Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

# ЛАБОРАТОРНА РОБОТА 4

З дисципліни “Теорія систем та системний аналіз”

Тема: Побудова діаграми класів та діаграми пакетів

**Виконали:**
студент групи ІА-11:
Воробей А. О
студенти групи ІА-13:
Середа А. А.
Павлюк О. І
студент групи ІА-14:
Фіалківський І. О.

**Перевірив:**
Барбарук В. М.

Київ 2024

**Тема:** Побудова діаграми класів та діаграми пакетів

**Мета:** Ознайомлення з методологією та інструментальними засобами моделювання класів моделі системи на основі мови UML.

## Хід роботи:

Діаграма пакетів є однією з ключових діаграм для візуалізації архітектурної структури програмного забезпечення, оскільки вона дозволяє зображати вищий рівень структуризації проєкту. Ця діаграма показує, як програмний проєкт поділений на пакети, організовуючи систему у логічні сегменти, що забезпечує кращий огляд залежностей та взаємозв'язків між різними частинами системи. Пакети можуть включати класи, інтерфейси та інші пакети, створюючи ієрархічну структуру, яка допомагає управлінню складністю проєкту. Кожен пакет слугує контейнером для групування пов'язаних елементів, що сприяє модульності та зниженню зв'язності між компонентами системи.

