Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут комп’ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни «Об’єктно орієнтоване програмування»

Тема: «Дизайн і розробка багатошарового веб-застосунку для гри GachaGirls»

Студентки Шкекіної Ю.А.
групи АІ-221

Спеціальність 122 - «Комп’ютерні науки»

Освітня Програма:
«Комп’ютерні науки»

Керівник:
Годовиченко Микола Анатолійович
Кандидат технічних наук, доцент

Одеса 2024

ЗМІСТ

ВСТУП

Метою курсової роботи є розробка багаторівневого веб-додатку для текстового гача-симулятора з елементом покрокової стратегії. Функціонал гри дозволяє нам отримувати різних персонажів під час гри в рулетку, покращувати їх і битися з різними ворогами за винагороду.

Програма побудована за багаторівневою архітектурою, яка включає наступні кроки:

- Дизайн і розробка веб-шару,
- Проектування та розробка сервісного рівня
- Проектування та розробка шару зберігання
- Проектування та розробка лісозаготівлі

У цій роботі буде детально розглянуто процес проектування та розробки кожного з цих рівнів, а також процес розгортання програми.

Функціональність програми

Керування гравцями

- Створення нового гравця з балансом за замовчуванням і початковим персонажем
- Видалення гравця за Id
- Перевірка даних гравця
- Управління персонажами в колекції гравця, наприклад продаж або оновлення

Управління персонажем

- Отримайте деталі конкретного персонажа
- Видалити персонажа зі своєї колекції
- Отримайте зображення певного персонажа Ascii
- Отримати список усіх можливих персонажів

Управління рулеткою

- Зробіть тягу для символів з різними ймовірностями

Управління боєм

- Отримайте список доступних боїв з численними труднощами
- Бийтеся з обраним ворогом

Дизайн і розробка веб-рівня (запити REST)

вступ

Веб-рівень є ключовим компонентом будь-якої веб-програми, оскільки він відповідає за обробку запитів від клієнтів і повернення відповідей. У нашій програмі ми використовуємо Spring Boot для реалізації веб-служб RESTful. Цей рівень включає контролери, які обробляють HTTP-запити та викликають відповідні служби для виконання бізнес-логіки.

Теоретичний

REST (Representational State Transfer) — це архітектурний стиль для розподілених систем, таких як веб-додатки. Основна ідея REST полягає у використанні стандартних методів HTTP (GET, POST, PUT, DELETE) для виконання операцій над ресурсами, визначеними URL-адресами. Веб-сервіси RESTful прості у використанні, високопродуктивні та масштабовані.

Основними принципами REST є:

- **Стан без громадянства:**Кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для обробки запиту. Сервер не зберігає жодної інформації про клієнта між запитами.
- **Уніфікований інтерфейс:**для взаємодії з ресурсами використовуються стандартні методи HTTP. Це робить систему легкою для розуміння та використання.
- **Архітектура клієнт-сервер:**Клієнт і сервер є незалежними компонентами, що дозволяє розробляти їх окремо один від одного.
- **Кешування:**Відповідь сервера може кешуватися клієнтом, щоб зменшити навантаження на сервер і підвищити продуктивність.
- Багаторівнева система: між клієнтом і сервером можна розмістити додаткові рівні (проксі, шлюзи), що допомагає покращити масштабованість і безпеку системи.

Spring Boot — це популярна платформа веб-додатків на основі Java, яка надає готові до використання компоненти та інструменти для реалізації веб-служб RESTful. Використання Spring Boot дозволяє швидко налаштувати та запустити веб-сервер, реалізувати маршрутизацію запитів та обробку даних.

Реалізація

Player controller

```
@RestController
public class PlayerController {

private final PlayerService playerService;
```

```
private final RouletteService rouletteService;
private final CharacterService characterService;

    public PlayerController(PlayerService playerService,  RouletteService roulet
teService, CharacterService characterService) {
        this.playerService = playerService;
        this.rouletteService = rouletteService;
        this.characterService = characterService;
    }

    @GetMapping(path = "/players/{id}")
    public ResponseEntity<Player> getById(@PathVariable("id") long id) {
        if (id < 0 || !playerService.exists(id)) {
            return ResponseEntity.badRequest().build();
        }
        return ResponseEntity.ok(playerService.getPlayerById(id));
    }

    @DeleteMapping(path = "/players/{id}")
    public ResponseEntity<Player> deleteById(@PathVariable long id) {
        if (id < 0 || !playerService.exists(id)) {
            return ResponseEntity.badRequest().build();
        }
        Player dp = playerService.getPlayerById(id);
        playerService.deletePlayer(id);
        return ResponseEntity.ok(dp);
    }

    @PostMapping("/players")
    public ResponseEntity<Player> createPlayer() {
        Player player = playerService.createPlayer();
        return ResponseEntity.ok(player);
    }

    @GetMapping("/players/{id}/balance")
    public ResponseEntity<Integer> getPlayerBalance(@PathVariable long id) {
        int balance = playerService.getPlayerBalance(id);
        return ResponseEntity.ok(balance);
    }

    @GetMapping("/players/{id}/characters")
    public ResponseEntity<List<OwnedCharacter>> ownedCharacters(@PathVariable lo
ng id){
        if (id < 0 || !playerService.exists(id)) {
            return ResponseEntity.badRequest().build();
        }
        return ResponseEntity.ok(playerService.getPlayerCharacters(id));
    }

    @PatchMapping("/players/{id}/characters/{code}/sell")
    public ResponseEntity<List<OwnedCharacter>> sellOwnedCharacter(@PathVariable
long id, @PathVariable long code){
```

```
        if (id < 0 || code < 0) return ResponseEntity.badRequest().build();

        try {
            return ResponseEntity.ok(playerService.sellGivenCharacter(id, code))
;
        }
        catch (IllegalStateException e) {
            return ResponseEntity.badRequest().build();
        }
        catch (NoSuchElementException e) {
            return ResponseEntity.notFound().build();
        }
    }

    @PatchMapping("/players/{id}/characters/{code}/upgrade")
    public ResponseEntity<OwnedCharacter> upgradeCharacter(@PathVariable long id
, @PathVariable long code){
        if (id < 0 || code < 0) return ResponseEntity.badRequest().build();
        try {
            return ResponseEntity.ok(playerService.upgradeOwnedCharacter(id, cod
e));
        }
        catch (IllegalStateException e) {
            return ResponseEntity.badRequest().build();
        }
        catch (NoSuchElementException e) {
            return ResponseEntity.notFound().build();
        }

    }

}
```

