25-08-2주차-진행현황-202058096-이재민

📱 1. 캡스톤 디자인 메인 UI 설계 및 구현

25-07-5주차-진행현황-202058096-이재민

🎭 1.1 프론트엔드 구현

CSS 디자인 시스템

- **Apple 스타일 글래스모피즘**: `backdrop-filter: blur(20px)` 활용한 반투명 효과
- **CSS 변수 시스템**: `--system-blue`, `--label-primary` 등 Apple 표준 색상 적용
- **다크모드 지원**: `@media (prefers-color-scheme: dark)` 활용
- **반응형 그리드**: `grid-template-columns: repeat(auto-fit, minmax(350px, 1fr))`

JavaScript 핵심 기능

헤더 프로필 드롭다운 (header.js)

```
toggleProfileDropdown() {
    const dropdownMenu = this.header.querySelector('.user-actions.dropdown-
menu');
    const isOpen = dropdownMenu.classList.contains('show');

    if (isOpen) {
        this.closeProfileDropdown();
    } else {
        this.openProfileDropdown();
    }
}
```

메인 콘텐츠 애니메이션 (main.js)

```
animateElementIn(element) {
    element.style.opacity = '0';
    element.style.transform = 'translateY(20px)';

    requestAnimationFrame(() => {
        element.style.transition = 'opacity 0.6s ease, transform 0.6s ease';
        element.style.opacity = '1';
        element.style.transform = 'translateY(0)';
    });
}
```

Vue.js 사이드바 위젯 시스템 (sidebar.js)

```
app.component('popular-games-widget', {
    props: ['games'],
    setup(props) {
        const listRef = ref(null);

        onMounted(() => {
            // 스태거드 애니메이션 적용
            setTimeout(animate, 100);
        });
    }
});
```

⚙️ 1.2 백엔드 구현

1.2.1 컨트롤러 아키텍처

게임 컨트롤러

```
package kr.plusb3b.games.gamehub.api.controller.game;

@Controller
@RequestMapping("/games")
public class GameController {
    // 현재는 빈 컨트롤러로 향후 게임 관련 기능 확장 예정
}
```

프로덕션 사용자 컨트롤러

```
package kr.plusb3b.games.gamehub.api.controller.user;

@Controller
@RequestMapping("/game-hub/prod")
public class UserProdController {

    //업로드 페이지로 이동
    @GetMapping("/upload")
    public String viewUploadPage(){
        return "game/upload/index";
    }
}
```

마이페이지 컨트롤러

```java
package kr.plusb3b.games.gamehub.api.controller.user;

@Controller
@RequestMapping("/my-page")
public class UserController {

    private final AccessControlService access;
    private final UserProvider userProvider;

    public UserController(AccessControlService access, UserProvider userProvider)
{
        this.access = access;
        this.userProvider = userProvider;
    }

    //프로필 변경 페이지 요청 처리
    @GetMapping("/edit-profile")
    public String showEditProfilePage(HttpServletRequest request, Model model) {
        User user = access.getAuthenticatedUser(request);
        UserDetailsDto userDetailsDto = userProvider.getUserDetails(user);
        model.addAttribute("userDetailsDto", userDetailsDto);

        return "profile/profile-edit-form";
    }
}
```

1.2.2 REST API 로그인 시스템

로그인 컨트롤러 전체 구조

```java
package kr.plusb3b.games.gamehub.api.controller.login.rest;

@RestController
@RequestMapping("/api/v1")
public class RestLoginController {

    @Value("${app.api.version}")
    private String appApiVersion;

    private final UserAuthRepository userAuthRepo;
    private final UserLoginServiceImpl userLoginService;

    public RestLoginController(UserAuthRepository userAuthRepo,
UserLoginServiceImpl userLoginService) {
        this.userAuthRepo = userAuthRepo;
        this.userLoginService = userLoginService;
    }
}
```

상세 로그인 검증 프로세스

```
@PostMapping("/login")
public ResponseEntity<?> checkLogin(@RequestParam("authUserId") String
authUserId,
                                    @RequestParam("authPassword") String
authPassword,
                                    HttpServletRequest request,
                                    HttpServletResponse response) {

    // 1. 유효성 검사: 빈 값 체크
    if (authUserId == null || authUserId.trim().isEmpty() ||
            authPassword == null || authPassword.trim().isEmpty()) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                .body("아이디 혹은 비밀번호가 누락되었습니다");
    }

    // 2. 아이디를 통해 DB에 존재하는지 확인
    Optional<UserAuth> userAuthOpt =
userLoginService.findUserAuthByLoginId(authUserId);
    if (userAuthOpt.isEmpty()) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND)
                .body("아이디가 존재하지 않습니다.");
    }

    UserAuth userAuth = userAuthOpt.get();

    // 3. 사용자 정보 가져오기
    Optional<User> userOpt = userLoginService.getUserByAuth(userAuth);
    if (userOpt.isEmpty()) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("사용자 정보를 찾을 수 없습니다.");
    }

    User user = userOpt.get();

    // 4. 계정의 활성화 상태 확인
    if (!userLoginService.isUserActivated(user)) {
        return ResponseEntity.status(HttpStatus.FORBIDDEN)
                .body("현재 탈퇴된 상태입니다. 관리자에게 문의해주세요.");
    }

    // 5. 비밀번호 검증
    if (!userLoginService.isPasswordMatch(userAuth, authPassword)) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body("비밀번호가 일치하지 않습니다.");
    }

    // 6. 로그인 성공 처리
    try {
        // JWT 토큰 발급 및 쿠키 설정
        userLoginService.issueJwtCookie(response, authUserId);

        // 로그인 기록 저장
```

```
        boolean isLoginHistorySaved = userLoginService.saveLoginHistory(user,
request);
        if (!isLoginHistorySaved) {
            System.err.println("로그인 기록 저장에 실패했습니다. 사용자 ID: " +
authUserId);
        }

        // 성공 응답에 사용자 역할 정보 포함
        Map<String, Object> loginResponse = new HashMap<>();
        loginResponse.put("message", "로그인 성공");
        loginResponse.put("role", user.getMbRole().toString());
        loginResponse.put("nickname", user.getMbNickname());

        return ResponseEntity.ok(loginResponse);

    } catch (Exception e) {
        System.err.println("로그인 처리 중 오류 발생: " + e.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("로그인 처리 중 오류가 발생했습니다.");
    }
}
```

1.2.3 역할 기반 리다이렉트 처리

헤더 템플릿의 로그인 상태 분기

```
<!-- 로그인 상태가 아닐 때 -->
<div class="guest-user" th:if="${not isLoggedIn}">
    <a href="/game-hub/login" class="login-btn btn-secondary">로그인</a>
    <a href="/game-hub/join" class="signup-btn btn-primary">회원가입</a>
    <a href="/game-hub/join-prod" class="team-signup-btn btn-outline">팀 회원가입
</a>
</div>

<!-- 로그인 상태일 때 -->
<div class="logged-user" th:if="${isLoggedIn}">
    <div class="user-profile">
        <img th:src="${profileImage}" alt="프로필 이미지" class="profile-img">
        <span class="user-nickname" th:text="${nickname}">닉네임</span>
    </div>
</div>
```

프로덕션 페이지 접근 제어

```
<!-- index-prod.html -->
<h2>PROD PAGE</h2>
<button type="button" id="game-upload" name="game-upload">게임 업로드 하기
</button>

<script>
```

```
document.getElementById("game-upload").addEventListener("click", function () {
    window.location.href = "/game-hub/prod/upload";
})
</script>
```

1.2.4 서비스 계층 의존성 구조

AccessControlService

- JWT 토큰 검증 및 사용자 정보 추출
- `getAuthenticatedUser(HttpServletRequest request)` 메서드 제공

UserLoginServiceImpl

- `findUserAuthByLoginId(String loginId)` - 로그인 ID로 인증 정보 조회
- `getUserByAuth(UserAuth userAuth)` - 인증 정보로 사용자 조회
- `isUserActivated(User user)` - 사용자 활성화 상태 확인
- `isPasswordMatch(UserAuth userAuth, String password)` - 비밀번호 검증
- `issueJwtCookie(HttpServletResponse response, String userId)` - JWT 토큰 발급
- `saveLoginHistory(User user, HttpServletRequest request)` - 로그인 기록 저장

