, :body\_id, :created\_at, :created\_by)";  
 jdbcTemplate.update(SQL\_INSERT, new MapSqlParameterSource()  
 .addValue("type", document.getType())  
 .addValue("body\_id", document.getBodyId())  
 .addValue("created\_at", document.getCreatedAt())  
 .addValue("created\_by", document.getCreatedBy()), keyHolder, new String[] {"document\_id"});  
 return keyHolder.getKey().intValue();  
}

Рис. 3.8. Метод save репозиторію документів

Код демонструє метод save, який перезаписує метод інтерфейсу. Анотація @Override вказує, що цей метод перевизначає метод з інтерфейсу.

У цьому методі виконується збереження об'єкта Document в базу даних. Для цього використовується об'єкт jdbcTemplate, який дозволяє виконувати SQL-запити до бази даних.

SQL-запит вставки insert виконується з використанням параметрів. Значення параметрів замінюються на відповідні значення з об'єкта document. Ключовим моментом є використання об'єкта MapSqlParameterSource, який дозволяє передати значення параметрів в запит.

Після виконання запиту, отримане значення згенерованого ключа зберігається в keyHolder, щоб повернути його як результат методу.

Таким чином, цей код здійснює збереження об'єкта Document в базу даних і повертає згенерований ідентифікатор запису.

**Створення контролеру**, для обробки HTTP-запитів.

@PostMapping("/Document")  
public DocumentResultDto registerDocument(@RequestBody DocumentDto documentDto) {  
 var resultDto = new DocumentResultDto();  
 BeanUtils.copyProperties(documentDto, resultDto);  
 resultDto.setDocument\_id(documentService.register(convertToEntity(documentDto)));  
 return resultDto;  
}

Рис. 3.9. Метод registerDocument класу контролеру

У цьому коді оголошено метод registerDocument, який обробляє POST-запит на шляху "/Document". Анотація @PostMapping позначає, що метод має бути викликаний тільки при POST-запиті.

Метод отримує об'єкт documentDto з тіла запиту, використовуючи анотацію @RequestBody. Цей об'єкт містить дані, які були передані у форматі JSON, і він автоматично прив'язується до параметра documentDto.

Далі створено об'єкт resultDto типу DocumentResultDto, який буде використовуватися для повернення результату. Використовуючи BeanUtils.copyProperties, копійовано властивості з documentDto в resultDto.

Потім викликано метод register сервісу documentService, передаючи йому об'єкт, що був отриманий з documentDto. Результат цього методу, який представляє ідентифікатор зареєстрованого документа, встановлюється в resultDto.

На останок, повернуто об'єкт resultDto, який буде перетворений у відповідь у форматі JSON.

@Entity(name = "documents")  
public class Document {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private int document\_id;  
  
 @Column(name = "type")  
 private String type;  
  
 @Column(name = "body\_id")  
 private Integer bodyId;  
  
 @Column(name = "created\_at", columnDefinition = "TIME")  
 private LocalDateTime createdAt;  
  
 @Column(name = "created\_by", columnDefinition = "TIME")  
 private Integer createdBy;  
  
 public Document() {  
 }  
  
 public Document(int document\_id, String type, Integer bodyId, LocalDateTime createdAt, Integer created\_by) {  
 this.document\_id = document\_id;  
 this.type = type;  
 this.bodyId = bodyId;  
 this.createdAt = createdAt;  
 this.createdBy = created\_by;  
 }  
  
}

Рис. 3.9. Клас-сутність документу

**Контролер та репозиторій** оперують сутностями, які вони отримують з HTTP-запиту, або потім відправлять на front-end користувачу. Для цього необхідно оголосити клас Document як сутність, використовуючи анотацію @Entity.

Анотація @Entity вказує, що цей клас представляє таблицю в базі даних.

Анотація @Entity(name = "documents") встановлює ім'я таблиці в базі даних як "documents". Це дозволяє Hibernate асоціювати цей клас з відповідною таблицею.

Поля класу Document мапуються на стовпці в таблиці бази даних за допомогою анотацій @Column. Наприклад, поле type мапується на стовпець з назвою "type".

