КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНИВЕРСИТЕТ

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

**Курсова робота**

За спеціальністю 121 Програмна інженерія

На тему:

**РОЗРОБКА ЕЛЕКТРОННОЇ БІБЛІОТЕКИ**

**ДЛЯ КУРСУ «ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ»**

Виконала студентка 3-го курсу

Нортман Юлія Олександрівна

Науковий керівник:

Професор

Терещенко Василь Миколайович

Засвідчую, що в цій курсовій роботі

немає запозичень з праць інших авторів без

відповідних посилань.

Студент

Київ – 2020

# Реферат

Обсяг роботи ?? сторінок, ?? ілюстрацій, ? таблиць, ?? джерел посилань

АЛГОРИТМ, БАЗА ДАНИХ, ЕЛЕКТРОННА БІБЛІОТЕКА, ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ. HIBERNATE, HTTP, JAVA, REST SERVICE, SPRING FRAMEWORK, SQL

Об’єктом розробки є створення електронної бібліотеки для курсу «Обчислювальна геометрія».

Метою роботи було дослідження алгоритмів побудови опуклої оболонки, знаходження найкоротшого шляху на місцевості з перешкодами та оптимальної тріангуляції.

Для розробки використовувалась Eclipse IDE For Enterprise Java Developers 2019-12, база даних PostgreSQL 12.1, HTTP клієнт Postman для тестування та хмарна PaaS-платформа Heroku. Мова програмування – Java.

У ході роботи були вивчені певні алгоритми обчислювальної геометрії а також був створений програмний продукт «Електронна бібліотека для курсу Обчислювальна Геометрія», який дозволяє переглядати опис алгоритму, відео-демонстрацію, а також підтримує операції додавання, видалення та зміни алгоритму.

За методами розробки та інструментальними засобами робота велась спільно з вивченням архітектури та шаблонів проектування.

Програмний продукт «Електронна бібліотека для курсу «Обчислювальна Геометрія»» може використовуватися під час вивчення курсу Обчислювальна Геометрія студентами вищих навчальних закладів.

У наступних версіях продукту планується додати інтернаціоналізацію та можливість реєстрації акаунту при додаванні коментарів.

**ЗМІСТ**

[Реферат 2](#_Toc41471971)

[Вступ 5](#_Toc41471972)

[теоретична частина 7](#_Toc41471973)

[Фреймворки Spring 5 та Spring Boot 2 7](#_Toc41471974)

[Основи 7](#_Toc41471975)

[Історія та розвиток Spring Framework 7](#_Toc41471976)

[Поява Spring Boot Framework 8](#_Toc41471977)

[Архітектура Spring 9](#_Toc41471978)

[Інверсія Управління та Впровадження Залежностей 10](#_Toc41471979)

[Переваги впровадження залежностей в порівнянні з традиційним підходом 11](#_Toc41471980)

[Різниця між Spring та Spring Boot 13](#_Toc41471981)

[Патерн Model-View-Controller (MVC) 14](#_Toc41471982)

[Основні поняття 14](#_Toc41471983)

[Використання у веб-додатках 15](#_Toc41471984)

[Архітектура Spring MVC 16](#_Toc41471985)

[REST веб-сервіси 17](#_Toc41471986)

[Основні принципи 17](#_Toc41471987)

[HTTP-методи 18](#_Toc41471988)

[Робота з базами даних та бібліотека Hibernate 19](#_Toc41471989)

[Основні поняття 19](#_Toc41471990)

[Рівень доступу до даних 21](#_Toc41471991)

[Практична частина 22](#_Toc41471992)

[Дослідження алгоритмів 22](#_Toc41471993)

[Проектування та створення бази даних. 23](#_Toc41471994)

[Розробка програми 25](#_Toc41471995)

[Розгортання програми [19] 26](#_Toc41471996)

[Перелік джерел посилання 28](#_Toc41471997)

[Додаток А 30](#_Toc41471998)

# Вступ

**Оцінка сучасного стану об’єкта дослідження або розробки.** Необхідність досягнення успіху в сучасному світі диктує серйозні вимоги до якості та швидкості навчання. Підвищення ефективності навчання в сучасних умовах вимагає якісного підходу до подачі матеріалу. У наш час це неможливо без побудови інформаційної системи та, що містить такий матеріал і дозволяє працювати з нею з використанням сучасних засобів роботи з інформацією. Максимально ефективною буде система, що дозволяє здійснювати повний цикл роботи з інформацією, а саме перегляд і пошук інформації, можливість обговорення з колегами та її візуалізація. На відміну від неорганізованого збереження різних програм для перегляду алгоритмів на диску, система має дозволяти побудувати чітку структуру алгоритмів, поділену на категорії, орієнтуватися та шукати у якій стає максимально швидко та просто.

Крім того, важливі питання інформаційної безпеки, які неможливо вирішити при хаотичному збереженні алгоритмів. По-перше, запуск різних програм, які написані різними користувачами є небезпечним та часто й неможливим для роботи у сучасних мережах. В кращому випадку їх не дозволить запустити антивірус, в гіршому – вони призведуть до проблем в роботі операційної системи, і в результаті це призведе до того, що користувачі відмовляться працювати з такими програмами. По-друге, система має дозволяти змінювати інформацію лише тим, хто має дозвіл на такі дії, що й досягається за рахунок сучасних засобів автентифікації та авторизації.