Character controller

```
@RestController
public class CharacterController {

    private final CharacterService characterService;

    private final RouletteService rouletteService;

    private final PlayerService playerService;


    @Autowired
    public CharacterController(CharacterService characterService,
     RouletteService rouletteService, PlayerService playerService) {
        this.characterService = characterService;
        this.rouletteService = rouletteService;
        this.playerService = playerService;
```

```
    }


    @PostMapping("/characters/seed")
    public void seedDatabase(){
        this.characterService.seedDatabase();
    }

    @GetMapping(path = "/characters/{code}")
    public ResponseEntity<Character> getById(@PathVariable long code) {
        if (code < 0 || !characterService.exists(code)) {
            return ResponseEntity.badRequest().build();
        }
        return ResponseEntity.ok(characterService.getCharacterById(code));
    }

    @DeleteMapping(path = "/characters/{code}")
    public ResponseEntity<Character> deleteById(@PathVariable long code) {
        if (code < 0 || !characterService.exists(code)) {
            return ResponseEntity.badRequest().build();
        }
        Character ch = characterService.getCharacterById(code);
        characterService.deleteCharacter(code);
        return ResponseEntity.ok(ch);
    }

    @GetMapping("/characters/{code}/appearance")
    public ResponseEntity<String> getCharacterAppearance(@PathVariable long code
) {
        String appearance = characterService.getCharacterAppearance(code);
        return ResponseEntity.ok(appearance);
    }

    @GetMapping("/characters")
    public ResponseEntity<List<Character>> getPossibleCharacters() {
        List<Character> possibleCharacters = characterService.getPossibleCharact
ers();
        return ResponseEntity.ok(possibleCharacters);
    }
}
```

Roulette controller

```
@RestController
public class RouletteController {

    private final RouletteService rouletteService;

    private final PlayerService playerService;

    private final CharacterService characterService;
```

```
    private final FightService fightService;

    public RouletteController(RouletteService rouletteService, PlayerService pla
yerService,
                              CharacterService characterService, FightService fi
ghtService) {
        this.rouletteService = rouletteService;
        this.playerService = playerService;
        this.characterService = characterService;
        this.fightService = fightService;
    }

    @PostMapping("/roulette/pulls")
    public ResponseEntity<List<OwnedCharacter>> pullRoulette(@RequestParam long
playerId){
        if (playerId < 0 || !playerService.exists(playerId)) {
            return ResponseEntity.badRequest().build();
        }
        Optional<List<OwnedCharacter>> ownedCharacters = this.rouletteService.pu
llRoulette(playerId);
        if (ownedCharacters.isEmpty())
            return ResponseEntity.noContent().build();

        return ResponseEntity.ok(ownedCharacters.get());
    }


}
```

Fight controller

```
@RestController
public class FightController {

    private final RouletteService rouletteService;

    private final PlayerService playerService;

    private final CharacterService characterService;

    private final FightService fightService;

    public FightController(RouletteService rouletteService, PlayerService player
Service,
                           CharacterService characterService, FightService fight
Service) {
        this.rouletteService = rouletteService;
        this.playerService = playerService;
        this.characterService = characterService;
```

```
        this.fightService = fightService;
    }

    @PostMapping("/fights/seed")
    public void seedDatabase(){
        this.fightService.seedDatabase();
    }

    @GetMapping("/fights")
    public ResponseEntity<List<Fight>> possibleFights(){
        List<Fight> fights = fightService.getPossibleFights();
        return ResponseEntity.ok(fights);
    }

    @PostMapping("/players/{id}/characters/{code}/fights/{fightId}")
    public ResponseEntity<Player> fightAnEnemy(@PathVariable long id, @PathVaria
ble long code, @PathVariable long fightId){
        if (id < 0 || code < 0) return ResponseEntity.badRequest().build();
        try {
            return ResponseEntity.ok(fightService.fightEnemy(id, code, fightId))
;
        } catch (NoSuchElementException e) {
            return ResponseEntity.notFound().build();
        }

    }
}
```

Проектування та розробка рівня обслуговування

вступ

Сервісний рівень є ключовим компонентом багаторівневої архітектури, що відповідає за бізнес-логіку програми. Він містить усі правила й алгоритми, необхідні для виконання таких операцій, як керування користувачами та персонажами, проведення боїв і рулетки. Використання сервісного рівня дозволяє відокремити логіку обробки даних від веб-рівня, що спрощує обслуговування та розширення функціональних можливостей програми.