Анотація @Id позначає поле document\_id як первинний ключ, а анотація @GeneratedValue вказує на автоматичне генерування значень для цього поля.

Клас Document також має конструктори, які приймають параметри для ініціалізації полів, а також пустий конструктор. Це дозволяє створювати об'єкти Document із заданими значеннями або без них.

У класі також визначені геттери та сеттери для доступу до полів.

В результаті, клас Document стає моделлю, яка відображає таблицю "documents" в базі даних.

* 1. **Налаштування Docker**

Налаштування Docker для проекту електронної системи документообігу зазвичай включає створення Docker-контейнерів для бази даних, додатків та інших необхідних компонентів.

version: "3.9"  
  
services:  
 db:  
 image: postgres  
 restart: always  
 ports:  
 - "5432:5432"  
 environment:  
 POSTGRES\_PASSWORD: 123  
 POSTGRES\_DB: "document-management-system"  
 volumes:  
 - ./log-directory:/var/lib/postgresql/document-management-system  
  
 app:  
 build: ../  
 restart: always  
 ports:  
 - "8081:8080"  
 environment:  
 - DATASOURCE\_HOST=db  
 - DATASOURCE\_PORT=5432  
 depends\_on:  
 - db  
 minio:  
 image: minio/minio:latest  
 command: server --console-address ":9001" /data/  
 ports:  
 - "9000:9000"  
 - "9001:9001"  
 environment:  
 MINIO\_ROOT\_USER: staaankey  
 MINIO\_ROOT\_PASSWORD: random123  
 volumes:  
 - minio-storage:/data

Рис. 3.10. Файл docker-compose.yml

Цей код є частиною файлу конфігурації Docker Compose і використовується для опису контейнерів, які будуть розгортатися і працювати у середовищі Docker.

Основні елементи коду:

* version: "3.9": Вказує на версію Docker Compose, яку використовуємо. У цьому випадку використовується версія 3.9.
* services: Вказує на початок опису послуг, які будуть розгортатися у контейнерах.
* db: Описує контейнер для бази даних PostgreSQL. Зазначена імейдж-версія postgres означає, що використовуватиметься офіційний Docker-образ PostgreSQL. restart: always вказує, що контейнер буде автоматично перезапускатися завжди, коли він зупиниться або перезавантажиться. ports визначає маппінг портів, у даному випадку порт 5432 контейнера PostgreSQL мапується на порт 5432 хоста. В environment вказуються змінні середовища, такі як пароль для бази даних (POSTGRES\_PASSWORD) і назва бази даних (POSTGRES\_DB). volumes встановлює маппінг томів між контейнером і хостом, у даному випадку папка log-directory на хості буде мапитися на папку /var/lib/postgresql/document-management-system у контейнері.
* app: Описує контейнер для додатку. build вказує шлях до Docker-файлу для збірки контейнера. restart: always означає автоматичний перезапуск контейнера. ports визначає маппінг портів, у даному випадку порт 8080 контейнера мапується на порт 8081 хоста. environment встановлює змінні середовища, зокрема DATASOURCE\_HOST і DATASOURCE\_PORT, які використовуються для налаштування підключення до бази даних. depends\_on вказує на залежність цього контейнера від контейнера db, що означає, що контейнер db буде запущений перед контейнером app.

Цей файл конфігурації дозволяє запускати контейнери для бази даних PostgreSQL і додатку у середовищі Docker, забезпечуючи ізольоване та портативне розгортання. Залежно від потреб вашого проекту, ви можете налаштувати додаткові параметри контейнерів, наприклад, мережі, обсяги, обмеження ресурсів і т.д.

* 1. **Spring Security**

JWT (JSON Web Token) - це механізм автентифікації та авторизації в Java-додатках, який використовується для безпечної передачі та підтвердження інформації між сторонами. JWT складається з трьох частин: заголовка (header), тіла (payload) та підпису (signature). Заголовок містить інформацію про тип токена та алгоритм шифрування. JWT токени використовуються для забезпечення безпеки і передачі інформації між сторонами без необхідності зберігання стану на сервері. Вони широко використовуються в аутентифікаційних та авторизаційних механізмах, таких як OAuth або власні механізми аутентифікації та авторизації.