UserProvider

- `getUserDetails(User user)` - 사용자 상세 정보 DTO 변환

---

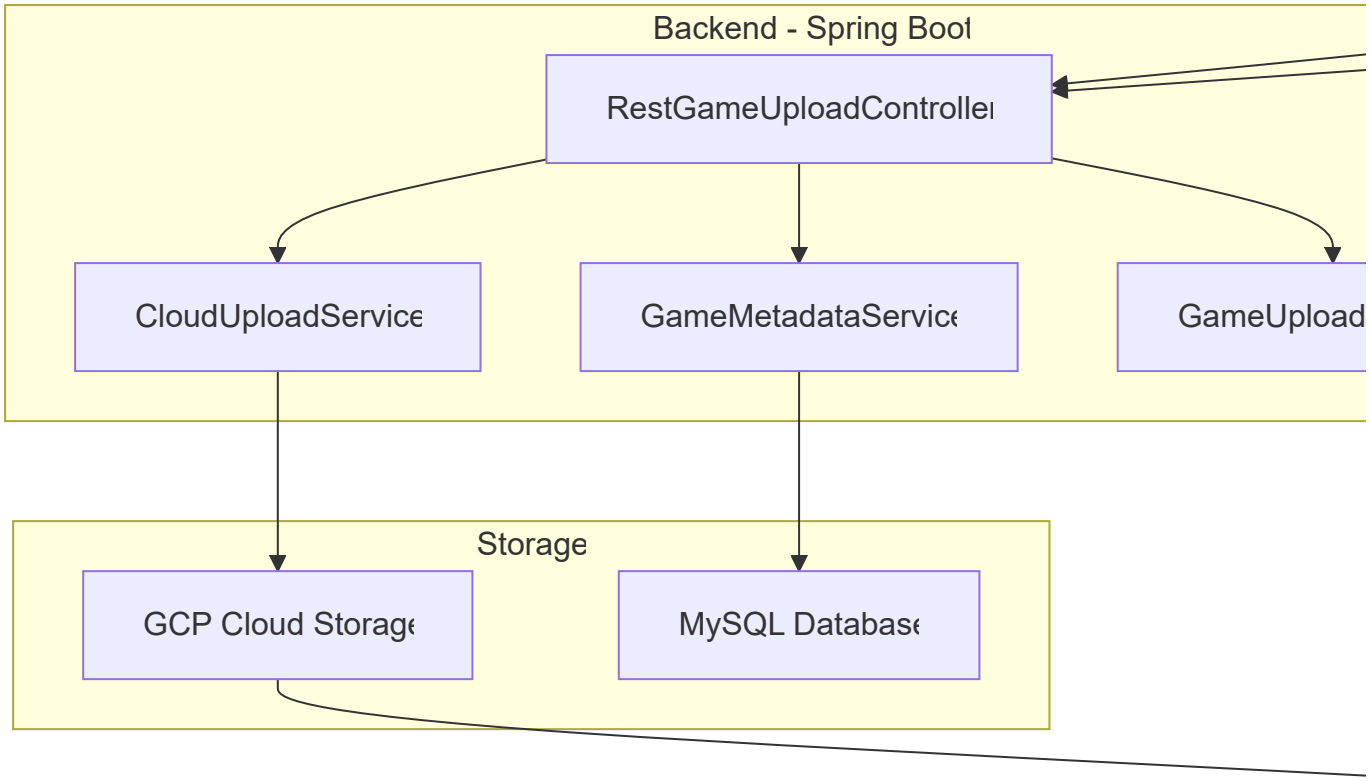
2. 게임 파일 업로드 및 실행 구조 검토

📋 시스템 개요

WebGL 게임을 업로드하고 실행할 수 있는 플랫폼을 구축하기 위해 **Spring Boot**와 **Google Cloud Platform (GCP)**을 활용한 시스템을 설계하고 구현했습니다. 이 시스템은 게임 개발자가 WebGL 빌드 파일을 업로드하면, 관리자 승인을 거쳐 사용자들이 브라우저에서 직접 게임을 플레이할 수 있도록 합니다.

2.1 업로드 및 파일 실행 구조 설계

🧬 전체 시스템 아키텍처

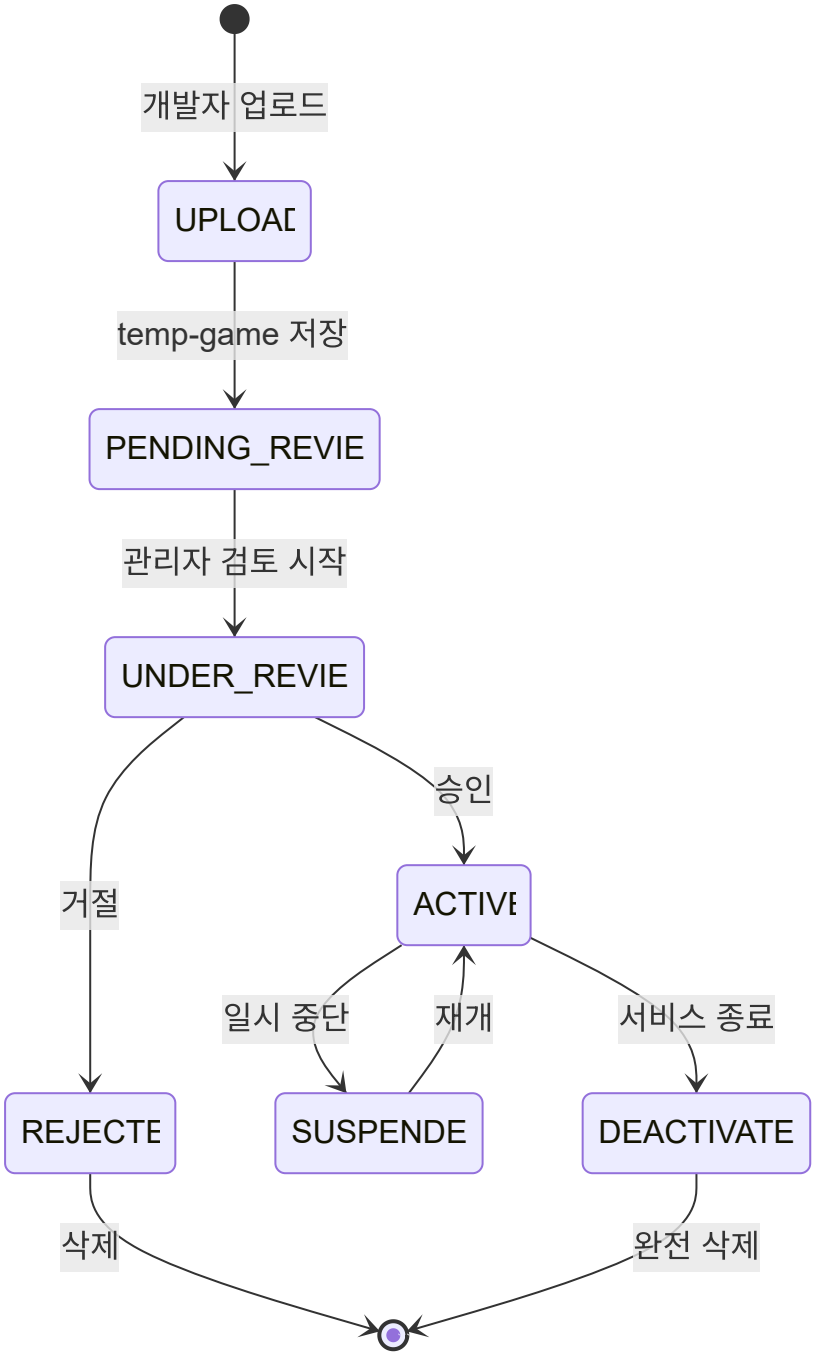


📁 GCS 폴더 구조 설계

```
gs://game-webgl-bucket-capstone/
│
├── 📁 temp-game/              # 임시 저장 (승인 대기)
│   ├── 1734567890_abc123.zip
│   └── 1734567891_def456.zip
│
├── 📁 activate-game/          # 활성 게임 (승인 완료)
│   ├── game-123/
│   │   ├── index.html
│   │   ├── Build/
│   │   └── TemplateData/
│   └── game-124/
```

```
│
└── 📁 deactivate-game/          # 비활성 게임 (서비스 종료)
    └── game-125/
```

🔄 게임 생명주기 (Game Lifecycle)



2.2 GCP 세팅

🔧 application.properties 설정

```
# Spring Boot 파일 업로드 설정
spring.servlet.multipart.max-file-size=100MB
```

```
spring.servlet.multipart.max-request-size=100MB
server.tomcat.max-swallow-size=100MB
server.tomcat.max-http-post-size=100MB

# Google Cloud Platform 설정
spring.cloud.gcp.project-id=your-project-id
spring.cloud.gcp.credentials.location=classpath:gcp-service-account-key.json
spring.cloud.gcp.storage.bucket=game-webgl-bucket-capstone

# 커스텀 경로 설정
app.temp.game.path=temp-game
app.activate.game.path=activate-game
app.deactivate.game.path=deactivate-game
```