Теоретична с

Основними принципами проектування сервісного рівня є:

- **Інкапсуляція:**бізнес-логіка повинна бути прихована від зовнішніх компонентів, що дозволяє зменшити залежності між компонентами та підвищити безпеку.
- **Повторне використання:**послуги мають бути достатньо гнучкими, щоб їх можна було повторно використовувати в різних контекстах додатків.
- **Тестування:**послуги повинні легко тестуватися, дозволяючи швидко виявляти та виправляти помилки.
- **Масштабованість:**послуги повинні бути розроблені так, щоб їх можна було легко масштабувати в міру збільшення навантаження.

У нашій програмі сервісний рівень реалізовано за допомогою Spring Service, який забезпечує інверсію керування та залежностей. Це полегшує керування створенням і життєвим циклом служб, а також їхніми залежностями.

Реалізація

Player service

```
@Service
public class PlayerService {
    private final PlayerRepository playerRepository;

    private final OwnedCharacterRepository ownedCharacterRepository;

    private final CharacterRepository characterRepository;

    @Autowired
    public PlayerService(PlayerRepository playerRepository, OwnedCharacterReposi
tory ownedCharacterRepository, CharacterRepository characterRepository) {
        this.playerRepository = playerRepository;
        this.ownedCharacterRepository = ownedCharacterRepository;
        this.characterRepository = characterRepository;
    }

    public Character getDefaultCharacter() {
        return this.characterRepository.findByName("Vendy").orElseThrow(() -> ne
w RuntimeException("Default character not found"));
```

```
    }

    public boolean exists(long id) {
        return playerRepository.existsById(id);
    }

    public Player getPlayerById(long id) {
        return playerRepository.findById(id).get();
    }
    public void deletePlayer(long id) {
        playerRepository.deleteById(id);
    }



    public Player createPlayer() {
        Player player = playerRepository.save(new Player());

        ownedCharacterRepository.save(new OwnedCharacter(new OwnedCharacterId(ge
tDefaultCharacter(), player)));
        return player;
    }

    public int getPlayerBalance(long id){
        Player player = playerRepository.findById(id).orElse(null);
        return player.getBalance();
    }



      public List<OwnedCharacter> getPlayerCharacters(long id){
        return ownedCharacterRepository.findAllByOwnedCharacterIdPlayerId(id);
    }



    public List<OwnedCharacter> sellGivenCharacter(long id, long code){
        if(code == getDefaultCharacter().code) {throw new IllegalStateException(
"This is your default character, it can't be sold!");}
        Player player = playerRepository.findById(id).orElseThrow();
        Character character = characterRepository.findById(code).orElseThrow();
        OwnedCharacter ownedCharacter = ownedCharacterRepository.findById(new Ow
nedCharacterId(character, player)).orElseThrow();
        player.setBalance((int) (player.getBalance() + character.getSELL_COEFF()
));
        ownedCharacterRepository.delete(ownedCharacter);
        playerRepository.save(player);

        return ownedCharacterRepository.findAllByOwnedCharacterIdPlayerId(id);
    }

    public OwnedCharacter upgradeOwnedCharacter(long id, long code){
        Player player = playerRepository.findById(id).orElseThrow();
        Character character = characterRepository.findById(code).orElseThrow();
```

```
        OwnedCharacter ownedCharacter = ownedCharacterRepository.findById(new Ow
nedCharacterId(character, player)).orElseThrow();

        if(player.getBalance()<500){
            throw new IllegalStateException("You don't have enough money for an
upgrade!");
        }
        player.setBalance(player.getBalance()-500);
        ownedCharacter.setDamage(ownedCharacter.getDamage()+1);
        ownedCharacter.setHealth(ownedCharacter.getHealth()+4);
        ownedCharacter.setStamina(ownedCharacter.getStamina()+2);
        ownedCharacter = ownedCharacterRepository.save(ownedCharacter);
        playerRepository.save(player);

        return ownedCharacter;
    }

}
```