**Актуальність роботи та підстави для її виконання.** При виконанні практичних та лабораторних робіт з курсу «Обчислювальна геометрія» студенти часто не розуміють, як саме має бути виконана робота та на що слід звернути увагу. Крім того, засвоєння будь-якого матеріалу краще за все відбувається при наявності наочних прикладів та розумінні в яких прикладних сферах це може бути застосовано. Тому було вирішено створити електронну бібліотеку, у якій буде зібрано основні алгоритми цього курсу з детальним описом та прикладами використання.

**Мета й завдання роботи.** Метою даної роботи було створення програмного продукту, який дає можливість читати, додавати нові та змінювати вже існуючі алгоритми. Для досягнення цієї мети були поставлені такі задачі.

* Дослідити та структурувати певний набір алгоритмів з області «Обчислювальної геометрії».
* Визначитися зі стеком технологій, які будуть використовуватися при розробці продукту.
* Створити ER-діаграму класів та відповідну базу даних.
* Безпосередня розробка серверної частини програми.
* Знайти підходящу платформу для розгортання і тестування програми.

**Об’єкт, методи і засоби розроблення.** Об’єктом розроблення програмного засобу є електронна бібліотека для вивчення алгоритмів обчислювальної геометрії.

Розробці програмного продукту передувало дослідження алгоритмів побудови опуклих оболонок, знаходження оптимальних шляхів на місцевості з перешкодами та діаграм Вороного.

**Можливі сфери застосування.** Програмний продукт «Електронна бібліотека алгоритмів для курсу Обчислювальна Геометрія» може застосовуватися в навчальному процесі під час вивчення курсу Обчислювальна Геометрія та схожих з ним.

**Взаємозв’язок з іншими роботами.** За методами розробки та інструментальними засобами робота виконувалась сумісно з вивченням певних алгоритмів обчислювальної геометрії.

# Теоретична частина

## Фреймворки Spring 5 та Spring Boot 2

Фреймворк (framework – конструкція, структура) – програмне середовище спеціального призначення, каркас, який використовується для того, щоб істотно полегшити процес об’єднання певних компонент при написанні програм; основа, що дозволяє додавати компоненти в залежності від потреб.

Spring Framework або просто Spring – один із найбільш популярних фреймворків з відкритим кодом для створення проектів на Java. Він призначений для написання будь-якого додатку (наприклад, автономних, веб-застосунків або корпоративних Java Enterprize (JEE) застосунків) на відміну від багатьох інших каркасів, зокрема від каркаса Apache Struts, який призначений лише для створення веб-проектів.

### Основи

#### Історія та розвиток Spring Framework

У жовтні 2002 року Род Джонсон (Rod Jonson) випустив книгу під назвою “Expert One-on-One J2EE Design and Development”. Там описувалося розробка програм на Java Enterprize, та водночас вказувалося на кілька основних недоліків фреймворків Java EE та Enterprize Java Beans (EJB). У своїй книзі Род Джонсон запропонував простіше рішення, яке базувалося на звичайних Java класах (POJO – Plain Old Java Objects) та впровадженню залежностей (Dependency Injection - DI).

“Expert One-on-One J2EE Design and Development” одразу стала бестселером, і що не менш цікаво, через двадцять років після першої публікації принципи, які описані у цій книзі, залишаються актуальними в розробці високоякісних застосунків на Java.

Скоро після виходу книги, у лютому 2003 року, розробники Юрген Хойлер (Juergen Hoeller), Ян Кероф (Yann Caroff) та Род Джонсон вирішили створити проект з відкритим кодом, і вже в червні 2003 року була випущена версія Spring 0.9 під ліцензією Apache 2.0 license. З того часу почався бурхливий розвиток фреймворку.

У жовтні 2006 року була випущена версія Spring 2.0 і до того часу кількість завантажень перевищило відмітку в один мільйон, а у грудні 2009 з’явилася версія Spring 3.0.

У грудні 2013 року Pivotal Inc. анонсувала випуск Spring 4.0, і ця подія стала важливим кроком у розвитку фреймворку. Ця версія охоплювала повну підтримку Java 8, сторонніх бібліотек таких як Groovy 1.8+, Ehcache 2.1+, Hibernate 3.6+), Java EE7, Websockets тощо. Нарешті у 2018 році була випущена версія Spring 5.0, яка включала у себе підтримку Java EE8 та інтеграцію з бібліотеками Reactor 3.1, Junit 5 та підтримку мови Kotlin.

#### Поява Spring Boot Framework

У жовтні 2012 року Майк Янгстром (Mike Youngstrom) запропонував спростити архітектуру веб проекту шляхом конфігурації служб веб контейнера всередині Spring Container та їх завантаження прямо з методу main(). Ця пропозиція призвела до появи Spring Boot 1.0.0 в квітні 2014 року. Простота цього рішення призвела до того, що фреймворк був сприйнятий програмістами та швидко став знаходити застосування серед Java розробників. Spring Boot вважається найшвидшим та найпростішим шляхом для розробки REST мікросервісів. У ньому також наявна підтримка Docker контейнерів.