☁️ GcsConfig.java - Google Cloud Storage 설정

```
package kr.plusb3b.games.gamehub.upload.googleCloud;

import com.google.auth.oauth2.GoogleCredentials;
import com.google.cloud.storage.Storage;
import com.google.cloud.storage.StorageOptions;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.Resource;

import java.io.IOException;

@Configuration
public class GcsConfig {

    @Value("${spring.cloud.gcp.project-id}")
    private String projectId;

    @Value("${spring.cloud.gcp.credentials.location}")
    private Resource credentialsPath;

    /**
     * Google Cloud Storage 클라이언트 빈 생성
     * Service Account 인증을 통해 GCS에 접근할 수 있는 Storage 객체를 생성합니다.
     */
    @Bean
    public Storage storage() throws IOException {
        // Service Account JSON 키 파일을 통한 인증
        GoogleCredentials credentials = GoogleCredentials
                .fromStream(credentialsPath.getInputStream());

        // Storage 클라이언트 생성 및 반환
        return StorageOptions.newBuilder()
                .setProjectId(projectId)
                .setCredentials(credentials)
                .build()
```

```
                .getService();
    }
}
```

🔐 Google Cloud Console 설정 사항

1. **Service Account 생성**
   - IAM & Admin → Service Accounts → Create Service Account
   - Role: `Storage Admin` 권한 부여
   - JSON 키 생성 및 다운로드
2. **Storage Bucket 생성**
   - Cloud Storage → Create Bucket
   - Name: `game-webgl-bucket-capstone`
   - Location: `asia-northeast3` (서울)
   - Access Control: `Uniform` (버킷 레벨 권한)
3. **버킷 공개 설정**
   - Permissions → Grant Access
   - Principal: `allUsers`
   - Role: `Storage Object Viewer`

2.3 Games 엔티티 보완 및 GamesFile 엔티티 추가

🎮 Games.java - 게임 메타데이터 엔티티

```java
package kr.plusb3b.games.gamehub.domain.game.entity;

import jakarta.persistence.*;
import kr.plusb3b.games.gamehub.domain.board.entity.Board;
import kr.plusb3b.games.gamehub.domain.user.entity.User;
import lombok.Getter;
import lombok.Setter;

import java.time.LocalDateTime;

@Entity
@Getter
@Setter
@Table(name = "games")
public class Games {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "game_id", nullable = false, unique = true)
    private Long gameId;

    // 🔗 연관관계 매핑
```

```java
    @ManyToOne
    @JoinColumn(name = "board_id")
    private Board board;  // 게시판 연결

    @ManyToOne
    @JoinColumn(name = "mb_id")
    private User user;    // 업로더 정보

    // 📝 게임 기본 정보
    @Column(name = "game_name", nullable = false, length = 200)
    private String gameName;

    @Column(name = "game_description", columnDefinition = "TEXT")
    private String gameDescription;

    @Column(name = "team_name", nullable = false, length = 100)
    private String teamName;

    @Column(name = "specs", length = 500)
    private String specs;  // 권장 사양

    @Column(name = "game_version", length = 50)
    private String gameVersion;

    @Column(name = "genre", nullable = false, length = 50)
    private String genre;

    @Column(name = "platform", nullable = false, length = 20)
    private String platform;  // PC, Mobile

    // 📅 시간 정보
    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;

    @Column(name = "approved_at")
    private LocalDateTime approvedAt;

    // 🚦 상태 관리
    @Enumerated(EnumType.STRING)
    @Column(name = "status", nullable = false, length = 20)
    private GameStatus status;

    @Column(name = "is_visible", nullable = false)
    private int isVisible;  // 0: 숨김, 1: 공개

    // 🔄 양방향 관계 설정 (Games ↔ GamesFile)
    @OneToOne(mappedBy = "game", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private GamesFile gamesFile;

    // 📊 GameStatus Enum
    public enum GameStatus {
        PENDING_REVIEW,    // 승인 대기
```

```
        UNDER_REVIEW,       // 검토 중
        ACTIVE,             // 활성
        REJECTED,           // 거절
        SUSPENDED,          // 일시 중단
        DEACTIVATED,        // 서비스 종료
        UPLOAD_FAILED       // 업로드 실패
    }

    // 🛠️ 편의 메서드
    public boolean isStatusPendingReview() {
        return this.status == GameStatus.PENDING_REVIEW;
    }

    public boolean isStatusActive() {
        return this.status == GameStatus.ACTIVE;
    }

    public boolean isVisible() {
        return this.isVisible == 1;
    }
}
```

📁 GamesFile.java - 게임 파일 정보 엔티티

```
package kr.plusb3b.games.gamehub.domain.game.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.time.LocalDateTime;

@Entity
@Getter
@Setter
@Table(name = "games_file")
public class GamesFile {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long fileId;

    // 🔗 Games와 1:1 관계
    @OneToOne
    @JoinColumn(name = "game_id", nullable = false, unique = true)
    private Games game;

    // 🔐 파일 무결성 정보
    @Column(name = "game_hash", unique = true, length = 64, nullable = false)
    private String gameHash;  // SHA-256 해시값

    @Column(name = "original_filename", length = 255)
```

```
    private String originalFilename;

    // 📍 저장 위치 정보
    @Column(name = "game_url", length = 500)
    private String gameUrl;  // GCS Public URL

    @Column(name = "file_size")
    private Long fileSize;  // 바이트 단위

    @Column(name = "uploaded_at", nullable = false)
    private LocalDateTime uploadedAt;

    // 🚦 파일 상태
    @Enumerated(EnumType.STRING)
    @Column(name = "file_status", nullable = false, length = 20)
    private FileStatus fileStatus = FileStatus.TEMP;

    // 📊 FileStatus Enum
    public enum FileStatus {
        TEMP,         // temp-game 폴더
        ACTIVE,       // activate-game 폴더
        DEACTIVATED   // deactivate-game 폴더
    }
}
```

2.4 Service Interface 설계

🎯 GameMetadataService.java - 게임 메타데이터 서비스 인터페이스

```
package kr.plusb3b.games.gamehub.domain.game.service;

import kr.plusb3b.games.gamehub.domain.board.entity.Board;
import kr.plusb3b.games.gamehub.domain.game.dto.GameUploadDto;
import kr.plusb3b.games.gamehub.domain.game.dto.GamesInfoDto;
import kr.plusb3b.games.gamehub.domain.game.entity.Games;
import kr.plusb3b.games.gamehub.domain.game.entity.GamesFile;
import kr.plusb3b.games.gamehub.domain.game.vo.GamesVO;
import kr.plusb3b.games.gamehub.domain.user.entity.User;

import java.util.Optional;

public interface GameMetadataService {

    /**
     * 게임 메타정보를 DB에 저장
     * @param dto 업로드 요청 데이터
     * @param gvo 기본값 설정 객체
     * @param user 업로더 정보
     * @param board 게시판 정보
```

```
     * @return 저장된 Games 엔티티
     */
    Games saveGameToDB(GameUploadDto dto, GamesVO gvo, User user, Board board);

    /**
     * 파일 정보를 DB에 저장
     * @param game 연관된 게임 엔티티
     * @param gamesFile GCS 업로드 결과
     * @return 저장된 GamesFile 엔티티
     */
    GamesFile saveGameFileToDB(Games game, GamesFile gamesFile);

    /**
     * 승인 대기 중인 게임 목록 조회
     * @return 승인 대기 게임 정보
     */
    Optional<GamesInfoDto> notApprovedGames();

    /**
     * 승인된 활성 게임 목록 조회
     * @return 활성 게임 정보
     */
    Optional<GamesInfoDto> approvedGames();
}
```

☁️ CloudUploadService.java - 클라우드 업로드 서비스 인터페이스

```
package kr.plusb3b.games.gamehub.upload.googleCloud;

import kr.plusb3b.games.gamehub.domain.game.entity.GamesFile;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

public interface CloudUploadService {

    /**
     * 단일 파일을 GCP에 업로드
     * @param file 업로드할 파일
     * @return 파일 정보 객체
     */
    GamesFile uploadFileToGCP(MultipartFile file);

    /**
     * 여러 파일을 GCP에 업로드
     * @param files 업로드할 파일 목록
     * @return 파일 정보 객체 리스트
     */
    List<GamesFile> uploadFileToGCP(List<MultipartFile> files);

    /**
     * 게임 승인 시 activate 폴더로 이동
```

```
     * @param gameId 게임 ID
     * @param currentUrl 현재 파일 URL
     * @return 새로운 파일 URL
     */
    String moveFileToActivateFolder(String gameId, String currentUrl);

    /**
     * 게임 비활성화 시 deactivate 폴더로 이동
     * @param gameId 게임 ID
     * @param currentUrl 현재 파일 URL
     * @return 새로운 파일 URL
     */
    String moveFileToDeactivateFolder(String gameId, String currentUrl);
}
```