Character service

```
@Service
public class CharacterService {

    private final CharacterRepository characterRepository;
    private final OwnedCharacterRepository ownedCharacterRepository;

    private final PlayerRepository playerRepository;


    public List<Character> possibleCharacters =
            List.of(new Character("Raiden Shogun", RARE, 38, 20, 8, 4000,raidenA
scii()),
                    new Character("Shrek", LEGENDARY, 50, 25, 9, 5000, shrekAsci
i()),
                    new Character("Ayanami Rei", REGULAR, 30, 15, 5, 2000, ayana
miAscii()),
                    new Character("Mitsuri Kanrodji", REGULAR, 27, 15, 6, 2500,
mitsuriAscii()),
                    new Character("Boa Hancock", LEGENDARY, 50, 25, 10, 5500, bo
aAscii()),
                    new Character("Asuka", RARE, 38, 20, 7, 3500, asukaAscii()),
                    new Character("Nezuko", RARE, 40, 20,6, 3500, nezukoAscii())
,
                    new Character("Sakura Haruno", REGULAR, 30, 15,5, 2000, saku
raAscii()),
                    new Character("Yumeko Yabami", REGULAR, 30, 15,5, 2000, yume
koAscii() ),
                    new Character("Vendy", REGULAR, 25, 15,4, 1500, vendyAscii()
)
            );
```

```
    @Autowired
    public CharacterService(CharacterRepository characterRepository, OwnedCharac
terRepository ownedCharacterRepository,
                            PlayerRepository playerRepository){
        this.characterRepository = characterRepository;
        this.ownedCharacterRepository = ownedCharacterRepository;
        this.playerRepository = playerRepository;
    }

    public List<Character> getPossibleCharacters() {
        return this.characterRepository.findAll();
    }

    public Character getCharacterById(long code) {
        return characterRepository.findById(code).get();
    }
    public void deleteCharacter(long code) {
        characterRepository.deleteById(code);
    }

    public void seedDatabase(){
        for(Character character : possibleCharacters){
            this.characterRepository.save(character);
        }
    }

    public String getCharacterAppearance(long code){
        Character character = characterRepository.findById(code).orElse(null);
        return character.getAppearance();
    }

    public List<OwnedCharacter> addCharacterToPlayer(long id, long code){
        Player player = playerRepository.findById(id).orElseThrow();
        Character character = characterRepository.findById(code).orElseThrow();
        OwnedCharacterId compositeId = new OwnedCharacterId(character, player);

        if (ownedCharacterRepository.existsById(compositeId))
            throw new IllegalStateException("Cannot save duplicate characters.")
;


        OwnedCharacter ownedCharacter = new OwnedCharacter(new OwnedCharacterId(
character, player));
        ownedCharacterRepository.save(ownedCharacter);

        return ownedCharacterRepository.findAllByOwnedCharacterIdPlayerId(id);
    }
```

```
    public boolean exists(long code) {
        return characterRepository.existsById(code);
    }
}
```

Roulette service

```
@Service
public class RouletteService {

    private final OwnedCharacterRepository ownedCharacterRepository;

    private final PlayerRepository playerRepository;

    private final CharacterRepository characterRepository;

    private final CharacterService characterService;
    @Autowired
    public RouletteService(OwnedCharacterRepository ownedCharacterRepository, Pl
ayerRepository playerRepository,
                           CharacterRepository characterRepository, CharacterSer
vice characterService) {
        this.ownedCharacterRepository = ownedCharacterRepository;
        this.playerRepository = playerRepository;
        this.characterRepository = characterRepository;
        this.characterService = characterService;
    }

    public Optional<List<OwnedCharacter>> pullRoulette(long id) {
        Player player = playerRepository.findById(id).orElseThrow();
        List<OwnedCharacter> ownedCharacters = ownedCharacterRepository.findAllB
yOwnedCharacterIdPlayerId(id);
        if (player.getBalance() < 50) {
            throw new IllegalStateException("You don't have enough gold for a pu
ll");
        }
        Random rng = new Random();
        int roll = rng.nextInt(100);

        Optional<Character> character;
        if (roll <= 4) {
            character = this.getRandomCharacter(ownedCharacters, Rarity.LEGENDAR
Y);
        }
        else if (roll <=29){
            character = this.getRandomCharacter(ownedCharacters, Rarity.RARE);
        }
        else
            character = this.getRandomCharacter(ownedCharacters, Rarity.REGULAR)
;
        //duplicates
        if (character.isEmpty()) {
```

```
            player.setBalance(player.getBalance() + 200);
            playerRepository.save(player);
            return Optional.empty();
        } else {
            return Optional.of(characterService.addCharacterToPlayer(id, charact
er.get().code)); //add character to collection
        }
    }

    private Optional<Character> getRandomCharacter(List<OwnedCharacter> ownedCha
racters, Rarity rarity) {
        Random rng = new Random();
        List<Character> legendaryCharacters = this.characterRepository.findAllBy
Rarity(rarity);
        int roll = rng.nextInt(legendaryCharacters.size());
        Character rolledChar = legendaryCharacters.get(roll);
        if (ownedCharacters.stream()
                .map(ownedCharacter -> ownedCharacter.ownedCharacterId.character
)
                .toList().contains(rolledChar)) {
            return Optional.empty();
        }
        return Optional.of(rolledChar);
    }
}
```