#### Архітектура Spring

Фреймворк Spring версії 5.0.0.RELEASE складається з двадцяти одного модуля, кожен з яких по суті є звичайним архівним jar-файлом, і які за основними функціями можуть бути розділені на сім основних груп:

1. Основний контейнер (Core Container)
   1. Модуль Core забезпечує основні частини платформи, включаючи інверсію управління (Inversion of Control - IoC) та впровадження залежностей (DI)
   2. Модуль Bean представляє BeanFactory, що являє собою складну реалізацію шаблону Фабрика
   3. Модуль Context – його основою є модулі Core та Bean, та представляє собою середовище для доступу до будь-яких визначених та налаштованих об’єктів. Інтерфейс ApplicationContext є координаційним центром даного модуля.
   4. Модуль SpEL представляє мову виразів для запитів та маніпулювання графом об’єктів під час виконання.
2. Доступ до даних/інтеграція (Data Access/Integration)
   1. Модуль JDBC – представляє рівень абстракції JDBC, який усуває необхідність для розробника вручну прописувати код, пов’язаний з встановленням з’єднання з базою даних.
   2. Модуль ORM – забезпечує інтеграцію з такими популярними ORM-бібліотеками як Hibernate, JDO, JPA.
   3. Модуль OXM – відповідає за зв’язок Об’єкт/XML – XMLBeans.
   4. Модуль JMS(Java Messaging Service) – відповідає за створення, передачу та отримання повідомлень.
   5. Модуль Transactions – підтримує управління транзакціями для класів, які реалізують певні методи.
3. Веб
   1. Модуль Web містить основні класи для застосування Spring у веб-проектах, у тому числі класи для автоматичного завантаження контексту, вивантаження файлів та ряд корисних класів для виконання таких повторюваних завдань, як витяг цілочисельних значень із строки запиту.
   2. Модуль Web-reactive містить базові інтерфейси та класи для моделі реактивного веб-програмування в Spring.
   3. Модуль Web-MVC містить всі класи для каркаса по проектному шаблону MVC.
   4. Модуль Web-Socket забезпечує підтримку двосторонніх зв’язків на основі WebSocket між клієнтом та сервером.
4. Модуль АОП забезпечує реалізацію аспектно-орієнтованого програмування, тобто парадигми, основаної на ідеї розділення функціональності для покращення розбиття програми на модулі.
5. Модуль Instrumentation забезпечує підтримку інструментарію класу та реалізації завантажувача класів.
6. Модуль Обміну Повідомленнями забезпечує підтримку STOMP в якості суб-протоколу WebSocket для використання в проектах.
7. Модуль Test підтримує тестування компонентів Spring за допомогою каркасів Junit та TestNG.

#### Інверсія Управління та Впровадження Залежностей

Ядро каркаса Spring засновано на принципі інверсії управління, коли створення об’єктів та управління ними здійснюється зовнішнім чином.

Як приклад можна розглянути клас A, який залежить від екземпляра класу B. Традиційно екземпляр класу В можна отримати шляхом операції new або якогось фабричного класу. Згідно з принципом інверсії управління екземпляр класу В надається класу А під час виконання програми деяким зовнішнім процесом.

Основою Spring фреймворка є контейнер інверсії управління, який налаштовує та керує Java об’єктами за допомогою рефлексії. Крім того, контейнер регулює життєвий цикл об’єктів: створює ці об’єкти, викликає методи для ініціалізації та конфігурує їх шляхом зв’язування між собою.

Об’єкти, створені контейнером, ще називаються керованими об’єктами або бінами. Контейнер може бути налаштований використовуючи XML-файли, анотації або конфігураційні класи.

Об’єкти можуть бути отримані шляхом пошуку чи впровадження залежностей. Пошук залежностей – це шаблон програмування, коли викликаючий об’єкт посилає контейнеру об’єктів запит для знаходження об’єкта, базуючись на певному імені або типі. Впровадження залежностей – це інший патерн, в якому контейнер передає екземпляри об’єктів по їх імені іншим об’єктам за допомогою конструктора або фабричного метода.

Розробники фреймворку стверджують, що Spring контейнер дозволяє створювати потужні програмні рішення так як програміст не прямо створює об’єкти, а описує як вони мають бути створені шляхом їх визначення в конфігураційних файлах. Подібно до цього, сервіси та компоненти також не викликаються безпосередньо; замість цього файли налаштувань визначають які сервіси та компоненти будуть викликані. Тобто контейнер інверсії управління спрямований на полегшення підтримки та тестування проекту.

#### Переваги впровадження залежностей в порівнянні з традиційним підходом

* До числа найголовніших переваг можна віднести можливість значного скорочення об’єму коду, який мав би бути написаний для зв’язування компонент програми.
* Впровадження залежностей також може значною мірою спростити конфігурацію програм, оскільки для налаштування класів, що впроваджуються в інші класи, можна обирати різні варіанти. Крім того, також спрощується заміна однієї реалізації залежності на іншу.
* При традиційному підході до управління залежностями спільних служб (наприклад для підключення до джерела даних чи обробки транзакцій) екземпляри залежностей отримуються там, де вони потрібні (зазвичай в залежному класі). Це призводить до розповсюдження залежностей серед багатьох класів в програмі, що може утруднити їх зміну. Але якщо застосувати впровадження залежностей, то вся інформація про спільні залежності зберігається єдиному місці, що значно спрощує управління ними та зменшує ризик помилок.
* Коли класи проектуються для впровадження залежностей стає можливим проста заміна залежностей. Це особливо зручно при тестуванні класів. Наприклад, потрібно протестувати об’єкт предметної області, що виконує складну обробку, використовуючи при цьому клас DAO для доступу до інформації, що зберігається в реляційній базі даних. При тестуванні програміста не цікавить перевірка об’єкта DAO, оскільки треба протестувати об’єкт предметної області на різних наборах даних. При традиційному підході тестування доволі ускладнюється, оскільки неможливо замінити реалізацію об’єкта DAO, реалізацією, яка б імітувала тестові дані. Але використовуючи впровадження залежностей можна імітувати реалізацію об’єкта DAO, який би повертав тестові дані.
* В цілому, проектування для впровадження залежностей означає проектування на рівні інтерфейсів. Типова програма, яка орієнтована на впровадження залежностей побудована таким чином, що спочатку всі його основні компоненти були визначені як інтерфейси а вже потім конкретні реалізації мають бути створені та зв’язані разом за допомогою контейнера впровадження залежностей. Застосувати таке проектне рішення в мові Java можна було й до виникнення фреймворку Spring, але з його застосуванням програміст отримує цілий ряд можливостей для впровадження залежностей та може сконцентруватися на побудові логіки програми, а не на самому каркасі, що підтримує впровадження залежностей.