✅ GameUploadValidator.java - 파일 검증 서비스 인터페이스

```
package kr.plusb3b.games.gamehub.domain.game.service;

import org.springframework.web.multipart.MultipartFile;

public interface GameUploadValidator {

    /**
     * WebGL 게임 구조 검증 (index.html 존재 확인)
     * @param zipFile 검증할 ZIP 파일
     * @return index.html 존재 여부
     */
    boolean isIndexHtml(MultipartFile zipFile);

    /**
     * 유효한 게임 ZIP 파일인지 검증
     * @param file 검증할 파일
     * @return 유효성 여부
     */
    boolean isValidGameZip(MultipartFile file);
}
```

2.5 Service Implementation

💾 GameMetadataServiceImpl.java - 게임 메타데이터 서비스 구현

```
package kr.plusb3b.games.gamehub.application.game;

import kr.plusb3b.games.gamehub.domain.board.entity.Board;
import kr.plusb3b.games.gamehub.domain.game.dto.GameUploadDto;
import kr.plusb3b.games.gamehub.domain.game.dto.GamesInfoDto;
import kr.plusb3b.games.gamehub.domain.game.entity.Games;
```

```java
import kr.plusb3b.games.gamehub.domain.game.entity.GamesFile;
import kr.plusb3b.games.gamehub.domain.game.exception.GameUploadException;
import kr.plusb3b.games.gamehub.domain.game.repository.GamesRepository;
import kr.plusb3b.games.gamehub.domain.game.repository.GamesFileRepository;
import kr.plusb3b.games.gamehub.domain.game.service.GameMetadataService;
import kr.plusb3b.games.gamehub.domain.game.vo.GamesVO;
import kr.plusb3b.games.gamehub.domain.user.entity.User;
import lombok.extern.slf4j.Slf4j;
import org.springframework.dao.DataAccessException;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Objects;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
@Slf4j
public class GameMetadataServiceImpl implements GameMetadataService {

    private final GamesRepository gamesRepo;
    private final GamesFileRepository gamesFileRepo;

    public GameMetadataServiceImpl(GamesRepository gamesRepo,
                                   GamesFileRepository gamesFileRepo) {
        this.gamesRepo = gamesRepo;
        this.gamesFileRepo = gamesFileRepo;
    }

    @Override
    public Games saveGameToDB(GameUploadDto dto, GamesVO gvo, User user, Board
board) {
        try {
            // 🎮 Games 엔티티 생성
            Games saveGame = new Games(
                    board, user,
                    dto.getGameName(),
                    dto.getGameDescription(),
                    dto.getTeamName(),
                    dto.getSpecs(),
                    LocalDateTime.now(),
                    dto.getGameVersion(),
                    dto.getGenre(),
                    dto.getPlatform(),
                    gvo.getGameStatus(),  // PENDING_REVIEW
                    gvo.getApprovedAt(),  // null
                    gvo.getIsVisible()    // 0 (숨김)
            );

            Games result = gamesRepo.save(saveGame);
            log.info("✅ 게임 정보 DB 저장 성공 - 게임ID: {}, 게임명: {}",
```

```
                    result.getGameId(), result.getGameName());
            return result;

        } catch (DataIntegrityViolationException e) {
            log.error("❌ DB 제약 조건 위반 - 게임 저장 실패: {}", e.getMessage());
            throw new GameUploadException("중복된 게임이거나 잘못된 데이터입니다.",
e);

        } catch (DataAccessException e) {
            log.error("❌ DB 접근 오류 - 게임 저장 실패: {}", e.getMessage());
            throw new GameUploadException("데이터베이스 저장 중 오류가 발생했습니
다.", e);

        } catch (Exception e) {
            log.error("❌ 예상치 못한 오류 - 게임 저장 실패: {}", e.getMessage());
            throw new GameUploadException("게임 정보 저장 중 오류가 발생했습니다.",
e);
        }
    }

    @Override
    public GamesFile saveGameFileToDB(Games game, GamesFile gamesFile) {
        try {
            // 🔗 Games와 GamesFile 관계 설정
            gamesFile.setGame(game);

            GamesFile result = gamesFileRepo.save(gamesFile);
            log.info("✅ 게임 파일 정보 DB 저장 성공 - 파일ID: {}, 해시: {}",
                    result.getFileId(), result.getGameHash());
            return result;

        } catch (DataIntegrityViolationException e) {
            log.error("❌ DB 제약 조건 위반 - 게임 파일 저장 실패: {}",
e.getMessage());
            throw new GameUploadException("중복된 파일이거나 잘못된 데이터입니다.",
e);

        } catch (Exception e) {
            log.error("❌ 예상치 못한 오류 - 게임 파일 저장 실패: {}",
e.getMessage());
            throw new GameUploadException("파일 정보 저장 중 오류가 발생했습니다.",
e);
        }
    }

    @Override
    public Optional<GamesInfoDto> notApprovedGames() {
        // 🔍 JOIN FETCH로 N+1 문제 해결
        List<Games> gamesWithFiles = gamesRepo.findPendingReviewGamesWithFiles();

        if (gamesWithFiles.isEmpty()) {
            return Optional.empty();
        }
```

```
        // 📦 DTO로 변환
        List<Games> gamesList = gamesWithFiles;
        List<GamesFile> gamesFileList = gamesWithFiles.stream()
                .map(Games::getGamesFile)
                .filter(Objects::nonNull)
                .collect(Collectors.toList());

        return Optional.of(new GamesInfoDto(gamesList, gamesFileList));
    }

    @Override
    public Optional<GamesInfoDto> approvedGames() {
        // 🔍 활성 게임 조회 (JOIN FETCH 사용)
        List<Games> gamesWithFiles = gamesRepo.findActiveGamesWithFiles();

        if (gamesWithFiles.isEmpty()) {
            return Optional.empty();
        }

        List<Games> gamesList = gamesWithFiles;
        List<GamesFile> gamesFileList = gamesWithFiles.stream()
                .map(Games::getGamesFile)
                .filter(Objects::nonNull)
                .collect(Collectors.toList());

        return Optional.of(new GamesInfoDto(gamesList, gamesFileList));
    }
}
```