Fight service

```
@Service
public class FightService {

    private final CharacterRepository characterRepository;
    private final OwnedCharacterRepository ownedCharacterRepository;

    private final PlayerRepository playerRepository;

    private final FightRepository fightRepository;
    @Autowired
    public FightService(CharacterRepository characterRepository, OwnedCharacterR
epository ownedCharacterRepository,
                        PlayerRepository playerRepository, FightRepository fight
Repository) {
        this.characterRepository = characterRepository;
        this.ownedCharacterRepository = ownedCharacterRepository;
        this.playerRepository = playerRepository;
        this.fightRepository = fightRepository;
    }

    public List<Fight> getPossibleFights() {
        return this.fightRepository.findAll();
    }
    private List<Fight> getFights() {
        List<Fight> enemiesList = List.of(
```

```
                new Fight(new Character("YaeMiko", RARE, 45, 20, 6, 0, yaeAscii(
))),
                new Fight(new Character("Mikasa", LEGENDARY, 55, 25, 10, 0, mika
saAscii())),
                new Fight(new Character("Lucy", REGULAR, 25, 10, 5, 0, lucyAscii
()))));
        return enemiesList;
    }
    public void seedDatabase(){
        List<Fight> fights = new LinkedList<>();
        for (Fight fight : getFights()) {
            fight.enemyCharacter = this.characterRepository.save(fight.enemyChar
acter);
            fights.add(fight);
        }

        for(Fight fight : fights){
            this.fightRepository.save(fight);
        }
    }
    public Player fightEnemy(long id, long code, long fightId){
        Player player = playerRepository.findById(id).orElseThrow();
        Character character = characterRepository.findById(code).orElseThrow();
        OwnedCharacter ownedCharacter = ownedCharacterRepository.findById(new Ow
nedCharacterId(character, player)).orElseThrow();
        Fight fight = fightRepository.findById(fightId).orElseThrow();

        Random rng = new Random();
        int charHealth = ownedCharacter.getHealth();
        int enemyHealth = fight.enemyCharacter.health;
        int charStamina = ownedCharacter.getStamina();
        int enemyStamina = fight.enemyCharacter.getStamina();
        int charDamage;
        int enemyDamage;
        boolean isVictory = false;

        while (charHealth > 0 && enemyHealth > 0) {
            // step 1: player turn
            charDamage = rng.nextInt(ownedCharacter.getDamage() + Math.max(owned
Character.getStamina(), 0));
            charDamage = charDamage > ownedCharacter.getDamage() ? ownedCharacte
r.getDamage() : charDamage;

            enemyHealth -= charDamage;
            charStamina -= 3;

            if (enemyHealth < 0) {
                isVictory = true;
                break;
            }

            // step 1: enemy turn
```

```
            enemyDamage = rng.nextInt(fight.enemyCharacter.getDamage() + Math.ma
x(fight.enemyCharacter.getStamina(), 0));
            enemyDamage = charDamage > fight.enemyCharacter.getDamage() ? fight.
enemyCharacter.getDamage() : enemyDamage;

            charHealth -= enemyDamage;
            enemyStamina -= 3;
        }

        if (isVictory){
            player.setBalance(player.getBalance()+100);

        }  else  player.setBalance(player.getBalance()-50);



        return playerRepository.save(player);

    }
}
```

Проектування та розробка рівня зберігання

вступ

Рівень зберігання відповідає за зберігання та керування даними в програмі. Він забезпечує взаємодію з базою даних, виконання операцій CRUD (Create, Read, Update, Delete) і підтримує цілісність даних. Наша програма використовує реляційну базу даних і взаємодіє з нею за допомогою JPA (Java Persistence API) і Spring Data JPA.

Теоретичний

Основними принципами проектування шару зберігання є:

- **Нормалізація даних:**Процес організації даних у базі даних для мінімізації надлишкових даних і забезпечення їх цілісності.
- **Зв'язки між таблицями:**встановлення асоціацій між різними таблицями бази даних для забезпечення цілісності даних.
- **Транзакційність:**забезпечення того, що всі операції з даними виконуються атомарно, тобто або вносяться всі зміни, або не вносяться жодні.
- **Кешування:**Використання кешу для зменшення навантаження на базу даних і підвищення продуктивності системи.

  Наша програма використовує такі основні компоненти:

- **Класи сутностей:**Класи Java, що відповідають таблицям у базі даних.
- **Інтерфейси репозиторію:**інтерфейси для виконання операцій CRUD з даними.
- **JPQL (мова збереження запитів Java):**мова запитів для взаємодії з даними в базі даних.

Реалізація

Entities

Player

```
@Entity
public class Player implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)

    public long id;
    public int balance;


    public Player() {
        this.balance = 100;
    }
```

```
    public int getBalance() {
        return balance;
    }

    public void setBalance(int balance) {
        this.balance = balance;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Player player = (Player) o;
        return id == player.id && balance == player.balance;
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, balance);
    }

    @Override
    public String toString() {
        return "Player{" +
                "id=" + id +
                ", balance=" + balance +
                '}';
    }
}
```