Отже, як можна побачити технологія впровадження залежностей надає немало переваг, однак є й певні недоліки. В основному, вони пов’язані з тим, що у програмістів, що недостатньо добре орієнтуються в коді, можуть виникнути проблеми розуміння яка саме реалізація прив’язана до конкретного об’єкту. Але, як правило, така проблема виникає лише через брак досвіду з подібними технологіями. Крім того, перелічені вище плюси значно перебільшують цей невеликий недолік, хоч про нього не треба забувати при проектування програмної архітектури.

#### Різниця між Spring та Spring Boot

Як було сказано, фреймворк Spring Boot з’явився для того, щоб полегшити створення веб-проектів на Spring, уникаючи написання шаблонних налаштувань. Нижче я наведу найголовніші відмінності між Spring та Spring Boot.

|  |  |
| --- | --- |
| Spring | Spring Boot |
| Spring широко використовуваний фреймворк Java EE для створення додатків | Фреймворк Spring Boot використовується для написання REST APIs |
| Націлений на те, щоб спростити розробку на Java EE | Головною метою є зменшення об’єму коду та спрощення розробки Web застосунків |
| Основною особливістю є впровадження залежностей | Особливістю є наявність автоконфігурації; фреймворк автоматично налаштовує необхідні класи |
| Для тестування проекту необхідно безпосередньо встановлювати сервер | Містить вбудований сервер, наприклад Jetty та Tomcat |
| Не підтримує бази даних, що зберігаються в пам’яті | Пропонує декілька плагінів для роботи з базами даних, що зберігаються в пам’яті, таких як H2 |
| Програмісти вручну визначають залежності в файлі pom.xml | Були створені спеціальні starters, які відповідають за скачування необхідних jar-залежностей. |

### Патерн Model-View-Controller (MVC)

#### Основні поняття

Model-View-Controller (Модель-Вид-Контролер) – схема розділення даних програми, користувацького інтерфейсу та управляючої логіки на три окремих компоненти: модель, представлення та контролер – таким чином, що модифікація кожної з компонент може відбуватися незалежно.

* Модель – представляє дані та методи роботи з ними: запити до бази даних, перевірка на коректність. Модель не залежить від представлення (не знає як візуалізувати дані) і контролера (не має точок взаємозв’язку з користувачем), а просто надає доступ до даних та керуванню ними.
* Представлення відповідає за відображення даних моделі користувачу, при цьому реагуючи на зміни моделі, але не обробляє введені користувачем дані.
* Контролер забезпечує зв’язок між користувачем та системою. Контролює та направляє дані від користувача до системи і навпаки. Використовує модель та представлення для реалізації необхідної дії.

Основна мета застосування цього шаблону полягає у відокремленні бізнес-логіки (моделі) від її візуалізації (представлення). За рахунок такого розділення збільшується можливість повторного використання коду. Найбільш корисно застосування даної концепції в тих випадках, коли користувач має бачити ті самі дані одночасно в різних контекстах та/або з різних точок зору. Також виконуються такі задачі:

* До однієї моделі можна підключити декілька видів, при цьому не чіпаючи реалізацію моделі. Наприклад одні й ті самі дані можуть бути представлені у вигляді таблиці, гістограми та кругової діаграми.
* Не змінюючи реалізацію видів можна змінити реакцію на дії користувача – для цього достатньо використати інший контролер.
* Ряд розробників спеціалізується тільки в одній з областей: або розробляють графічний інтерфейс або бізнес-логіку. Тому є можливість досягти того, що програмісти, які займаються розробкою бізнес-логіки, взагалі не будуть знати проте, яке представлення буде використовуватися.

#### Використання у веб-додатках

Не дивлячись на те, що на початку свого існування патерн використовувався для написання десктопних програм, MVC став широко використовуваний у проектуванні веб-додатків у багатьох мовах програмування. Крім того, декілька веб-фреймворків були створені на основі цього шаблона. Ці програмні фреймворки різняться у своїх інтерпретаціях, в основному тим, як функції MVC розподілені між сервером та клієнтом.

Частина веб MVC фреймворків використовують підхід тонкого клієнта, коли майже вся логіка моделі, представлення та контролера розміщується на сервері. Це відображено у таких фреймворках як Django, Rails, Spring та ASP.NET MVC. При цьому підході клієнт надсилає запит контролеру а потім отримує веб-сторінку або інший документ (наприклад JSON). В іншому випадку такі фреймворки як Angular, JavaScriptMVC та Backbone дозволяють компонентам MVC виконуватися частково на стороні клієнта.

#### Архітектура Spring MVC

Фреймворк Spring MVC забезпечує архітектуру патерна MVC за допомогою слабо зв’язаних готових компонентів. Вся логіка роботи Spring MVC побудована навколо DispatcherServlet – реалізації шаблона Front Controller, який приймає та обробляє усі HTTP-запити та відповіді на них.

Архітектура Spring Web MVC зображена на малюнку (рисунок 1), а нижче буде наведено короткий опис того, що на ньому зображено.