☁️ CloudUploadServiceImpl.java - 클라우드 업로드 서비스 구현

```
package kr.plusb3b.games.gamehub.application.googleCloud;

import kr.plusb3b.games.gamehub.domain.game.entity.GamesFile;
import kr.plusb3b.games.gamehub.domain.game.exception.GameUploadException;
import kr.plusb3b.games.gamehub.upload.googleCloud.CloudUploadService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import com.google.cloud.storage.*;

import java.security.MessageDigest;
import java.time.LocalDateTime;
import java.util.UUID;
import java.util.List;
import java.util.stream.Collectors;

@Service
@Slf4j
public class CloudUploadServiceImpl implements CloudUploadService {
```

```
    private final Storage storage;
    private final String bucketName;
    private final String tempGamePath;
    private final String activateGamePath;
    private final String deactivateGamePath;

    public CloudUploadServiceImpl(Storage storage,
                                  @Value("${spring.cloud.gcp.storage.bucket}")
String bucketName,
                                  @Value("${app.temp.game.path}") String
tempGamePath,
                                  @Value("${app.activate.game.path}") String
activateGamePath,
                                  @Value("${app.deactivate.game.path}") String
deactivateGamePath) {
        this.storage = storage;
        this.bucketName = bucketName;
        this.tempGamePath = tempGamePath;
        this.activateGamePath = activateGamePath;
        this.deactivateGamePath = deactivateGamePath;
    }

    @Override
    public GamesFile uploadFileToGCP(MultipartFile file) {
        try {
            // 1️⃣ 파일 유효성 검증
            if (file == null || file.isEmpty()) {
                throw new GameUploadException("업로드할 파일이 없습니다.");
            }

            // 2️⃣ 고유 파일명 생성 (타임스탬프 + UUID)
            String originalFileName = file.getOriginalFilename();
            String extension = "";
            if (originalFileName != null && originalFileName.contains(".")) {
                extension =
originalFileName.substring(originalFileName.lastIndexOf("."));
            }

            String uniqueFileName = System.currentTimeMillis() + "_" +
                    UUID.randomUUID().toString().substring(0, 8) + extension;

            // 3️⃣ GCS 업로드 경로 설정
            String cleanTempPath = tempGamePath.startsWith("/") ?
                    tempGamePath.substring(1) : tempGamePath;
            String objectName = cleanTempPath + "/" + uniqueFileName;

            // 4️⃣ 공개 URL 생성
            String gameUrl =
String.format("https://storage.googleapis.com/%s/%s",
                    bucketName, objectName);

            log.info("🚀 GCS 업로드 시작 - 파일: {}, 대상 URL: {}",
```

```
                    uniqueFileName, gameUrl);

            // 5️⃣ GCS에 파일 업로드
            BlobId blobId = BlobId.of(bucketName, objectName);
            BlobInfo blobInfo = BlobInfo.newBuilder(blobId)
                    .setContentType(file.getContentType())
                    .build();

            Blob uploadedBlob = storage.create(blobInfo, file.getBytes());

            if (uploadedBlob != null) {
                log.info("✅ GCS 업로드 성공 - 파일: {}, 크기: {} bytes",
                        uniqueFileName, file.getSize());
            } else {
                throw new GameUploadException("파일 업로드가 완료되지 않았습니
다.");
            }

            // 6️⃣ GamesFile 객체 생성 및 반환
            GamesFile gamesFile = new GamesFile();
            gamesFile.setGameHash(generateFileHash(file));
            gamesFile.setOriginalFilename(originalFileName);
            gamesFile.setGameUrl(gameUrl);
            gamesFile.setFileSize(file.getSize());
            gamesFile.setUploadedAt(LocalDateTime.now());
            gamesFile.setFileStatus(GamesFile.FileStatus.TEMP);

            return gamesFile;

        } catch (Exception e) {
            log.error("❌ GCS 업로드 실패: {}", e.getMessage(), e);
            throw new GameUploadException("파일 업로드 중 오류가 발생했습니다: " +
e.getMessage(), e);
        }
    }

    @Override
    public List<GamesFile> uploadFileToGCP(List<MultipartFile> files) {
        return files.stream()
                .map(this::uploadFileToGCP)
                .collect(Collectors.toList());
    }

    @Override
    public String moveFileToActivateFolder(String gameId, String currentUrl) {
        // TODO: 파일 이동 로직 구현
        // temp-game/파일명 → activate-game/gameId/압축해제된파일들
        return null;
    }

    @Override
    public String moveFileToDeactivateFolder(String gameId, String currentUrl) {
        // TODO: 파일 이동 로직 구현
```

```
        return null;
    }

    /**
     * 파일의 SHA-256 해시값 생성
     * @param file 해시를 생성할 파일
     * @return 20자리 해시 문자열
     */
    private String generateFileHash(MultipartFile file) {
        try {
            byte[] fileBytes = file.getBytes();
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hashBytes = digest.digest(fileBytes);

            StringBuilder hashString = new StringBuilder();
            for (byte b : hashBytes) {
                hashString.append(String.format("%02x", b));
            }

            String fullHash = hashString.toString();
            String shortHash = fullHash.substring(0, Math.min(20,
fullHash.length()));

            log.debug("🔐 파일 해시 생성 완료: {} -> {}",
                    file.getOriginalFilename(), shortHash);
            return shortHash;

        } catch (Exception e) {
            log.warn("⚠️ 파일 해시 생성 실패, 대체 해시 사용: {}", e.getMessage());
            String fallback = (file.getOriginalFilename() +
System.currentTimeMillis())
                    .hashCode() + "";
            return fallback.substring(0, Math.min(20, fallback.length()));
        }
    }
}
```

✅ GameUploadValidatorImpl.java - 파일 검증 서비스 구현

```
@Service
@Slf4j
public class GameUploadValidatorImpl implements GameUploadValidator {

    @Override
    public boolean isIndexHtml(MultipartFile file) {
        try (ZipInputStream zis = new ZipInputStream(file.getInputStream())) {
            ZipEntry entry;

            // 🔍 ZIP 파일 내부를 순회하면서 index.html 찾기
            while ((entry = zis.getNextEntry()) != null) {
                String fileName = entry.getName();
```

```
                // ✅ index.html 파일 찾으면 true 반환
                if ("index.html".equals(fileName) ||
fileName.endsWith("/index.html")) {
                    log.info("✅ index.html 파일 확인됨: {}", fileName);
                    return true;
                }
                zis.closeEntry();
            }

            log.warn("⚠️ index.html 파일이 발견되지 않았습니다.");
            return false;

        } catch (IOException e) {
            log.error("❌ ZIP 파일 구조 검증 실패: {}", e.getMessage());
            return false; // 손상된 ZIP 파일이거나 읽기 실패
        }
    }

    @Override
    public boolean isValidGameZip(MultipartFile file) {
        // TODO: 추가 검증 로직 구현
        // - 파일 크기 체크
        // - 필수 폴더 구조 확인 (Build/, TemplateData/)
        // - 악성 파일 패턴 검사
        return true;
    }
}
```

2.6 Repository

🗄️ GamesRepository.java - 게임 데이터 접근 계층

```
package kr.plusb3b.games.gamehub.domain.game.repository;

import kr.plusb3b.games.gamehub.domain.game.entity.Games;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface GamesRepository extends JpaRepository<Games, Long> {

    /**
     * 승인 대기 중인 게임 조회 (기본 쿼리)
     */
    @Query("SELECT g FROM Games g WHERE g.status = 'PENDING_REVIEW' AND
g.isVisible = 0")
    List<Games> findPendingReviewGames();
```

```
    /**
     * 활성 상태 게임 조회 (기본 쿼리)
     */
    @Query("SELECT g FROM Games g WHERE g.status = 'ACTIVE' AND g.isVisible = 1")
    List<Games> findActiveGames();

    /**
     * 🚀 승인 대기 게임 + 파일 정보 JOIN FETCH (N+1 문제 해결)
     * 한 번의 쿼리로 Games와 GamesFile을 함께 조회
     */
    @Query("SELECT g FROM Games g JOIN FETCH g.gamesFile WHERE g.status =
'PENDING_REVIEW'")
    List<Games> findPendingReviewGamesWithFiles();

    /**
     * 🚀 활성 게임 + 파일 정보 JOIN FETCH
     * 파일 상태도 ACTIVE인 게임만 조회
     */
    @Query("SELECT g FROM Games g JOIN FETCH g.gamesFile gf " +
           "WHERE g.status = 'ACTIVE' AND g.isVisible = 1 AND gf.fileStatus =
'ACTIVE'")
    List<Games> findActiveGamesWithFiles();
}
```

📁 GamesFileRepository.java - 게임 파일 데이터 접근 계층

```
package kr.plusb3b.games.gamehub.domain.game.repository;

import kr.plusb3b.games.gamehub.domain.game.entity.GamesFile;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import java.util.List;

public interface GamesFileRepository extends JpaRepository<GamesFile, Long> {

    /**
     * 파일 상태별 조회
     * @param status 파일 상태 (TEMP, ACTIVE, DEACTIVATED)
     * @return 해당 상태의 파일 목록
     */
    @Query("SELECT gf FROM GamesFile gf WHERE gf.fileStatus = :status")
    List<GamesFile> findByFileStatus(@Param("status") GamesFile.FileStatus
status);

    /**
     * 해시값으로 중복 파일 확인
     * @param gameHash SHA-256 해시값
     * @return 중복 여부
     */
```

```
    boolean existsByGameHash(String gameHash);
}
```

2.7 DTO 및 VO

📤 GameUploadDto.java - 게임 업로드 요청 DTO

```
package kr.plusb3b.games.gamehub.domain.game.dto;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.springframework.web.multipart.MultipartFile;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class GameUploadDto {

    @NotBlank(message = "게임 이름은 필수입니다")
    @Size(max = 200, message = "게임 이름은 200자를 초과할 수 없습니다")
    private String gameName;

    @NotBlank(message = "게임 설명은 필수입니다")
    @Size(max = 1000, message = "게임 설명은 1000자를 초과할 수 없습니다")
    private String gameDescription;

    @NotBlank(message = "팀 이름은 필수입니다")
    @Size(max = 100, message = "팀 이름은 100자를 초과할 수 없습니다")
    private String teamName;

    @Size(max = 50, message = "게임 버전은 50자를 초과할 수 없습니다")
    private String gameVersion;

    @Size(max = 500, message = "권장사양은 500자를 초과할 수 없습니다")
    private String specs;

    @NotBlank(message = "장르 선택은 필수입니다")
    private String genre;

    @NotBlank(message = "플랫폼 선택은 필수입니다")
    private String platform;
```

```
    // 파일 관련 필드
    private MultipartFile gameFile;
}
```