Character

```
@Entity
public class Character implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    public long code;
    public String name;
    public Rarity rarity;

    public int health;
    public int stamina;
    public int damage;
    @Column(length = 10000)
    public String appearance;
    public int price;

    public final float SELL_COEFF = (float) 0.65;
```

```
    public Character() {
        //for object mappers
    }

    public Character( String name, Rarity rarity,  int health, int stamina, int
damage, int price, String appearance) {
        this.name = name;
        this.rarity = rarity;
        this.health = health;
        this.stamina = stamina;
        this.damage = damage;
        this.price = price;
        this.appearance = appearance;
    }

    public String getName() {
        return name;
    }

    public Rarity getRarity() {
        return rarity;
    }

    public int getHealth() {
        return health;
    }

    public int getStamina() {
        return stamina;
    }

    public int getDamage() {
        return damage;
    }

    public int getPrice() {
        return price;
    }

    public float getSELL_COEFF() {
        return SELL_COEFF;
    }

    public void setHealth(int health) {
        this.health = health;
    }

    public void setStamina(int stamina) {
        this.stamina = stamina;
    }
```

```
    public void setDamage(int damage) {
        this.damage = damage;
    }

    public String getAppearance() {
        return appearance;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Character character = (Character) o;
        return code == character.code && health == character.health && stamina =
= character.stamina
                && damage == character.damage && price == character.price
                && Objects.equals(name, character.name) && rarity == character.r
arity
                && Objects.equals(appearance, character.appearance);
    }

    @Override
    public int hashCode() {
        return Objects.hash(code, name, rarity, health, stamina, damage, appeara
nce, price, SELL_COEFF);
    }

    @Override
    public String toString() {
        return "Character{" +
                "code=" + code +
                ", name='" + name + '\'' +
                ", rarity=" + rarity +
                ", health=" + health +
                ", stamina=" + stamina +
                ", damage=" + damage +
                ", appearance='" + appearance + '\'' +
                ", price=" + price +
                ", SELL_COEFF=" + SELL_COEFF +
                '}';
    }
}
```

Fight

```
@Entity
public class Fight  implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    public long id;
```

```
    @ManyToOne
    @JoinColumn(name = "code", referencedColumnName = "code")
    public Character enemyCharacter;

    public Fight() {
    }

    public Fight(Character enemyCharacter) {
        this.enemyCharacter = enemyCharacter;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Fight fight = (Fight) o;
        return id == fight.id && Objects.equals(enemyCharacter, fight.enemyChara
cter);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, enemyCharacter);
    }

    @Override
    public String toString() {
        return "Fight{" +
                "id=" + id +
                ", enemyCharacter=" + enemyCharacter +
                '}';
    }
}
```

OwnedCharacter

```
@Entity
public class OwnedCharacter implements Serializable {

    @EmbeddedId
    public OwnedCharacterId ownedCharacterId;

    public String name;
    public Rarity rarity;

    public int health;
    public int stamina;
    public int damage;
    @Column(length = 10000)
    public String appearance;
```

```java
public int price;

public final float SELL_COEFF = (float) 0.65;

public OwnedCharacter() {

}

public OwnedCharacter(OwnedCharacterId ownedCharacterId) {
    this.ownedCharacterId = ownedCharacterId;
    this.name = this.ownedCharacterId.character.name;
    this.rarity = this.ownedCharacterId.character.rarity;
    this.health = this.ownedCharacterId.character.health;
    this.stamina = this.ownedCharacterId.character.stamina;
    this.damage = this.ownedCharacterId.character.damage;
    this.appearance = this.ownedCharacterId.character.appearance;
    this.price = this.ownedCharacterId.character.price;
}

public int getHealth() {
    return health;
}

public void setHealth(int health) {
    this.health = health;
}

public int getStamina() {
    return stamina;
}

public void setStamina(int stamina) {
    this.stamina = stamina;
}

public int getDamage() {
    return damage;
}

public void setDamage(int damage) {
    this.damage = damage;
}



@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    OwnedCharacter that = (OwnedCharacter) o;
    return Objects.equals(ownedCharacterId, that.ownedCharacterId);
```

```
    }

    @Override
    public int hashCode() {
        return Objects.hash(ownedCharacterId);
    }

}
```

OwnedCharacterId

```
@Embeddable
public class OwnedCharacterId implements Serializable {

    @ManyToOne
    @MapsId("characterCode")
    @JoinColumn(name = "code", referencedColumnName = "code")
    public Character character;

    @ManyToOne
    @MapsId("playerId")
    @JoinColumn(name = "id",referencedColumnName = "id")
    public Player player;

    public OwnedCharacterId() {
    }

    public OwnedCharacterId(Character character, Player player) {
        this.character = character;
        this.player = player;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        OwnedCharacterId that = (OwnedCharacterId) o;
        return Objects.equals(character, that.character) &&
                Objects.equals(player, that.player);
    }

    @Override
    public int hashCode() {
        return Objects.hash(character, player);
    }
}
```

Also the class Character is using enum Rarity for splitting characters on 3 categories: Regular, Rare and Legendary.