@Component

public class TokenManager implements Serializable {

private static final long serialVersionUID = 7008375124389347049L;

public static final long TOKEN\_VALIDITY = 10 \* 60 \* 60;

public static final String AUTHORITIES\_KEY = "scopes";

@Value("${secret}")

private String jwtSecret;

public String generateJwtToken(UserDetails userDetails) {

final String authorities = userDetails.getAuthorities().stream()

.map(GrantedAuthority::getAuthority)

.collect(Collectors.joining(","));

return Jwts.builder()

.claim(AUTHORITIES\_KEY, authorities)

.setSubject(userDetails.getUsername())

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + TOKEN\_VALIDITY \* 1000))

.signWith(SignatureAlgorithm.HS512, jwtSecret).compact();

}

Рис. 3.11. Налаштування JWT Токену

Анотація @Component вказує, що клас TokenManager є компонентом Spring і повинен бути керованим контейнером Spring. Це означає, що Spring створить і керуватиме екземпляром цього класу, і ви зможете використовувати його в інших компонентах або сервісах.

У цьому прикладі, TokenManager відповідає за генерацію JWT токенів. Метод generateJwtToken отримує об'єкт UserDetails, який містить інформацію про користувача (наприклад, ім'я користувача, пароль, ролі тощо). Він використовує цю інформацію для створення JWT токену.

У методі generateJwtToken використовуються класи та методи з пакету io.jsonwebtoken, які дозволяють побудувати JWT токен з необхідними полями, такими як підтвердження (claim), підпис (signature), термін дії (expiration) та інші. Ключ для підпису та перевірки цілісності токену отримується зі значення, яке витягується з конфігураційного файлу за допомогою анотації @Value.

Отриманий JWT токен повертається як рядок, і його можна використовувати для автентифікації та авторизації користувача.

* 1. **Висновок за розділом**

У процесі розробки системи документообігу для університету були враховані основні потреби та вимоги, пов'язані з ефективним керуванням документами в університетському середовищі. Розробка системи була спрямована на полегшення процесів створення, зберігання, обробки та поширення документів в університеті.

Під час розробки було використано сучасні технології, такі як Java, Spring Framework та база даних PostgreSQL. Архітектура системи була побудована з урахуванням принципів модульності, розширюваності та гнучкості, що дозволяє легко вносити зміни та додавати нові функціональні можливості.

Система документообігу включає в себе модулі для створення, збереження, пошуку та обробки документів, а також для автоматизації різних процесів, наприклад, розсилки документів, підписання, затвердження та архівування. Користувачі системи мають можливість виконувати різні дії з документами залежно від їх ролі та прав доступу, що забезпечує контрольований та безпечний доступ до інформації.

Під час розробки системи були враховані принципи безпеки, такі як шифрування даних, механізми автентифікації та авторизації, аудит дій користувачів та інші заходи для захисту конфіденційної інформації. Також було розглянуто можливість інтеграції з іншими системами університету, наприклад, системою управління студентами чи кадровою системою.

**ВИСНОВКИ**

У процесі розробки системи документообігу для університету було проведено дослідження та аналіз потреб користувачів, їх вимог та проблем, пов'язаних з управлінням документами. На основі цього аналізу була спроектована та реалізована система, яка забезпечує ефективне, безпечне та зручне управління документами університетської спільноти.

Були враховані специфічні потреби та вимоги університетського середовища, такі як обмеження доступу до даних, вимоги до зберігання та архівації документів, а також потреба в забезпеченні швидкого та ефективного пошуку та обробки інформації.

Система документообігу університету включає в себе різноманітні функціональні можливості, такі як створення, редагування, видалення та перегляд документів, керування правами доступу до документів, механізми контролю версій та аудиту дій користувачів. Вона також забезпечує можливість автоматичного генерації звітів та статистики, спрощує процеси спільної роботи над документами та сприяє покращенню комунікації між різними структурними підрозділами університету.

Розробка системи документообігу вимагала використання сучасних технологій та інструментів програмування, таких як Java, Spring Framework, бази даних PostgreSQL, Docker для контейнеризації, а також інших допоміжних бібліотек та фреймворків. Були застосовані найкращі практики в галузі розробки програмного забезпечення, зокрема використання шаблонів проектування, принципів SOLID, тестування та безпеки даних.