📥 GameUploadResponseDto.java - 게임 업로드 응답 DTO

```
package kr.plusb3b.games.gamehub.domain.game.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDateTime;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class GameUploadResponseDto {
    private Long gameId;
    private String gameName;
    private String teamName;
    private String status;
    private String fileUrl;
    private LocalDateTime uploadedAt;
    private String message;
    private boolean success = true;
}
```

📊 GamesInfoDto.java - 게임 정보 통합 DTO

```
package kr.plusb3b.games.gamehub.domain.game.dto;

import kr.plusb3b.games.gamehub.domain.game.entity.Games;
import kr.plusb3b.games.gamehub.domain.game.entity.GamesFile;
import lombok.Getter;

import java.util.List;

@Getter
public class GamesInfoDto {

    private List<Games> gamesList;
    private List<GamesFile> gamesFiles;

    public GamesInfoDto() {}

    public GamesInfoDto(List<Games> gamesList, List<GamesFile> gamesFiles) {
        this.gamesList = gamesList;
        this.gamesFiles = gamesFiles;
```

```
    }
}
```

❌ ErrorResponseDto.java - 에러 응답 DTO

```
package kr.plusb3b.games.gamehub.domain.game.dto;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class ErrorResponseDto {
    private boolean success = false;
    private String message;
    private String errorCode;
    private LocalDateTime timestamp;

    /**
     * 메시지만 포함한 에러 응답 생성
     */
    public static ErrorResponseDto of(String message) {
        return ErrorResponseDto.builder()
                .success(false)
                .message(message)
                .timestamp(LocalDateTime.now())
                .build();
    }

    /**
     * 메시지와 에러 코드를 포함한 에러 응답 생성
     */
    public static ErrorResponseDto of(String message, String errorCode) {
        return ErrorResponseDto.builder()
                .success(false)
                .message(message)
                .errorCode(errorCode)
                .timestamp(LocalDateTime.now())
                .build();
    }
}
```

🎯 GamesVO.java - 게임 기본값 설정 VO

```
package kr.plusb3b.games.gamehub.domain.game.vo;
```

```java
import kr.plusb3b.games.gamehub.domain.game.entity.Games;
import lombok.Getter;

import java.time.LocalDateTime;

import static
kr.plusb3b.games.gamehub.domain.game.entity.Games.GameStatus.PENDING_REVIEW;

/**
 * Value Object: 불변 객체로 게임의 기본값 설정
 */
@Getter
public final class GamesVO {

    private final LocalDateTime approvedAt = null;  // 초기값 null
    private final Games.GameStatus gameStatus = PENDING_REVIEW;  // 초기 상태
    private final int isVisible = 0;  // 초기에는 비공개
}
```

🎨 Frontend 구현

📝 게임 업로드 폼 (index.html)

```html
<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="_csrf" th:content="${_csrf.token}" />
    <meta name="_csrf_header" th:content="${_csrf.headerName}" />
    <title>게임 정보 입력</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            max-width: 600px;
            margin: 0 auto;
            padding: 20px;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            min-height: 100vh;
        }

        .form-container {
            background: white;
            padding: 30px;
            border-radius: 15px;
            box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
        }

        h1 {
```

```
        text-align: center;
        color: #333;
        margin-bottom: 30px;
        border-bottom: 3px solid #667eea;
        padding-bottom: 10px;
        font-size: 28px;
    }

    .form-group {
        margin-bottom: 20px;
    }

    label {
        display: block;
        margin-bottom: 5px;
        font-weight: bold;
        color: #555;
        font-size: 14px;
    }

    input[type="text"], select {
        width: 100%;
        padding: 12px;
        border: 2px solid #e0e0e0;
        border-radius: 8px;
        font-size: 14px;
        transition: all 0.3s;
        box-sizing: border-box;
    }

    input[type="text"]:focus, select:focus {
        outline: none;
        border-color: #667eea;
        box-shadow: 0 0 0 3px rgba(102, 126, 234, 0.1);
    }

    .file-upload-area {
        position: relative;
        border: 2px dashed #667eea;
        border-radius: 8px;
        padding: 20px;
        text-align: center;
        background: #f8f9ff;
        transition: all 0.3s;
    }

    .file-upload-area:hover {
        background: #eef1ff;
        border-color: #764ba2;
    }

    input[type="file"] {
        width: 100%;
```

```
            padding: 10px;
            cursor: pointer;
        }

        .submit-btn {
            width: 100%;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
            padding: 15px;
            border: none;
            border-radius: 8px;
            cursor: pointer;
            font-size: 16px;
            font-weight: bold;
            margin-top: 20px;
            transition: transform 0.2s;
        }

        .submit-btn:hover:not(:disabled) {
            transform: translateY(-2px);
            box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);
        }

        .submit-btn:disabled {
            opacity: 0.6;
            cursor: not-allowed;
        }

        .loading-spinner {
            display: inline-block;
            width: 16px;
            height: 16px;
            margin-left: 10px;
            border: 2px solid #ffffff;
            border-top-color: transparent;
            border-radius: 50%;
            animation: spin 0.8s linear infinite;
        }

        @keyframes spin {
            to { transform: rotate(360deg); }
        }
    </style>
</head>
<body>
<div class="form-container">
    <h1>🎮 게임 정보 입력</h1>

    <form id="gameUploadForm">
        <div class="form-group">
            <label for="gameName">게임 이름 *</label>
            <input type="text" id="gameName" name="gameName" required
                   placeholder="예: Super Adventure Game">
```

```
        </div>

        <div class="form-group">
            <label for="gameDescription">게임 설명 *</label>
            <input type="text" id="gameDescription" name="gameDescription"
required
                   placeholder="게임에 대한 간단한 설명을 입력하세요">
        </div>

        <div class="form-row" style="display: flex; gap: 15px;">
            <div class="form-group" style="flex: 1;">
                <label for="teamName">팀 이름 *</label>
                <input type="text" id="teamName" name="teamName" required
                       placeholder="예: Dream Team">
            </div>

            <div class="form-group" style="flex: 1;">
                <label for="gameVersion">게임 버전</label>
                <input type="text" id="gameVersion" name="gameVersion"
                       placeholder="예: 1.0.0">
            </div>
        </div>

        <div class="form-group">
            <label for="specs">권장사양</label>
            <input type="text" id="specs" name="specs"
                   placeholder="예: CPU i5, RAM 8GB, GPU GTX 1060">
        </div>

        <div class="form-group">
            <label for="genre">게임 장르 *</label>
            <select name="genre" id="genre" required>
                <option value="">장르를 선택하세요</option>
                <option value="action">🔫 액션</option>
                <option value="adventure">🗺️ 어드벤처</option>
                <option value="rpg">⚔️ RPG</option>
                <option value="simulation">🏗️ 시뮬레이션</option>
                <option value="strategy">♟️ 전략</option>
                <option value="puzzle">🧩 퍼즐</option>
                <option value="racing">🏎️ 레이싱</option>
                <option value="sports">⚽ 스포츠</option>
                <option value="fighting">🥊 격투</option>
                <option value="horror">👻 호러</option>
                <option value="casual">🎯 캐주얼</option>
                <option value="mmo">🌍 MMO</option>
            </select>
        </div>

        <div class="form-group">
            <label>게임 지원 플랫폼 *</label>
            <div style="display: flex; gap: 30px; margin-top: 10px;">
                <label style="font-weight: normal; cursor: pointer;">
                    <input type="radio" name="platform" value="PC" required>
```

```
                    💻 PC
                </label>
                <label style="font-weight: normal; cursor: pointer;">
                    <input type="radio" name="platform" value="Mobile">
                    📱 모바일
                </label>
            </div>
        </div>

        <div class="form-group">
            <label for="gameFile">WebGL 게임 파일 *</label>
            <div class="file-upload-area">
                <input type="file" id="gameFile" name="gameFile"
                       accept=".zip,.rar,.7z" required>
                <div style="margin-top: 10px; color: #666; font-size: 12px;">
                    📦 ZIP, RAR, 7Z 형식 (최대 100MB)
                </div>
                <div id="fileInfo" style="margin-top: 10px; color: #667eea; font-
weight: bold;"></div>
            </div>
        </div>

        <button type="submit" class="submit-btn" id="submitBtn">
            게임 업로드
        </button>
    </form>
</div>

<script>
    // 파일 선택 시 파일 정보 표시
    document.getElementById('gameFile').addEventListener('change', function(e) {
        const file = e.target.files[0];
        const fileInfo = document.getElementById('fileInfo');

        if (file) {
            const sizeMB = (file.size / (1024 * 1024)).toFixed(2);
            fileInfo.textContent = `선택된 파일: ${file.name} (${sizeMB} MB)`;

            // 파일 크기 검증
            if (file.size > 100 * 1024 * 1024) {
                fileInfo.style.color = '#f44336';
                fileInfo.textContent += ' - ⚠️ 파일 크기 초과!';
            }
        }
    });

    // 폼 제출 처리
    document.getElementById('gameUploadForm').addEventListener('submit', async
function(e) {
        e.preventDefault();

        const submitBtn = document.getElementById('submitBtn');
        const originalText = submitBtn.innerHTML;
```

```
        // 파일 크기 검증
        const fileInput = document.getElementById('gameFile');
        const file = fileInput.files[0];

        if (file && file.size > 100 * 1024 * 1024) {
            alert('❌ 파일 크기는 100MB 이하로 제한됩니다.');
            return;
        }

        // 필수 필드 검증
        const requiredFields = ['gameName', 'gameDescription', 'teamName',
'genre'];
        for (let fieldId of requiredFields) {
            const field = document.getElementById(fieldId);
            if (!field.value.trim()) {
                alert(`❌ ${field.previousElementSibling.textContent.replace('
*', '')}을(를) 입력해주세요.`);
                field.focus();
                return;
            }
        }

        // 플랫폼 선택 검증
        const platform =
document.querySelector('input[name="platform"]:checked');
        if (!platform) {
            alert('❌ 게임 지원 플랫폼을 선택해주세요.');
            return;
        }

        // 파일 선택 검증
        if (!file) {
            alert('❌ WebGL 게임 파일을 선택해주세요.');
            return;
        }

        // 로딩 상태 표시
        submitBtn.disabled = true;
        submitBtn.innerHTML = '업로드 중...<span class="loading-spinner"></span>';

        try {
            // CSRF 토큰 가져오기
            const csrfToken =
document.querySelector("meta[name='_csrf']").getAttribute("content");
            const csrfHeader =
document.querySelector("meta[name='_csrf_header']").getAttribute("content");

            // FormData 생성
            const formData = new FormData();
            formData.append('gameName',
document.getElementById('gameName').value.trim());
            formData.append('gameDescription',
```

```
document.getElementById('gameDescription').value.trim());
            formData.append('teamName',
document.getElementById('teamName').value.trim());
            formData.append('gameVersion',
document.getElementById('gameVersion').value.trim() || '1.0.0');
            formData.append('specs',
document.getElementById('specs').value.trim());
            formData.append('genre', document.getElementById('genre').value);
            formData.append('platform', platform.value);
            formData.append('gameFile', file);

            // API 요청
            const response = await fetch('/game-hub/api/v1/games/upload', {
                method: 'POST',
                headers: {
                    [csrfHeader]: csrfToken
                },
                body: formData
            });

            if (response.ok) {
                const result = await response.json();

                // 성공 메시지
                alert(`✅ 게임 업로드 성공!\n\n` +
                      `게임명: ${result.gameName}\n` +
                      `팀명: ${result.teamName}\n` +
                      `상태: ${result.status}\n\n` +
                      `${result.message}`);

                // 제작자 페이지로 이동
                window.location.href = '/game-hub/prod';

            } else {
                // 에러 처리
                let errorMessage = '업로드에 실패했습니다.';

                try {
                    const errorData = await response.json();
                    if (errorData.message) {
                        errorMessage = errorData.message;
                    }
                } catch (parseError) {
                    errorMessage = `업로드 실패 (HTTP ${response.status})`;
                }

                // 상태 코드별 안내
                let detailedMessage = '';
                switch (response.status) {
                    case 400:
                        detailedMessage = '\n\n💡 index.html 파일이 포함되어 있는
지 확인해주세요.';
                        break;
```

```
                    case 401:
                        detailedMessage = '\n\n💡 로그인이 필요합니다.';
                        break;
                    case 413:
                        detailedMessage = '\n\n💡 파일 크기를 100MB 이하로 줄여주세
요.';
                        break;
                    case 500:
                        detailedMessage = '\n\n💡 잠시 후 다시 시도해주세요.';
                        break;
                }

                alert(`❌ ${errorMessage}${detailedMessage}`);
            }

        } catch (error) {
            console.error('업로드 요청 중 오류:', error);
            alert('❌ 네트워크 오류가 발생했습니다.\n인터넷 연결을 확인해주세요.');

        } finally {
            // 버튼 복원
            submitBtn.disabled = false;
            submitBtn.innerHTML = originalText;
        }
    });
</script>
</body>
</html>
```