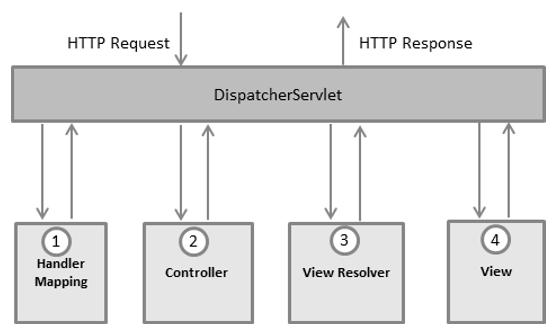


Рисунок 1 - Архітектура Spring Web MVC

1. Взаємодія між компонентами, зображеними на малюнку, починається з моменту, коли DispatcherServlet отримує запит від клієнта.
2. Після отримання HTTP-запиту DispatcherServlet звертається до інтерфейса HandlerMapping, що визначає який Контролер має викликатися, після чого відправляє запит до потрібного Контролера.
3. Контролер приймає запит та викликає відповідний службовий метод, що оснований на GET чи POST. Метод, що був викликаний, визначає дані, що основані на певній бізнес-логікі та повертає DispatcherServlet ім’я Виду.
4. За допомогою інтерфейсу ViewResolver DispatcherServlet визначає який Вид потрібно використати на основі отриманого імені.
5. Після того, як Вид був створений, DispatcherServlet відправляє дані Моделі у вигляді атрибутів до Виду, який врешті-решт відображується у браузері.

Всі вищезгадані компоненти, а саме HandlerMapping, Controller та ViewResolver є частинами інтерфейсу WebApplicationContext, який наслідується від інтерфейсу ApplicationContext, з деякими додатковими особливостями, що необхідні для створення веб-додатків.

### REST веб-сервіси

#### Основні принципи

Веб-сервіс (веб-служба) – це система, що доступна в інтернет-просторі і працює на основі спеціальної програми, ідентифікація якої здійснюється за допомогою URL-строки. Веб-служби можуть взаємодіяти одна з одною та зі сторонніми програмами за допомогою повідомлень, що основуються на певних протоколах (SOAP, XML, REST тощо).

Simple Object Access Protocol (SOAP) – вважається основним підходом при написанні веб-сервісів. SOAP – це протокол, який дозволяє відправляти та отримувати повідомлення, використовуючи в якості основи мову розмітки XML. Однак в останні роки Representational State Transfer (REST) став достатньо популярною альтернативою традиційним SOAP веб-сервісам. Термін REST був вперше введений у 2000 році Роєм Філдінгом (Roy Fielding), одним з авторів протоколу HTTP. Він також описав шість основних принципів, які стосуються архітектури RESTful веб-сервісів.

* Уніфікований інтерфейс: Основою REST є ресурси, які ідентифікуються за допомогою Uniform Resource Identifiers (URI). Концептуально ресурси відокремлюються від їх представлення (тобто від формату, в якому вони передаються клієнту). REST не передбачає якогось конкретного формату але зазвичай це XML та JSON. Іншою особливістю REST сервісів є те, що взаємодія з клієнтами відбувається використовуючи гіпермедії (hypermedia), які надаються серверною частиною додатку. Це означає, що
* Клієнт-сервер: Клієнт-серверна модель дає можливість розділити обов’язки клієнта (взаємодія з користувачем) від обов’язків сервера (збереження даних, масштабуємість). Таке розмежування дозволяє розробляти клієнтську та серверну частини абсолютно незалежно одне від одного, що допомагає зменшити складність та покращити продуктивність.
* Відсутність стану: Жодний стан клієнта не зберігається на сервері, вся інформація, яка необхідна для виконання операцій зберігається у запитах (як частина URL, тіло запиту (request body) або HTTP заголовок)
* Кешування: REST-сервіси мають надавати можливість кешування. Сервери можуть вказувати як та скільки зберігати відповіді. Клієнти можуть використовувати кешованих дані замість того, щоб звертатися до сервера.
* Система наявності слоїв: Зазвичай клієнту неможливо точно визначити, чи він взаємодіє напряму з сервером чи з проміжним вузлом, у зв’язку з ієрархічною структурою мережі. Тоді застосування проміжних серверів надає можливість підвищити масштабуємість за рахунок балансування навантажень і розподіленого кешування.
* Код за вимогою: Цей принцип є опціональним. Під ним мається на увазі, що REST може дозволити розширити функціональність клієнта за рахунок завантаження коду з сервера у вигляді аплетів або сценарієв.

#### HTTP-методи

HTTP (HyperText Transfer Protocol – протокол передачі гіпертексту) – протокол прикладного рівня передачі даних спочатку – у вигляді гіпертекстових документів у форматі HTML, зараз використовується для передачі довільних даних.

Стандарт HTTP пропонує вісім HTTP-методів, які дозволяють клієнтам маніпулювати та взаємодіяти з ресурсами. Найбільш поширеними методами є GET, POST, PUT та DELETE.

HTTP-метод вважається безпечним, якщо він не викликає ніяких змін на стороні сервера. Такі методи (наприклад HEAD або GET) використовуються лише для отримання інформації або ресурсів від сервера. Тим не менш, це не є гарантією того, що кожен раз будуть повертатися одні й ті самі дані.

Операція вважається ідемпотентною, якщо вона призводить до одного й того стану сервера, незалежно від того скільки разів ця операція застосовувалась. HTTP-методи GET, HEAD, PUT та DELETE вважаються ідемпотентними, гарантуючи те, що клієнти можуть повторювати запит та очікувати одного й того ж самого ефекту, щ був досягнутий при першому запиті. Наприклад, якщо користувач видаляє ресурс, то у разі успішного виконання запиту, ресурс більше не існує на сервері, звідси випливає, що всі наступні спроби видалити цей самий ресурс призведуть до того самого стану сервера.