Результатом розробки системи документообігу став функціональний та ефективний інструмент, який допомагає університетській спільноті ефективно управляти документами, забезпечувати їх безпеку та зручний доступ до них. Реалізація системи документообігу сприяє автоматизації процесів, покращенню ефективності роботи та забезпеченню високого рівня якості та точності документування в університеті.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1.https://en.wikipedia.org/wiki/Document\_management\_system [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Document_management_system>.

2. Spring Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://spring.io/quickstart>.

3. MVC Design Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://ru.wikipedia.org/wiki/Model-View-Controller>.

4. GitLab Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.gitlab.com/>.

5. JetBrains IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: <https://www.jetbrains.com/help/idea/getting-started.html>.

6. Sutton M. Document Management for the Enterprise: Principles, Techniques, and Applications / Michael Sutton., 1996. – 400 с.

7. Smallwood R. Managing Electronic Records: Methods, Best Practices, and Technologies / Robert Smallwood., 2013. – 464 с.

8. Azad A. Implementing Electronic Document and Record Management Systems / Adam Azad., 2007. – 280 с.

9. Wiggins B. Effective Document and Data Management: / Bob Wiggins., 2016. – 266 с.

10. Middleton M. Integrative Document & Content Management: Strategies for Exploiting Enterprise Knowledge / Michael Middleton., 2003. – 526 с

11. Bloch J. Effective Java / Joshua Bloch., 2001 – 312 с.

12. Chan J. Learn Java in One Day and Learn It Well / Jamie Chan., 2016. – 237 с.

13. Koskela L. Test Driven: TDD and Acceptance TDD for Java Developers / Lasse Koskela., 2007. – 470 с.

14. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship / Robert Martin., 2008. – 464 с.

15. Walls C. Spring in Action, Fifth Edition / Craig Walls., 520. – 2018 с.

16. Java Language and Virtual Machine Specifications [Електронний ресурс] // Oracle – Режим доступу до ресурсу: <https://docs.oracle.com/javase/specs/>.

17. Long J. Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry / Josh Long., 2017. – 648 с.

18. Mak G. Spring Enterprise Recipes: A Problem-Solution Approach / Garry Mak., 2009. – 400 с.

19. Guruzu S. Hibernate Recipes: A Problem-Solution Approach / Srinivas Guruzu., 2010. – 312 с.

20. Документація PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org/docs/>.

21. Obe R. PostgreSQL: Up and Running / Regina Obe., 2015. – 213 с.

22. Schönig H. Troubleshooting PostgreSQL / Hans-Jürgen Schönig., 2015. – 164 с.

23. Geschwinde E. PostgreSQL Developer's Handbook / Ewald Geschwinde., 2002. – 753 с.

24. Docker Documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/>.

25. Turnbull J. The Docker Book / James Turnbull., 2014. – 386 с.

26. Pethuru R. Learning Docker / Raj Pethuru., 2015. – 240 с.

27. Cochrane K. Docker Cookbook / Ken Cochrane., 2018. – 352 с.

28. Spring Security [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-security/reference/index.html.

29. Splica L. Spring Security in Action / Laurentiu Splica., 2020. – 560 с.

30. Knutson M. Spring Security - Third Edition / Mick Knutson., 2017. – 542 с.

31. Scarioni C. Pro Spring Security / Carlo Scarioni., 2019. – 410 с.

**ДОДАТОК А**

package com.staaankey.documentmanagementsystem.documents.controllers;  
  
  
import com.staaankey.documentmanagementsystem.documents.controllers.dto.DocumentDto;  
import com.staaankey.documentmanagementsystem.documents.controllers.dto.DocumentResultDto;  
import com.staaankey.documentmanagementsystem.documents.entities.Document;  
import com.staaankey.documentmanagementsystem.documents.services.DocumentServiceImpl;  
import org.springframework.beans.BeanUtils;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class DocumentController {  
 private final DocumentServiceImpl documentService;  
  
 public DocumentController(DocumentServiceImpl documentService) {  
 this.documentService = documentService;  
 }  
  