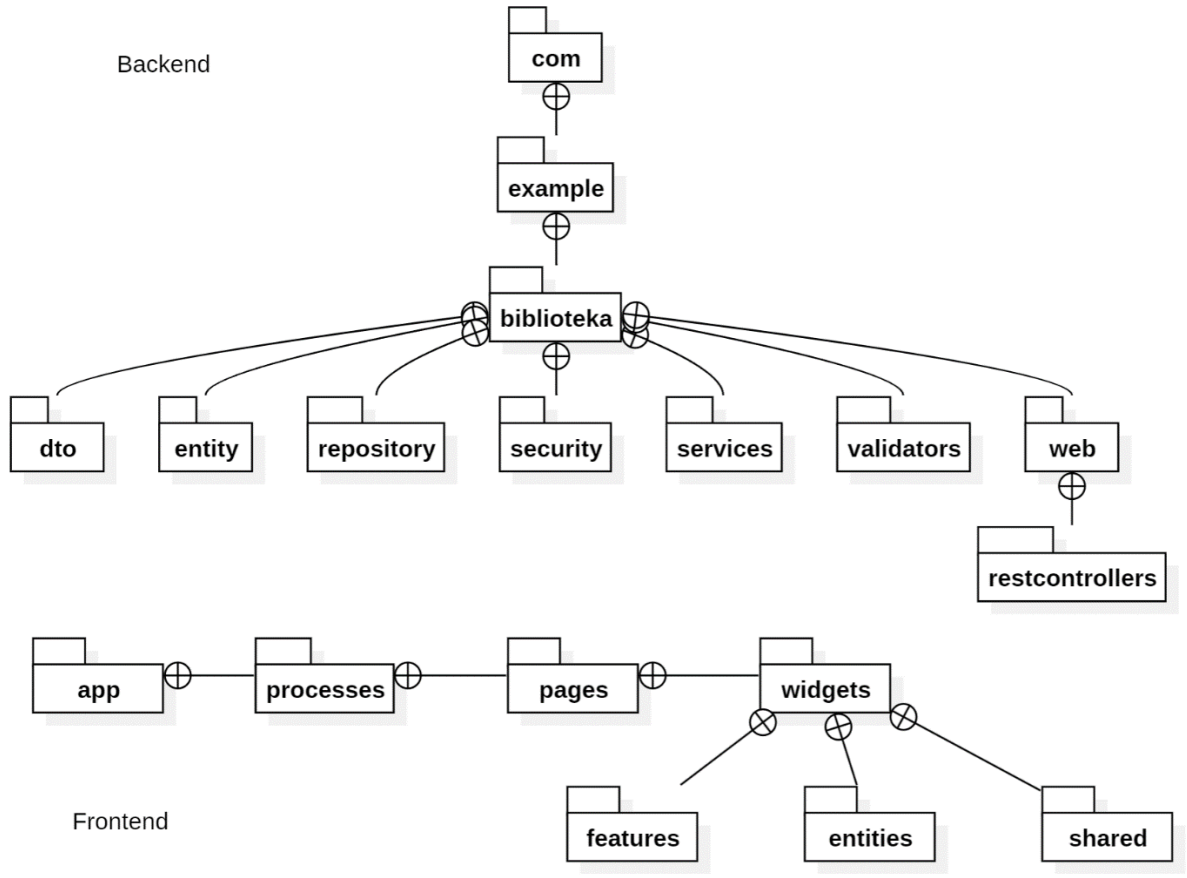