На відміну від цього, POST метод не є ні безпечним ні ідемпотентним. У разі успішного виконання запиту сервер створить новий ресурс стільки разів скільки POSTзапитів прийде на нього.

## Робота з базами даних та бібліотека Hibernate

#### Основні поняття

Майже кожна програма потребує можливість збереження даних в якомусь інформаційному сховищі. Найбільш широковживаним та зручним таким сховищем є реляційна база даних.

JDBC (Java DataBase Connectivity) - це технологія, яка забезпечує доступ Java API до реляційних баз даних, тому Java-програми можуть виконувати SQL-запити та взаємодіяти з БД, які підтримують SQL.

JDBC є достатньо гнучкою технологією та дозволяє створювати програми, які не залежать від конкретної платформи та можуть взіємодіяти з різними СУБД без яких-небудь змін у програмному коді. Серед плюсів JDBC є проста та зрозуміла обробка SQL-запитів та зрозумілий синтаксис. Тим не менш існує ряд суттєвих мінусів таких як складність використання у великих проектах, велика кількість коду та складна реалізація концепції MVC.

При створенні програми на Java та її інтеграції з БД необхідно розуміти, що існує суттєва різниця між об’єктною та реляційною моделями. СУБД дає інформацію у вигляді таблиць, в той час як Java – у вигляді деякого графу об’єктів.

ORM (Object-Relational Mapping – об’єктно реляційний зв’язок) – технологія програмування, яка створена для того, щоб забезпечити перетворення даних при їх обміні між реляційною БД та Java. Серед переваг ORM у порівнянні з JDBC можна виділити наступне:

* Пришвидшення розробки програми
* Відокремлення SQL запитів від об’єктно-орієнтованої моделі
* Управління транзакціями
* Підтримка багатопоточності

Серед усіх ORM бібліотек, які доступні з відкритим кодом, Hibernate вважається найбільш вдалою зважаючи на її функціональні можливості, що включають підхід, який заснований на POJO об’єктах, простоту розробки та підтримку визначень складних відношень.

Співставлення Java класів з таблицями БД здійснюється за допомогою конфігураційних XML-файлів або Java анотацій і допомагає встановити зв’язок між двома таблицями. Зв’язок може бути одним з наступних варіантів: один-до-одного, багато-до-одного/один-до-багатьох та багато-до-багатьох.

#### Рівень доступу до даних

У багаторівневій архітектурі розробки програмного забезпечення рівень, який надає доступ до даних, що зберігаються у сховищі називається рівнем доступу до даних (persistence layer). У архітектурі Spring фреймворк цей рівень поділяється на три частини: модель, рівень доступу до об’єктів (Data Access Object – DAO) та службовий рівень.

* Модель представляє ключові сутності у програмі, визначаючи яким чином вони співвідносяться одне з одним. Кожна сутність визначає кілька властивостей, якими описуються їх властивості та зв’язок з іншими сутностями. Зазвичай, кожна Модель співставляється з таблицею у БД, але це не обов’язково.
* Рівень DAO визначає методи для збереження та отримання Моделей. Ціль шаблона DAO полягає у абстрагуванні технології збереження та того, як саме дані зберігаються чи завантажуються від самої Моделі. Ключова перевага патерна DAO це відділення низькорівневих деталей від іншої частини програми за допомогою набору методів. Якщо зміниться технологія доступу до даних то всі необхідні зміни будуть полягати у створенні нової реалізації DAO, який імплементує той самий інтерфейс.

# Практична частина

## Дослідження алгоритмів

Моєю першочерговою задачею було дослідження та структурування алгоритмів з курсу sОбчислювальна Геометрія. На початку я отримала набір звітів лабораторних робіт студентів попередніх років. Необхідно було їх розібрати, згрупувати відповідно до теми, виділити найголовніші моменти та врешті-решт створити певний підсумок.

Одним з різновидів алгоритмів був пошук найкоротших шляхів на мережі з перешкодами. Задача звучить так: задана множина перешкод у вигляді многокутників та дві запитні точки Початок і Кінець; необхідно побудувати найкоротший шлях в обхід перешкод без використання трінгуляції. Розв’язки цієї задачі мають велике практичне застосування і може використовуватися у системах GPS, робототехніці, для побудови траєкторії руху робота, людини або транспортного засобу. Також ця задача може знайти призначення у питаннях прокладання маршруту для перевезень, військових місій, комп’ютерних ігор, прокладання кабелів та іншої інфраструктури.

Наступним типом алгоритмів була мінімально зважена тріангуляція. Це є однією з базових задач в обчислювальній геометрії, до якої зводиться багато інших задач. Вона широко використовується в машинній графіці та геоінформаційних системах для моделювання поверхонь та вирішень просторових задач. Постановка задачі є наступною: необхідно побудувати тріангуляцію, сумарна довжина ребер якої є найменшою. Однак побудова мінімально зваженої тріангуляції є NP-складною задачею, тому для більшості реальних задач існуючі алгоритми побудови оптимальної тріангуляції неприйнятні через надмірно високу складність.

Нарешті третім різновидом алгоритмів була побудова опуклих оболонок у тривимірному просторі. Це є одним з найпоширених типів задач, розв’язки якої застосовуються в алгоритмах маршрутизації, кристалографії та робототехніці. Існує декілька підходів до знаходження опуклих оболонок, серед них можна виділити алгоритм Джарвіса, стратегію «розділяй та володарюй» та швидкий метод (Quick Hull).