  
 @PostMapping("/Document")  
 public DocumentResultDto registerDocument(@RequestBody DocumentDto documentDto) {  
 var resultDto = new DocumentResultDto();  
 BeanUtils.copyProperties(documentDto, resultDto);  
 resultDto.setDocument\_id(documentService.register(convertToEntity(documentDto)));  
 return resultDto;  
 }  
  
 private Document convertToEntity(DocumentDto documentDto) {  
 Document document = new Document();  
 BeanUtils.copyProperties(documentDto, document);  
 return document;  
 }  
}

**ДОДАТОК Б**

package com.staaankey.documentmanagementsystem.security;  
  
import com.staaankey.documentmanagementsystem.registration.entities.User;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.util.Collection;  
import java.util.Collections;  
import java.util.List;  
  
public class SecurityUser implements UserDetails {  
 private final String username;  
 private final String password;  
 private final List<GrantedAuthority> authorities;  
 private final boolean active;  
  
 public SecurityUser(  
 String username,  
 String password,  
 List<GrantedAuthority> authorities,  
 boolean active  
 ) {  
 this.username = username;  
 this.password = password;  
 this.authorities = authorities;  
 this.active = active;  
 }  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return authorities;  
 }  
  
 @Override  
 public String getPassword() {  
 return password;  
 }  
  
 @Override  
 public String getUsername() {  
 return username;  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return active;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return active;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return active;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return active;  
 }  
  
 public static UserDetails fromUser(User user) {  
 return new SecurityUser(user.getUsername(), user.getPassword(), Collections.emptyList(), true);  
 }  
}

**ДОДАТОК В**

CREATE TABLE "documents" (  
 "document\_id" serial PRIMARY KEY NOT NULL,  
 "type" varchar(255) NOT NULL,  
 "body\_id" integer NOT NULL,  
 "created\_at" time(255) NOT NULL,  
 "created\_by" integer NOT NULL  
);  
  
CREATE TABLE "users" (  
 "user\_id" serial PRIMARY KEY NOT NULL,  
 "username" varchar(100) NOT NULL,  
 "password" varchar(255) NOT NULL,  
 "full\_name" varchar(255) NOT NULL,  
 "mail" varchar(255) NOT NULL,  
 "role\_id" integer NOT NULL UNIQUE  
);  
  
CREATE TABLE "roles" (  
 "role\_id" serial NOT NULL,  
 "name" varchar(255) NOT NULL  
);  
  
CREATE TABLE "body" (  
 "body\_id" serial PRIMARY KEY ,  
 scans varchar(255) NOT NULL,  
 "uri" varchar(255) NOT NULL,  
 "size" integer NOT NULL,  
 "description" varchar(512) NOT NULL,  
 "status" varchar(512) NOT NULL,  
 "changed\_at" time(255) NOT NULL,  
 "changed\_by" time(255) NOT NULL  
);  
  
  
  
ALTER TABLE "documents" ADD CONSTRAINT "documents\_fk0" FOREIGN KEY ("body\_id") REFERENCES "body" ("body\_id");  
  
ALTER TABLE "documents" ADD CONSTRAINT "documents\_fk1" FOREIGN KEY ("created\_by") REFERENCES "users" ("user\_id");  
  
ALTER TABLE "roles" ADD CONSTRAINT "roles\_fk0" FOREIGN KEY ("role\_id") REFERENCES "users" ("role\_id");

**ДОДАТОК Г**

version: "3.9"  
  
services:  
 db:  
 image: postgres  
 restart: always  
 ports:  
 - "5432:5432"  
 environment:  
 POSTGRES\_PASSWORD: 123  
 POSTGRES\_DB: "document-management-system"  
 volumes:  
 - ./log-directory:/var/lib/postgresql/some-name  
  
 app:  
 build: ../  
 restart: always  
 ports:  
 - "8081:8080"  
 environment:  
 - DATASOURCE\_HOST=db  
 - DATASOURCE\_PORT=5432  
 depends\_on:  
 - db  
 minio:  
 image: minio/minio:latest  
 command: server --console-address ":9001" /data/  
 ports:  
 - "9000:9000"  
 - "9001:9001"  
 environment:  
 MINIO\_ROOT\_USER: staaankey  
 MINIO\_ROOT\_PASSWORD: random123  
 volumes:  
 - minio-storage:/data