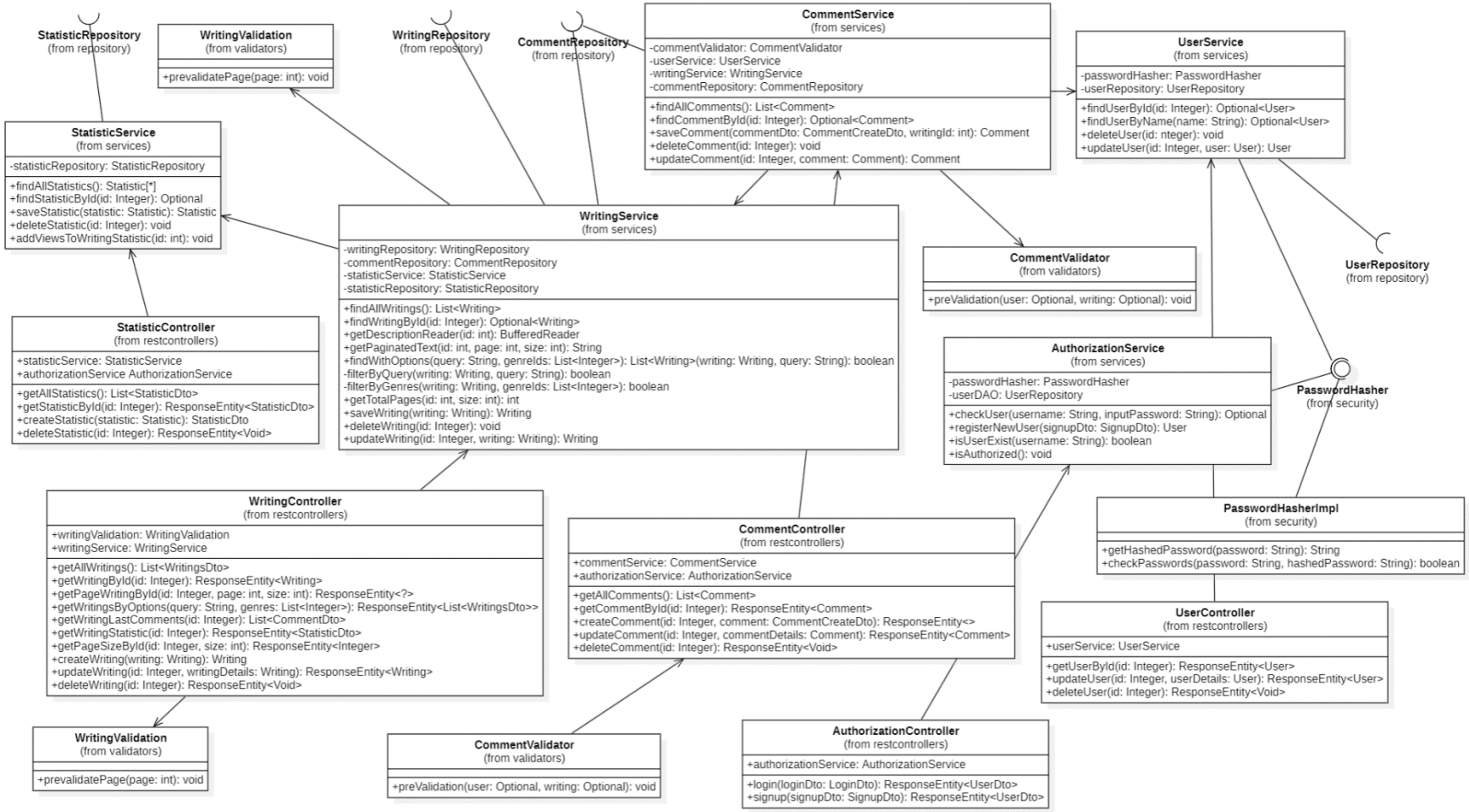