Enum Rarity

```
public enum Rarity {

    REGULAR,
    RARE,
    LEGENDARY

}
```

Repository interfaces

Character repository

```
@Repository
public interface CharacterRepository extends JpaRepository<Character, Long> {
    public Optional<Character> findByName(String name);
    public List<Character> findAllByRarity(Rarity rarity);
}
```

Player repository

```
@Repository
public interface PlayerRepository extends JpaRepository<Player, Long> {
}
```

Fight repository

```
@Repository
public interface FightRepository extends JpaRepository<Fight, Long> {

}
```

OwnedCharacter repository

```
@Repository
public interface OwnedCharacterRepository extends JpaRepository<OwnedCharacter,
OwnedCharacterId> {

    public List<OwnedCharacter> findAllByOwnedCharacterIdPlayerId(long id);
}
```

Проектування та розробка журналювання

вступ

Ведення журналів є важливою частиною будь-якого програмного забезпечення, оскільки воно дозволяє відстежувати події та дії в системі, що допомагає виявляти та усувати помилки, контролювати продуктивність системи та гарантувати безпеку. У нашому додатку журналювання реалізовано за допомогою аспектно-орієнтованого програмування (AOP) у поєднанні з бібліотекою SLF4J і Logback.

Теоретичний

Аспектно-орієнтоване програмування (AOP) дозволяє відокремити бізнес-логіку програми від технічної логіки, такої як журналювання, безпека, транзакції тощо. Основні поняття AOP:

- **Аспект:**Модуль, що містить технічну логіку, яку можна застосовувати до різних частин програми.
- **Точка приєднання:**Місце в коді, де можна застосувати аспект.
- **Pointcut:**Вираз, що визначає точки з'єднання, до яких буде застосовано аспект.
- **Порада:**код, що виконується в точках приєднання.

У випадку GacgaGirls програма Spring Boot може чудово працювати без таких файлів, як logback-spring.xml, оскільки Spring Boot надає стандартну конфігурацію журналювання, яка є достатньою для багатьох програм. Ми можемо налаштувати основні параметри журналювання у файлі application.properties:

Implementation

application.properties

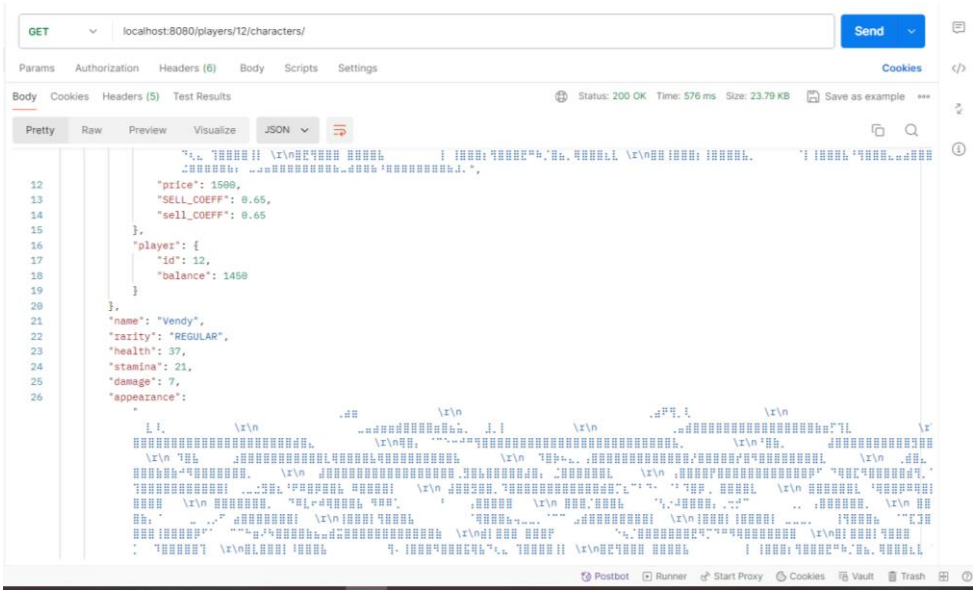
```
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# use one of these alternatives...
# ... purely in-memory, wiped on restart, but great for testing
#spring.datasource.url=jdbc:h2:mem:testdb
# ... persisted on disk (in project directory)
spring.datasource.url=jdbc:h2:file:./database

# enable DB view on http://localhost:8080/h2-console
spring.h2.console.enabled=true

# strategy for table (re-)generation
spring.jpa.hibernate.ddl-auto=update
# show auto-generated SQL commands
#spring.jpa.hibernate.show_sql=true
```

Приклади виконання програми

Для прикладу виконання було використано додаток Postman, який дозволяє надсилаті різноманітні запроси.

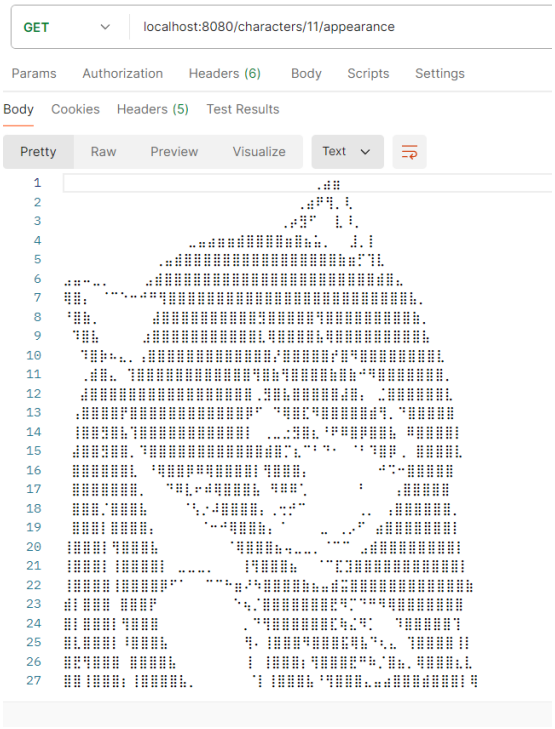
Приклад запросу localhost:8080/players/12/characters/