## Проектування та створення бази даних.

Наступним кроком було проектування та створення бази даних. Після обговорення та аналізу схеми побудови клієнтської та серверної частини додатку, було вирішено створити наступну модель БД (Рисунок 2).

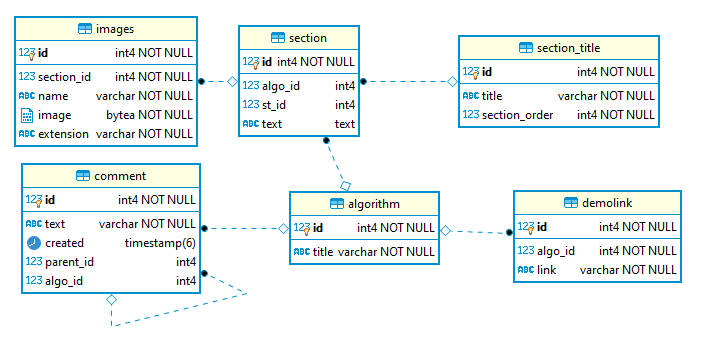


Рисунок 2 - ER діаграма

* Algorithm містить інформацію про назву алгоритму (title) та первинний ключ (id).
* Demolink – таблиця, у якій зберігаються дані про посилання на демонстраційні відео (link), має первинний ключ id та зовнішній ключ algo\_id, який вказує на id у таблиці algorithm. Demolink має зв’язок багато-до-одного з таблицею algorithm.
* Comment зберігає дані про коментарі, а саме текст коментаря (text), час створення (created), первинний ключ (id), зовнішній ключ parent\_id на самого себе (сам-на-себе), який вказує на батьківський коментар (-1 якщо немає батька) та зовнішній ключ algo\_id на таблицю algorithm (багато-до-одного).
* Section\_title – таблиця, у якій зберігається інформація про назву розділу алгоритму. Має такі поля: title – назва розділу; order – порядок, у якому розділи слідують один за одним, та id – первинний ключ.
* Section містить дані про розділ, які включають в себе текст (text) [[1]](#footnote-1), який містить розділ, первинний ключ id, зовнішній ключ algo\_id, який вказує до якого алгоритму належить цей розділ (багато-до-одного), та зовнішній ключ st\_id, який вказує на таблицю section\_title, тобто на назву цього розділу (багато-до-одного).
* Images[[2]](#footnote-2) містить інформацію про малюнок, має такі поля як image (набір байтів, що утворюють малюнок), extension – розширення малюнку (jpg, png тощо), name – назва малюнку, первинний ключ id та зовнішній ключ section\_id, який вказує на те, до якого розділу відноситься цей малюнок.

Всі первинні ключі мають відповідні sequence для автогенерації.

Для розгортання БД була використана бібліотека для відслідковування, управління та зміни схеми БД Liquibase.[17] Ця бібліотека була створена у 2006 році для спрощення відслідковування змін у БД, особливо у гнучкому середовищі розробки ПЗ.

Усі зміни в БД зберігаються у текстових файлах (XML) та ідентифікуються комбінацією тегів “id” та “author”, а також іменем самого файлу (додаток А, Рисунок 5). Додатково Liquibase створює дві допоміжні таблиці databasechangelog та databasechangeloglock для відслідковування змін, що були внесені (додаток А, Рисунок 6).

Крім того, БД знаходиться у трьох нормальних формах [16], так як:

* Всі атрибути, що зберігаються на перетині стовбця та строки є атомарними, а таблиці не містять однакових строк. (1НФ)
* Кожен стовпчик, що не є первинним ключем, залежить від первинного ключа. (2НФ)
* Кожен стовпчик, що не є первинним ключем, залежить тільки від первинного ключа. (3НФ)

## Розробка програми

Для розробки серверної частини програми використовувався фреймворк Spring Boot 2. Під час проектування за основу було взято патерн MVC.

В якості моделей було створено шість класів, які представлені на діаграмі класів.

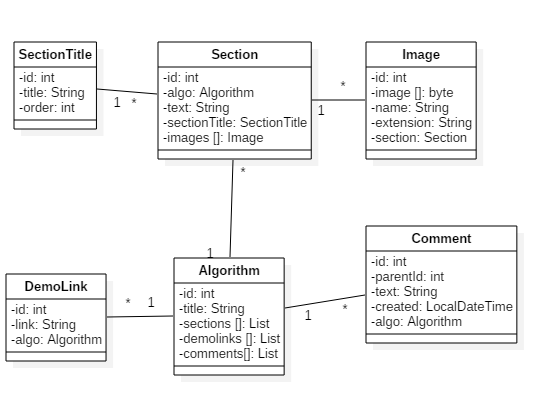


Рисунок 3 - діаграма класів

Для зменшення шаблонного коду була використана бібліотека Lombok, яка підключається у проект за допомогою Spring Starters.

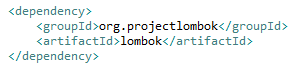


Рисунок 4 - Spring starter Lombok

До її складу входять багато корисних анотацій. Наприклад, анотація @Data, яка включає в себе одразу декілька анотацій (@ToString, @EqualsAndHashCode, @Getter, @Setter, @RequiredArgsConstructor) та автоматично генерує гетери, сетери, імплементує методи toString(), equals() та hashCode() та створює конструктор без параметрів (Додаток А, Рисунок 7).