Рис. 1- Діаграма пакетів

На нашій діаграмі пакетів ми відобразили як Frontend, так і Backend частини проекту. Для Frontend частини це в основному відображення структуру звичайного Frontend application, з різними сторінками та їх взаємодій, в той час як Backend діаграма пакетів включає зображення серверної логіки, і сервісів, які обробляють бізнес-логіку та взаємодію з Frontend.

З іншого боку, діаграма класів є більш деталізованою і використовується для представлення структури даних і взаємодії в межах системи на більш низькому рівні. Вона показує класи системи, їх атрибути, методи та взаємозв’язки між ними.

В діаграмі класів для нашого проекту, зображеної нижче на рисунків 2-3, ми зосередили увагу на BackEnd частині, оскільки саме тут зосереджена основна логіка обробки даних. Ця діаграма детально відображає, як різні класи спілкуються між собою, які об'єкти створюються, та які дані передаються у процесі роботи системи.


StatisticRepository
(from repository)
WritingValidation
(from validators)
+prevalidatePage(page: int): void
WritingRepository
(from repository)
CommentRepository
(from repository)
CommentService
(from services)
-commentValidator: CommentValidator
-userService: UserService
-writingService: WritingService
-commentRepository: CommentRepository
+findAllComments(): List<Comment>
+findCommentById(id: Integer): Optional<Comment>
+saveComment(commentDto: CommentCreateDto, writingId: int): Comment
+deleteComment(id: Integer): void
+updateComment(id: Integer, comment: Comment): Comment
UserService
(from services)
-passwordHasher: PasswordHasher
-userRepository: UserRepository
+findUserById(id: Integer): Optional<User>
+findUserByName(name: String): Optional<User>
+deleteUser(id: nteger): void
+updateUser(id: Integer, user: User): User
StatisticService
(from services)
-statisticRepository: StatisticRepository
+findAllStatistics(): Statistic[*]
+findStatisticById(id: Integer): Optional
+saveStatistic(statistic: Statistic): Statistic
+deleteStatistic(id: Integer): void
+addViewsToWritingStatistic(id: int): void
WritingService
(from services)
-writingRepository: WritingRepository
-commentRepository: CommentRepository
-statisticService: StatisticService
-statisticRepository: StatisticRepository
+findAllWritings(): List<Writing>
+findWritingById(id: Integer): Optional<Writing>
+getDescriptionReader(id: int): BufferedReader
+getPaginatedText(id: int, page: int, size: int): String
+findWithOptions(query: String, genreIds: List<Integer>): List<Writing>(writing: Writing, query: String): boolean
-filterByQuery(writing: Writing, query: String): boolean
-filterByGenres(writing: Writing, genreIds: List<Integer>): boolean
+getTotalPages(id: int, size: int): int
+saveWriting(writing: Writing): Writing
+deleteWriting(id: Integer): void
+updateWriting(id: Integer, writing: Writing): Writing
CommentValidator
(from validators)
+preValidation(user: Optional, writing: Optional): void
UserRepository
(from repository)
StatisticController
(from restcontrollers)
+statisticService: StatisticService
+authorizationService AuthorizationService
+getAllStatistics(): List<StatisticDto>
+getStatisticById(id: Integer): ResponseEntity<StatisticDto>
+createStatistic(statistic: Statistic): StatisticDto
+deleteStatistic(id: Integer): ResponseEntity<Void>
AuthorizationService
(from services)
-passwordHasher: PasswordHasher
-userDAO: UserRepository
+checkUser(username: String, inputPassword: String): Optional
+registerNewUser(signupDto: SignupDto): User
+isUserExist(username: String): boolean
+isAuthorized(): void
PasswordHasher
(from security)
WritingController
(from restcontrollers)
+writingValidation: WritingValidation
+writingService: WritingService
+getAllWritings(): List<WritingsDto>
+getWritingById(id: Integer): ResponseEntity<Writing>
+getPageWritingById(id: Integer, page: int, size: int): ResponseEntity<?>
+getWritingsByOptions(query: String, genres: List<Integer>): ResponseEntity<List<WritingsDto>>
+getWritingLastComments(id: Integer): List<CommentDto>
+getWritingStatistic(id: Integer): ResponseEntity<StatisticDto>
+getPageSizeById(id: Integer, size: int): ResponseEntity<Integer>
+createWriting(writing: Writing): Writing
+updateWriting(id: Integer, writingDetails: Writing): ResponseEntity<Writing>
+deleteWriting(id: Integer): ResponseEntity<Void>
CommentController
(from restcontrollers)
+commentService: CommentService
+authorizationService: AuthorizationService
+getAllComments(): List<Comment>
+getCommentById(id: Integer): ResponseEntity<Comment>
+createComment(id: Integer, comment: CommentCreateDto): ResponseEntity<>
+updateComment(id: Integer, commentDetails: Comment): ResponseEntity<Comment>
+deleteComment(id: Integer): ResponseEntity<Void>
PasswordHasherImpl
(from security)
+getHashedPassword(password: String): String
+checkPasswords(password: String, hashedPassword: String): boolean
UserController
(from restcontrollers)
+userService: UserService
+getUserById(id: Integer): ResponseEntity<User>
+updateUser(id: Integer, userDetails: User): ResponseEntity<User>
+deleteUser(id: Integer): ResponseEntity<Void>
WritingValidation
(from validators)
+prevalidatePage(page: int): void
CommentValidator
(from validators)
+preValidation(user: Optional, writing: Optional): void
AuthorizationController
(from restcontrollers)
+authorizationService: AuthorizationService
+login(loginDto: LoginDto): ResponseEntity<UserDto>
+signup(signupDto: SignupDto): ResponseEntity<UserDto>