Виводяться усі персонажі, які має гравець та іх дані

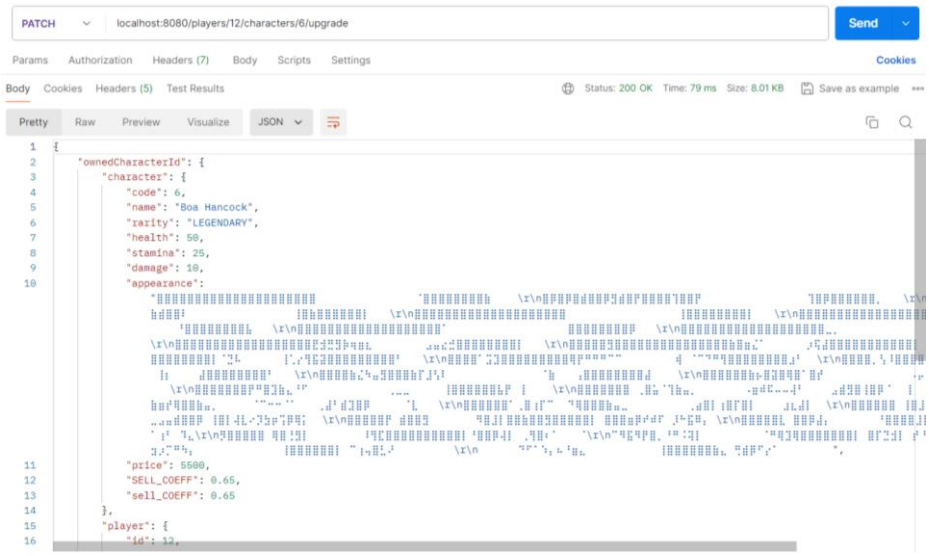


Приклад запросу localhost:8080/characters/11/appearance

Отримуємо зовнішність певного персонажа за айді



Приклад запросу для покращення статистик персонажа за ігрові кошти.



ВИСНОВКИ

Створення багаторівневого веб-додатку для управління банківськими транзакціями є складною і багатогранною задачею, яка включає розробку та впровадження різноманітних компонентів, кожен з яких відіграє важливу роль у загальній архітектурі системи. Цей проект дав мені змогу зануритися в різні аспекти розробки програмного забезпечення, вивчити та застосувати сучасні технології та інструменти на практиці.

Одним з найважливіших завдань проекту було розробити та впровадити веб-шар. Веб-рівень відповідає за обробку HTTP-запитів від клієнтів та повернення відповідей. Використання Spring Boot для реалізації веб-сервісів RESTful дозволило створити ефективний та гнучкий інтерфейс для взаємодії між клієнтом і сервером. Це забезпечило надійність і масштабованість системи, що є критично важливим для будь-якого сучасного веб-додатку.

Сервісний рівень забезпечує виконання бізнес-логіки додатку. Використання Spring Service та інверсії керування (IoC) дозволило організувати бізнес-логіку як окремі компоненти, які легко підтримувати, тестувати та розширювати. Інкапсуляція бізнес-логіки в сервіси підвищила надійність і безпеку системи, забезпечивши ефективне управління даними та операціями.

Рівень зберігання відповідає за зберігання та керування даними в додатку. Використання JPA та Spring Data JPA забезпечило ефективну взаємодію з базою даних, а також виконання операцій CRUD. Розробка та реалізація інтерфейсів Entity-класів і Repository дозволила забезпечити цілісність і узгодженість даних. Важливим аспектом було також забезпечення нормалізації даних, що дозволило мінімізувати надмірність даних і уникнути аномалій.

Цей проект дав мені можливість вивчити та реалізувати різні аспекти розробки багаторівневої веб-програми, від архітектурного дизайну до реалізації та тестування компонентів системи. Я здобув цінний досвід роботи з сучасними технологіями та інструментами, такими як Spring Boot, Spring Security, JPA та аспектно-орієнтоване програмування (AOP). Результатом моїх зусиль став функціональний та безпечний веб-додаток для управління банківськими операціями, який легко розширювати та підтримувати в майбутньому.

Проект також підкреслив важливість найкращих методів проектування, таких як інверсія керування, аспектно-орієнтоване програмування та належна обробка даних. Впровадження цих практик дозволяє створювати надійні, гнучкі та масштабовані системи. Розподіливши обов'язки між різними архітектурними рівнями, я зміг забезпечити високу якість коду та легкість обслуговування.

У майбутньому я планую продовжувати вдосконалювати програму, додаючи нові функції та покращуючи існуючі. Я також розглядаю можливість інтеграції з іншими службами та платформами для підвищення зручності і функціональності додатку.