В якості ORM бібліотеки була обрана бібліотека Hibernate. Для створення зв’язків між полями використовувалися анотації @ManyToOne та @OneToMany (Додаток А, Рисунок 8).

## Розгортання програми [19]

В якості платформи для розгортання програми була обрана хмарна платформа Heroku (<https://heroku.com>). Одна з найголовніших переваг цього сервісу полягає в тому, що він дає можливість використовувати безкоштовні віртуальні машини з обмеженням на час роботи, при чому для цього достатньо лише зареєструватися. Крім того, є декілька способів розгорнути додаток: використовуючи heroku git репозиторій, прив’язка до репозиторію на github (<https://github.com> ), використання контейнеру docker тощо.

Для розгортання застосунку був обраний перший варіант. Для цього спочатку потрібно було створити git-репозиторій та додати туди код, скориставшись наступними командами:

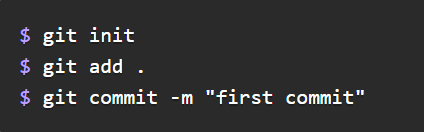


Рисунок 5 - команди для ініціалізації репозиторію

Після цього необхідно створити нову програму на Heroku, запустивши команду heroku create. Після цього можна розгортати код командою git push heroku master. Платформа автоматично розпізнає проект як Maven/Java через наявність файлу pom.xml.

# Перелік джерел посилання

1. History of Spring Framework and Spring Boot [Електронний ресурс] – Режим доступу до ресурсу: <https://www.quickprogrammingtips.com/spring-boot/history-of-spring-framework-and-spring-boot.html>.
2. Spring 5 для профессионалов / Ю.Козмина, Р. Харроп, К. Шефер, К. Хо. – Санкт-Петербург: Диалектика, 2019. – 1120 с. – (5).
3. Spring Modules [Електронний ресурс] – Режим доступу до ресурсу: <https://www.javatpoint.com/spring-modules>.
4. Spring vs. Spring Boot vs. Spring MVC [Електронний ресурс] – Режим доступу до ресурсу: <https://www.javatpoint.com/spring-vs-spring-boot-vs-spring-mvc>.
5. Spring Framework [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Spring\_Framework.
6. Spring Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Spring_Framework>.
7. Model-View-Controller [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>.
8. Model-View-Controller [Електронний ресурс] – Режим доступу до ресурсу: <https://ru.wikipedia.org/wiki/Model-View-Controller>.
9. Tejaswini Mandar Jog. Learning Spring 5.0 / Tejaswini Mandar Jog. – Birmingham: Packt Publishing Ltd, 2017. – 422 с.
10. Чернишов В. Spring MVC - Основные принципы [Електронний ресурс] / В'ячеслав Чернишов. – 2017. – Режим доступу до ресурсу: <https://habr.com/ru/post/336816/>.
11. Dewailly L. Building a RESTful Web Service with Spring / Ludovic Dewailly. – Birmingham: Packt Publishing Ltd., 2015. – 123 с.
12. Varanasi B. Spring REST / B. Varanasi, S. Belida. – New York: Apress, 2015. – 221 с.
13. Hibernate (framework) [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Hibernate_(framework)>.
14. Руководство по Hibernate [Електронний ресурс] – Режим доступу до ресурсу: <https://proselyte.net/tutorials/hibernate-tutorial/introduction/>.
15. Fisher P. Spring Persistence with Hibernate / P. Fisher, B. D. Murphy. – New York: Apress, 2016. – 178 с. – (2).
16. Сивков С. Нормализация отношений. Шесть нормальных форм [Електронний ресурс] / Сергій Сивков. – 2015. – Режим доступу до ресурсу: https://habr.com/ru/post/254773/.
17. Liquibase [Електронний ресурс] – Режим доступу до ресурсу: https://ru.bmstu.wiki/Liquibase.
18. Walls C. Spring in Action / Craig Walls. – New York: Manning, 2019. – 498 с. – (5).
19. Deploying Spring Boot Applications to Heroku [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: <https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku>.

Висновки

<https://topjava.ru/blog/spring-framework-vs-spring-boot-differences>

# Додаток А



Рисунок 6 - Створення таблиці demolink

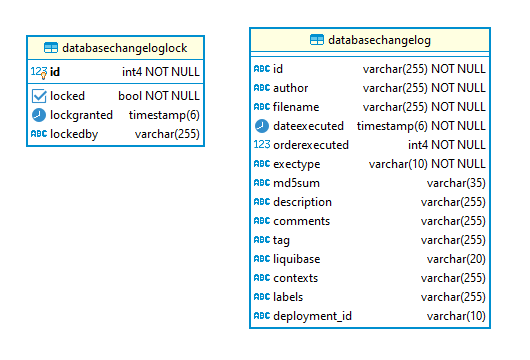


Рисунок 7 -таблиці databasechangeloglock та databasechangelog



Рисунок 8 - Використання Lombok анотацій

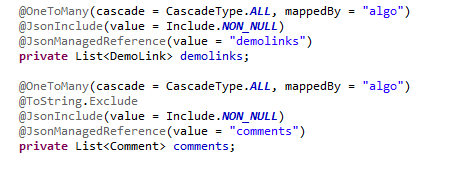


Рисунок 9 - Використаня Hibernate анотацій

1. Насправді зберігається не сам текст, а HTML-розмітка, що відповідає даному розділу, разом із зображеннями. [↑](#footnote-ref-1)
2. Після обговорень з розробником клієнтської частини було вирішено не використовувати цю таблицю так як зображення зберігаються прямо в HTML-розмітці. [↑](#footnote-ref-2)