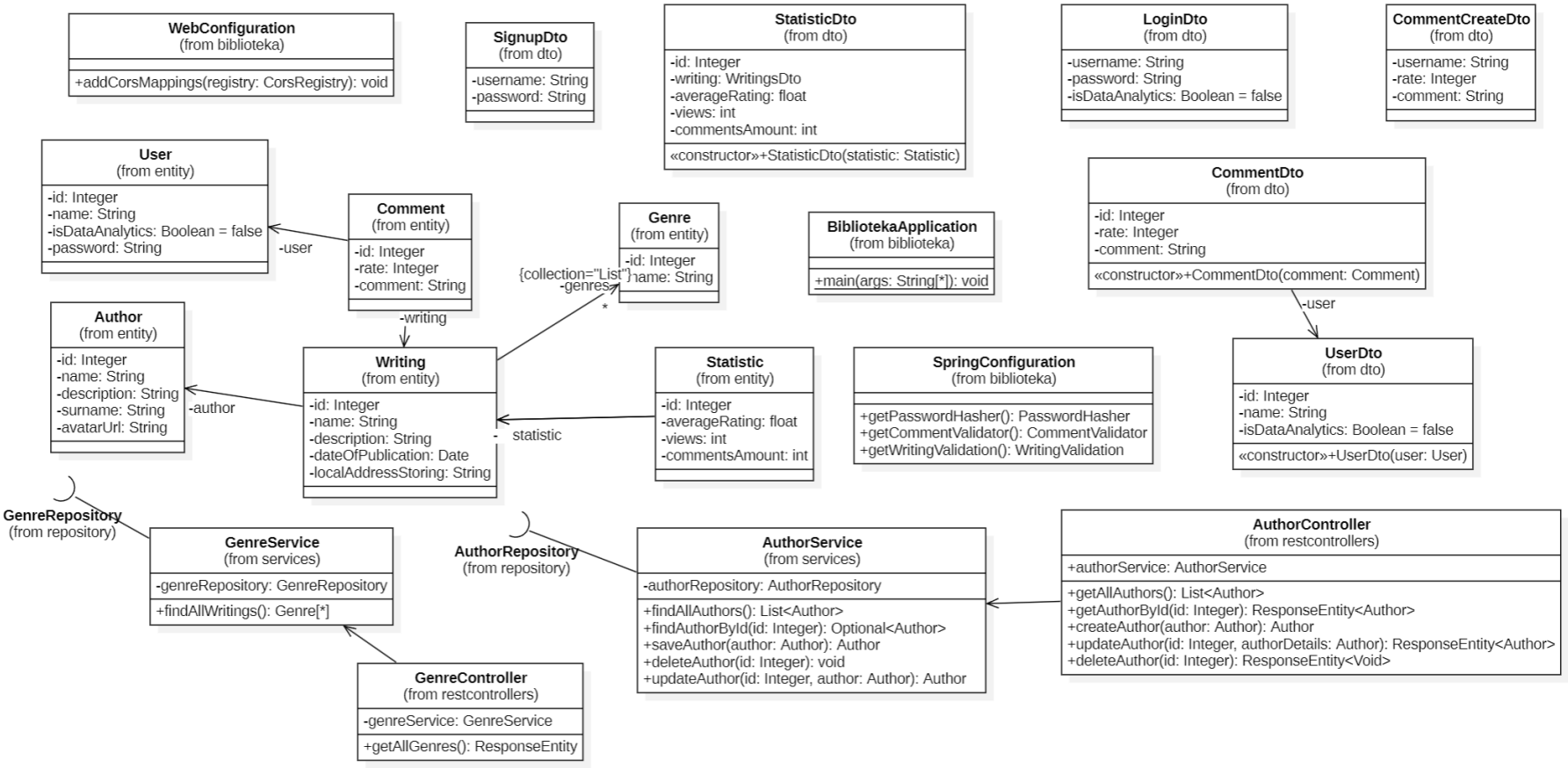



Рис. 2-3 - Діаграми класів

Також, програма StarUML, за допомогою якої ми робимо діаграми класів, дозволяє реалізувати ці класи одразу у вигляді готового проекту. Для цього потрібно скачати плагін «Java», і потім в панелі «Tools -> Java» натиснути «Generate Code», як зображено на рисунку 4.

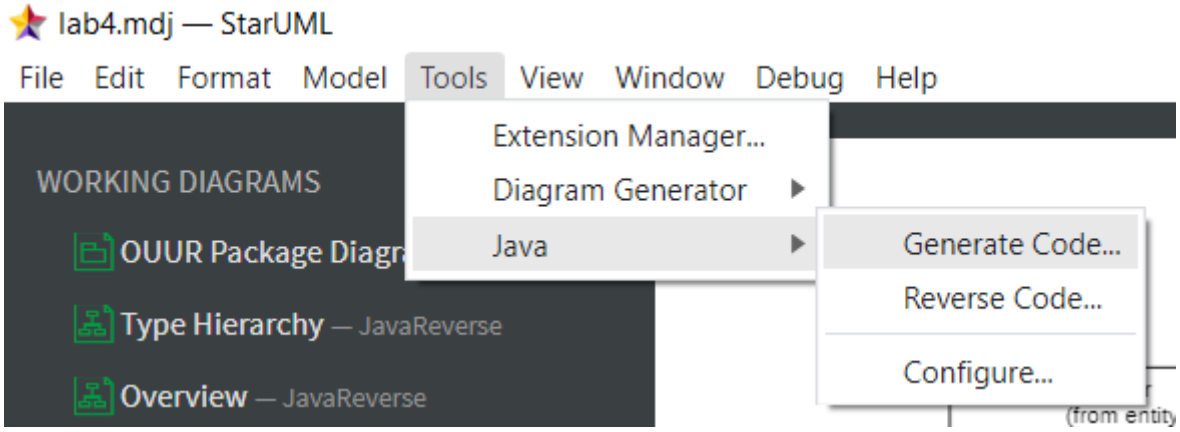



Рис. 4- Міграція класів з діаграми класів

На виході ми отримуємо готовий проект з усіма пакетами і класами, зображений на рисунку 5.