🎮 제작자 대시보드 (index-prod.html)

```
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <title>제작자 대시보드</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            min-height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
            margin: 0;
        }

        .dashboard-container {
            background: white;
            padding: 40px;
            border-radius: 15px;
            box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
```

```
            text-align: center;
        }

        h2 {
            color: #333;
            margin-bottom: 30px;
            font-size: 32px;
        }

        #game-upload {
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
            border: none;
            padding: 15px 40px;
            font-size: 18px;
            border-radius: 8px;
            cursor: pointer;
            transition: transform 0.2s, box-shadow 0.2s;
        }

        #game-upload:hover {
            transform: translateY(-2px);
            box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);
        }
    </style>
</head>
<body>
    <div class="dashboard-container">
        <h2>🎮 제작자 대시보드</h2>
        <button type="button" id="game-upload" name="game-upload">
            게임 업로드 하기
        </button>
    </div>

    <script>
        document.getElementById("game-upload").addEventListener("click",
function() {
            window.location.href = "/game-hub/prod/upload";
        });
    </script>
</body>
</html>
```

🌐 Backend REST API Controller

🎯 RestGameUploadController.java - 게임 업로드 API (계속)

```
    @PostMapping("/upload")
    @Transactional
```

```
    public ResponseEntity<?> uploadGame(@Valid @ModelAttribute GameUploadDto
gameUploadDto,
                                        HttpServletRequest request) {

        log.info("🎮 게임 업로드 요청 시작 - 게임명: {}, 팀명: {}",
                gameUploadDto.getGameName(), gameUploadDto.getTeamName());

        try {
            // 1️⃣ 로그인 사용자 검증
            User user = access.getAuthenticatedUser(request);
            if (user == null) {
                log.warn("⚠️ 미인증 사용자의 업로드 시도");
                return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                        .body(ErrorResponseDto.of("로그인 후 이용할 수 있습니다.",
"AUTH_REQUIRED"));
            }

            // 2️⃣ 파일 존재 여부 검증
            if (gameUploadDto.getGameFile() == null ||
gameUploadDto.getGameFile().isEmpty()) {
                log.warn("⚠️ 빈 파일 업로드 시도 - 사용자: {}",
user.getUserAuth().getAuthUserId());
                return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                        .body(ErrorResponseDto.of("업로드할 파일을 선택해주세요.",
"FILE_REQUIRED"));
            }

            // 3️⃣ HTML 파일 존재 여부 검증
            boolean hasIndexHtml =
gameUploadValidator.isIndexHtml(gameUploadDto.getGameFile());
            if (!hasIndexHtml) {
                log.warn("⚠️ index.html 파일 누락 - 사용자: {}, 파일: {}",
                        user.getUserAuth().getAuthUserId(),
                        gameUploadDto.getGameFile().getOriginalFilename());
                return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                        .body(ErrorResponseDto.of(
                            "해당 파일에 index.html 파일이 존재하지 않습니다.",
                            "INVALID_WEBGL_STRUCTURE"));
            }

            // 4️⃣ 게시판 조회
            Board board = boardRepo.findById("gameBoard")
                    .orElseThrow(() -> new IllegalArgumentException("게임 게시판을
찾을 수 없습니다."));

            // 5️⃣ GCP에 파일 업로드
            log.info("☁️ GCP 업로드 시작 - 파일: {}",
                    gameUploadDto.getGameFile().getOriginalFilename());
            GamesFile uploadResult =
gcpUploadService.uploadFileToGCP(gameUploadDto.getGameFile());

            // 6️⃣ 게임 메타데이터 저장
            log.info("💾 게임 메타데이터 저장 시작 - 게임명: {}",
```

```
gameUploadDto.getGameName());
            Games savedGame = gameMetadataService.saveGameToDB(
                    gameUploadDto, new GamesVO(), user, board);

            // 7️⃣ 게임 파일 정보 저장
            log.info("📁 게임 파일 정보 저장 시작 - 게임ID: {}",
savedGame.getGameId());
            GamesFile savedGamesFile = gameMetadataService.saveGameFileToDB(
                    savedGame, uploadResult);

            // 8️⃣ 성공 응답 생성
            GameUploadResponseDto responseDto = GameUploadResponseDto.builder()
                    .gameId(savedGame.getGameId())
                    .gameName(savedGame.getGameName())
                    .teamName(savedGame.getTeamName())
                    .status(savedGame.getStatus().name())
                    .fileUrl(savedGamesFile.getGameUrl())
                    .uploadedAt(savedGamesFile.getUploadedAt())
                    .message("성공적으로 업로드되었습니다. 관리자의 승인을 기다리세
요.")
                    .success(true)
                    .build();

            log.info("✅ 게임 업로드 완료 - 게임ID: {}, 파일ID: {}",
                    savedGame.getGameId(), savedGamesFile.getFileId());

            return ResponseEntity.status(HttpStatus.CREATED).body(responseDto);

        } catch (GameUploadException e) {
            log.error("❌ 게임 업로드 실패 - 사용자 오류: {}", e.getMessage());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                    .body(ErrorResponseDto.of(e.getMessage(), "UPLOAD_ERROR"));

        } catch (IllegalArgumentException e) {
            log.error("❌ 게임 업로드 실패 - 잘못된 인수: {}", e.getMessage());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST)
                    .body(ErrorResponseDto.of("잘못된 요청입니다: " +
e.getMessage(), "INVALID_REQUEST"));

        } catch (Exception e) {
            log.error("❌ 게임 업로드 중 예상치 못한 오류 발생", e);
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                    .body(ErrorResponseDto.of(
                        "서버 오류가 발생했습니다. 잠시 후 다시 시도해주세요.",
                        "INTERNAL_ERROR"));
        }
    }
}
```