**ДОДАТОК Є**

plugins {  
 id 'java'  
 id 'org.springframework.boot' version '2.6.3'  
 id 'io.spring.dependency-management' version '1.1.0'  
}  
  
group = 'com.staaankey'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = '17'  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 implementation 'org.springframework.boot:spring-boot-starter-web'  
 implementation 'org.springframework.boot:spring-boot-starter-jdbc'  
 implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
 implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
 implementation 'org.springframework.boot:spring-boot-starter-security'  
 implementation "io.springfox:springfox-boot-starter:3.0.0"  
 implementation 'org.flywaydb:flyway-core:8.4.4'  
 runtimeOnly('org.postgresql:postgresql:42.3.4')  
 testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}  
  
tasks.named('test') {  
 useJUnitPlatform()  
}

**ДОДАТОК Ж**

package com.staaankey.documentmanagementsystem.registration.repository;  
  
import com.staaankey.documentmanagementsystem.registration.entities.User;  
import com.staaankey.documentmanagementsystem.registration.util.UserRowMapper;  
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;  
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;  
import org.springframework.jdbc.support.GeneratedKeyHolder;  
import org.springframework.jdbc.support.KeyHolder;  
import org.springframework.stereotype.Repository;  
  
import javax.sql.DataSource;  
import java.util.Map;  
import java.util.Optional;  
  
@Repository  
public class UserRepositoryImpl implements UserRepository {  
 private final NamedParameterJdbcTemplate jdbcTemplate;  
 private final UserRowMapper userRowMapper;  
  
 public UserRepositoryImpl(DataSource dataSource, UserRowMapper userRowMapper) {  
 this.jdbcTemplate = new NamedParameterJdbcTemplate(dataSource);  
 this.userRowMapper = userRowMapper;  
 }  
  
  
 @Override  
 public Integer save(User user) {  
 KeyHolder keyHolder = new GeneratedKeyHolder();  
 final String SQL\_INSERT = "insert into users (username, password, mail, role\_id, full\_name) values " +  
 "(:username, :password, :mail, :role\_id, :full\_name)";  
  
 jdbcTemplate.update(SQL\_INSERT, new MapSqlParameterSource()  
 .addValue("username", user.getUsername())  
 .addValue("password", user.getPassword())  
 .addValue("mail", user.getMail())  
 .addValue("role\_id", user.getRole\_id())  
 .addValue("full\_name", user.getFullName()), keyHolder, new String[] {"user\_id"});  
  
 return keyHolder.getKey().intValue();  
  
 }  
  
 public Optional<User> getUser(String username) {  
 var sql = "SELECT \* FROM users WHERE username=:username";  
 Map<String, String> parameters = Map.ofEntries(Map.entry("username", username));  
 return jdbcTemplate.query(sql, parameters, userRowMapper).stream().findFirst();  
 }  
  
  
}

**ДОДАТОК З**

package com.staaankey.documentmanagementsystem.documents.controllers.dto;  
  
import java.time.LocalDateTime;  
  
public class DocumentDto {  
 private String type;  
 private Integer bodyId;  
 private LocalDateTime createdAt;  
 private Integer createdBy;  
  
 public DocumentDto() {  
 }  
  
 public DocumentDto(String type, Integer bodyId, LocalDateTime createdAt, Integer created\_by) {  
 this.type = type;  
 this.bodyId = bodyId;  
 this.createdAt = createdAt;  
 this.createdBy = created\_by;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 public Integer getBodyId() {  
 return bodyId;  
 }  
  
 public void setBodyId(Integer body\_id) {  
 this.bodyId = body\_id;  
 }  
  
 public LocalDateTime getCreatedAt() {  
 return createdAt;  
 }  
  
 public void setCreatedAt(LocalDateTime created\_at) {  
 this.createdAt = created\_at;  
 }  
  
 public Integer getCreatedBy() {  
 return createdBy;  
 }  
  
 public void setCreatedBy(Integer created\_by) {  
 this.createdBy = created\_by;  
 }  
}