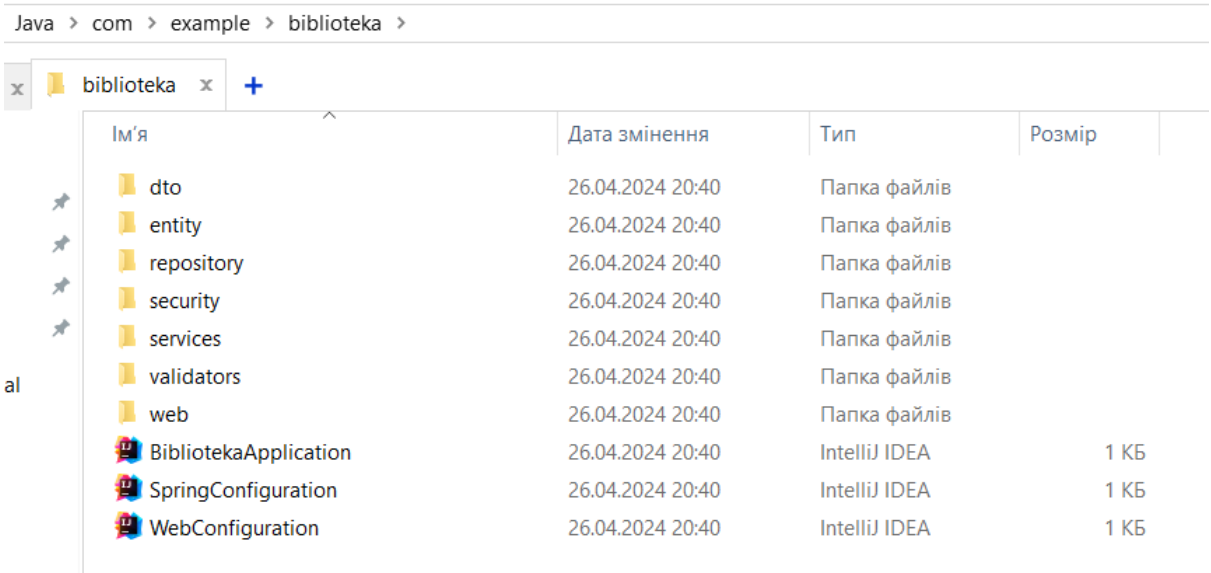



Рис. 5- Готовий проект, згенерований за діаграмою

Перейшовши в один із файлів, ми можемо побачити автозгенеровані класи, які слугують початком для реалізації нашого проекту. Приклад аткого класу можна побачити на рисунку 6.