🎮 UserProdController.java - 제작자 페이지 컨트롤러

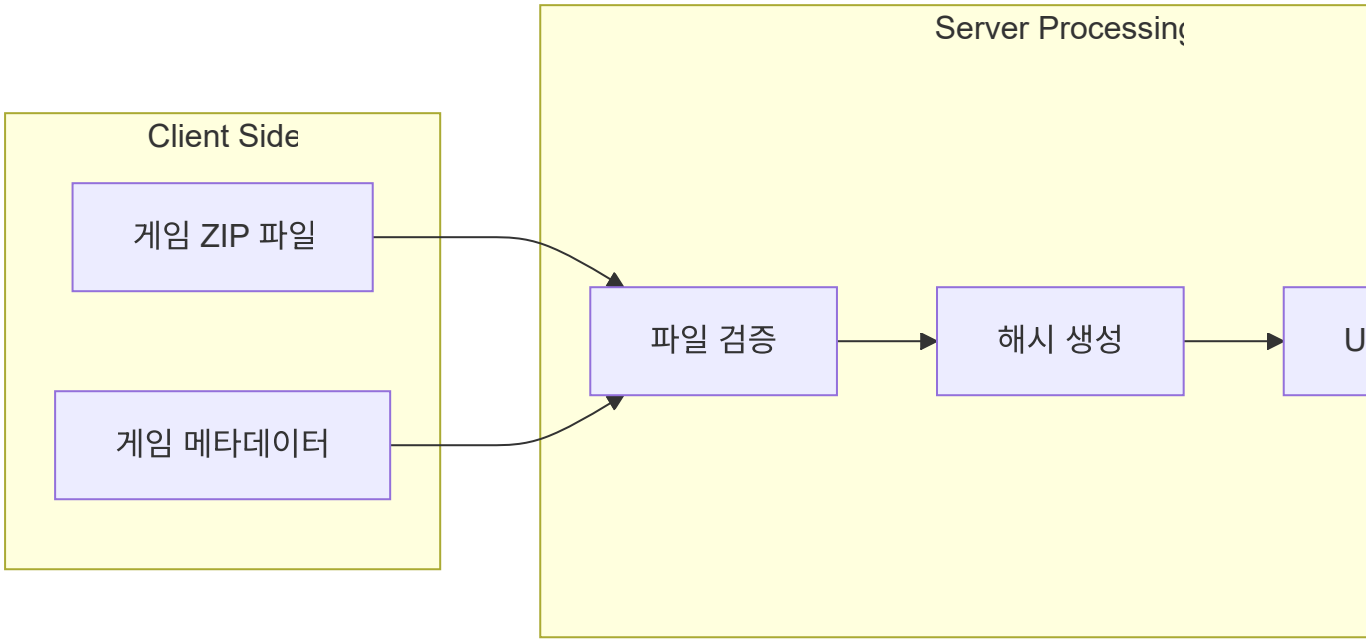
```java
package kr.plusb3b.games.gamehub.api.controller.user;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/game-hub/prod")
public class UserProdController {

    /**
     * 업로드 페이지로 이동
     * @return 업로드 페이지 뷰
     */
    @GetMapping("/upload")
    public String viewUploadPage(){
        return "game/upload/index";  // templates/game/upload/index.html
    }

    /**
     * 제작자 대시보드로 이동
     * @return 대시보드 페이지 뷰
     */
    @GetMapping("")
    public String viewProducerDashboard(){
        return "producer/dashboard/index";  //
templates/producer/dashboard/index.html
    }
}
```

🔄 전체 시스템 플로우

📊 업로드 프로세스 시퀀스 다이어그램


사용자(개발자)
Frontend
Controlle
게임 정보 입력 + 파일 선택
클라이언트 검증
POST /api/v1/games/upload
사용자 인증
alt
400 Bad Request
에러 메시지 표시

🎯 데이터 흐름 다이어그램



🚦 예외 처리 및 검증

❌ GameUploadException.java - 커스텀 예외 클래스

```
package kr.plusb3b.games.gamehub.domain.game.exception;

public class GameUploadException extends RuntimeException {

    /**
     * 메시지만 받는 생성자
     */
    public GameUploadException(String message) {
        super(message);
    }

    /**
     * 메시지와 원인 예외를 받는 생성자
     */
    public GameUploadException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

🛡️ 검증 체크리스트

검증 항목	구현 위치	검증 내용
사용자 인증	Controller	로그인 여부 확인
파일 존재	Controller	파일이 비어있지 않은지 확인
파일 크기	Frontend/Properties	100MB 제한
파일 형식	Frontend	ZIP/RAR/7Z 확장자
WebGL 구조	Validator	index.html 존재 확인
필수 필드	DTO/Frontend	@NotBlank 검증
문자열 길이	DTO	@Size 검증
파일 중복	Service	SHA-256 해시 비교

📈 성능 최적화 전략

🚀 구현된 최적화

1. **N+1 쿼리 문제 해결**

```
// JOIN FETCH를 사용하여 한 번의 쿼리로 연관 데이터 조회
@Query("SELECT g FROM Games g JOIN FETCH g.gamesFile WHERE g.status =
```

```
'PENDING_REVIEW'")
List<Games> findPendingReviewGamesWithFiles();
```

2. **지연 로딩 설정**

```
@OneToOne(mappedBy = "game", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
private GamesFile gamesFile;
```

3. **트랜잭션 관리**

```
@Transactional  // 전체 업로드 과정을 하나의 트랜잭션으로 처리
public ResponseEntity<?> uploadGame(...) { }
```

4. **파일 스트림 자동 해제**

```
try (ZipInputStream zis = new ZipInputStream(file.getInputStream())) {
    // try-with-resources로 자원 자동 해제
}
```

🔐 보안 고려사항

🛡️ 구현된 보안 기능

1. **파일 업로드 보안**
   - 파일 크기 제한 (100MB)
   - 허용된 확장자만 업로드 (ZIP, RAR, 7Z)
   - 파일 내용 검증 (index.html 필수)
2. **인증 및 권한**
   - 사용자 로그인 검증
   - CSRF 토큰 검증
   - 역할 기반 접근 제어
3. **데이터 무결성**
   - SHA-256 해시를 통한 파일 무결성 검증
   - 중복 파일 방지
   - 트랜잭션을 통한 데이터 일관성 보장
4. **입력 검증**
   - Bean Validation (@NotBlank, @Size)
   - 클라이언트/서버 양방향 검증
   - SQL Injection 방지 (JPA 사용)

📊 모니터링 및 로깅

📝 로깅 전략

```
// 단계별 상세 로깅
log.info("🎮 게임 업로드 요청 시작 - 게임명: {}, 팀명: {}", ...);
log.info("☁️ GCP 업로드 시작 - 파일: {}", ...);
log.info("💾 게임 메타데이터 저장 시작 - 게임명: {}", ...);
log.info("✅ 게임 업로드 완료 - 게임ID: {}, 파일ID: {}", ...);

// 에러 레벨별 로깅
log.warn("⚠️ 미인증 사용자의 업로드 시도");
log.error("❌ 게임 업로드 실패 - 사용자 오류: {}", e.getMessage());
```