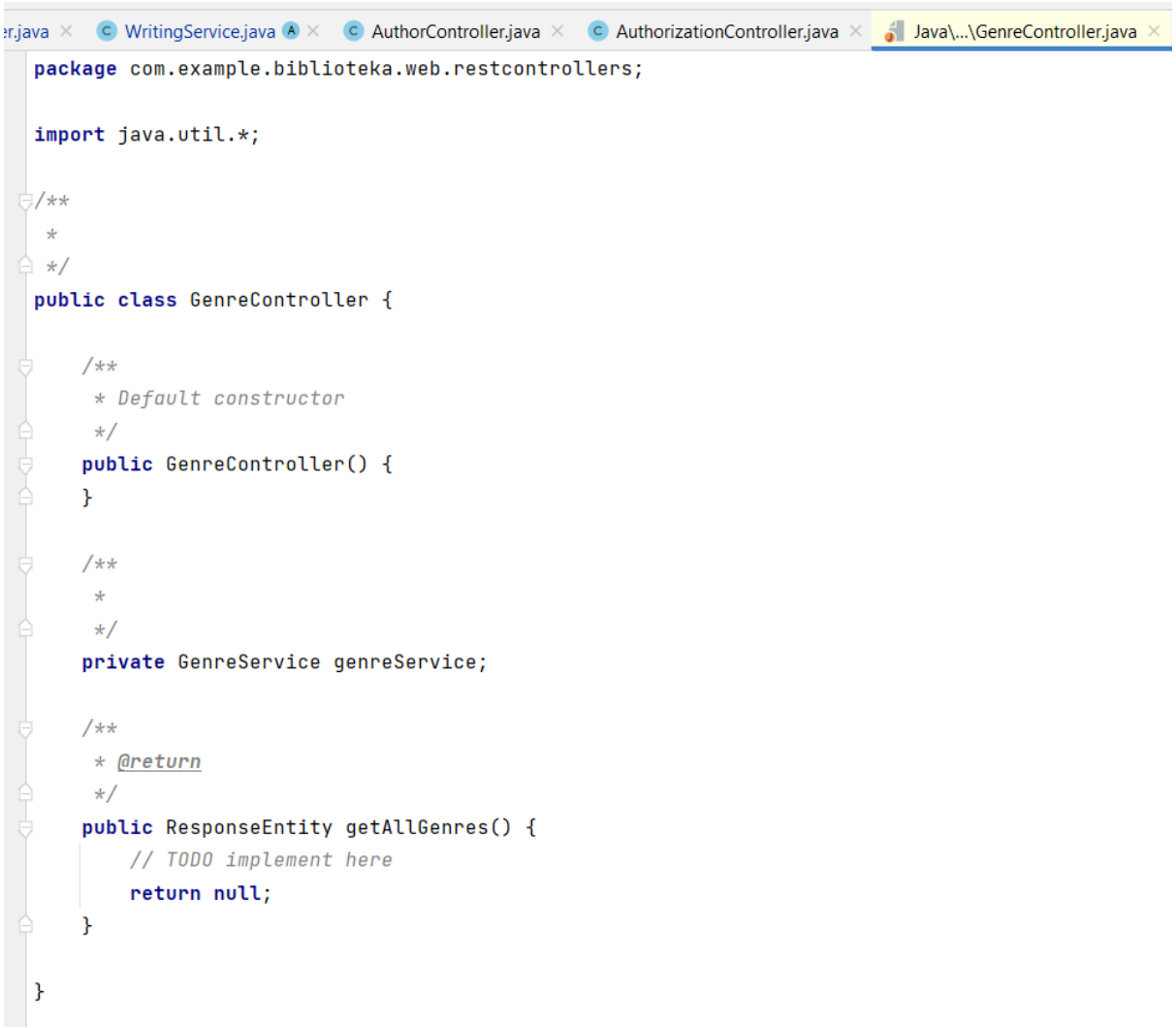



Рис. 6- Автозгенерований клас GenreController

Після реалізації цього класу ми отримуємо вже готовий до використання клас, з усіма залежностями і методами. Приклад вже реалізованого класу зображено на рисунку 7.

```
package com.example.biblioteka.web.restcontrollers;

import ...

@RestController
@RequestMapping("/api/genres")
public class GenreController {
    @Autowired
    private GenreService genreService ;
    @GetMapping
    public ResponseEntity<List<Genre>>  getAllGenres() { return ResponseEntity.ok(genreService.findAllWritings()); }

}
```

Рис. 7- Реалізований клас GenreController

**Посилання**

Код діаграм та звіти розміщені в репозиторії

https://github.com/korovkaK22/LibraryDiargarm

Код backend частини розміщений в репозиторії

https://github.com/korovkaK22/biblioteka

Код frontend частини розміщений в репозиторії

https://github.com/vergovters/library/tree/master

**Висновки:** На цій лабораторній роботі ми навчилися створювати діаграму пакетів та діаграму класів. Також вивчили, які між ними різниці, та як їх правильно створювати. Закріпили набуті знання на практиці, продумавши та реалізувавши дані діаграми для нашого проекту.