



README

1 Description

- 1) 게임 소개
- 2) 개발 배경
- 3) 기능 명세서
- 4) 활용 언어 및 주요 기술
- 5) 프로젝트 아키텍처
- 6) ERD

2 FRONT-END



설명

3 WEB SERVER



주요 기술 및 기능

4 BUSINESS SERVER



HOW TO

Netty

5 CHATTING SERVER

Boost.Asio 라이브러리

비동기 데이터 처리

MessagePack

데이터 관리

서버 동작 시나리오

6 LIVE SERVER



HOW TO

Boost.Asio 라이브러리

MessagePack

멀티쓰레드

UDP / TCP

서버 동작 시나리오

7 UNITY

8 API 명세서

9 Convention



Branch종류



커밋 브랜치



팀 커밋 컨벤션

10 팀원 별 담당 역할 및 업무

1 Description

1) 게임 소개

게임 장르: 하이퍼 캐주얼 레이싱

플랫폼: 모바일 / PC

플레이어 수: 최대 6인 멀티플레이

게임 개요:

빠른 속도와 간편한 조작으로 누구나 쉽게 즐길 수 있는 하이퍼 캐주얼 레이싱 게임입니다. 최대 6명의 플레이어가 동시에 참여할 수 있어, 친구들과 함께하거나 전 세계의 유저들과 경쟁하며 스릴 넘치는 레이싱 경험을 할 수 있습니다.

🏎️ 스피드 모드:

- 목표: 다양한 맵을 주행하며 누구보다 빠르게 결승선에 도착하는 것이 목표입니다.
- 특징: 각 맵은 독특한 디자인과 장애물을 갖추고 있어, 전략적인 주행과 빠른 반사 신경이 필요합니다.
- 경쟁 요소: 실시간으로 다른 플레이어들과 속도를 겨루며, 최종 결승선에 가장 먼저 도착하는 플레이어가 승리합니다.

⚽ 축구 모드:

- 목표: 플레이어는 자신의 카트를 운전하며 공을 상대방의 골대에 넣는 것을 목표로 합니다.
- 특징: 축구와 레이싱의 결합으로, 단순한 주행뿐만 아니라 전략적인 플레이와 팀워크가 중요합니다.
- 경쟁 요소: 제한된 시간 동안 누가 더 많은 골을 넣는가를 겨루며, 최종적으로 더 많은 골을 넣은 플레이어가 승리합니다.

게임 특징:

- 간편한 조작: 직관적인 인터페이스와 간편한 조작법으로 누구나 쉽게 즐길 수 있습니다.
- 다양한 맵: 스피드 모드에서 다양한 테마와 난이도를 가진 맵들이 제공되어, 플레이할 때마다 새로운 재미를 느낄 수 있습니다.

- 멀티플레이: 최대 6명의 플레이어가 동시에 경쟁할 수 있어, 친구들과 함께 즐기거나 전 세계의 유저들과 실력을 겨룰 수 있습니다.
- 빠른 경기 진행: 짧고 긴박한 게임 플레이로 언제 어디서나 간편하게 즐길 수 있습니다.

2) 개발 배경

팀원들과 함께 스트레스에 대해 이야기 하다가 가장 많은 공감을 얻었던 해소 방법이 드라이브였습니다. 자동차를 운전하며 다양한 풍경을 감상하고, 뻥 뚫린 도로를 달릴 때면 자연스레 쌓였던 답답함이 날아갑니다. 하지만, 현실의 도로에서는 도로 위의 많은 차량과 교통법규로 제약이 있습니다.

이러한 점을 바탕으로 게임이라는 가상 환경에서 자유롭게 드라이빙 하는 컨텐츠를 구상했습니다. 여기에 단순한 드라이빙이 아닌 경쟁의 재미를 더하고자 했고, 아이템을 사용해 다양한 변수를 만들어 내는 방법을 선택하게 되었습니다.

3) 기능 명세서

* 프로젝트 진행을 위해 분석된 요구사항과 기능 정의를 아래 표에 작성합니다.

- 공통

요구사항	기능 정의
회원 관리	회원 이메일 주소(수정 불가)와 닉네임을 등록/수정/삭제
로그인/로그아웃	회원의 웹사이트 이용을 위한 로그인/로그아웃 기능을 구현

- 인게임

요구사항	기능 정의
게임 방 생성	게임 플레이를 위한 방을 생성
게임 방 참가	생성된 게임 방에 참가
게임 방 나가기	로비로 이동 및 방장 권한 인가 및 방에 남은 인원이 없을 경우 방 제거
채팅	로비 및 게임 방 내에서 채팅 가능
캐릭터 선택	레이싱에서 사용할 캐릭터를 선택

요구사항	기능 정의
게임 시작	게임을 시작
카트 운행	레이싱 중 카트의 운행을 조작
게임 종료	최초로 결승점을 통과할 경우 현재 게임의 남은 시간을 10초로 설정하여 게임 종료 예고
점수 종합	각 유저들의 결승점 통과 기록을 산출하여 서버로 전송 및 기록 저장
로비로 이동	게임 종료 이후 모든 유저들을 로비로 이동
부스터 획득	레이싱 중 부스터를 획득
부스터 이용	획득한 부스터를 사용
게임 종료	최초로 결승점을 통과할 경우 현재 게임의 남은 시간을 10초로 설정하여 게임 종료 예고
점수 종합 및 업데이트	각 유저들의 결승점 통과 기록을 산출하여 랭킹에 반영
로비로 이동	게임 종료 이후 모든 유저들을 로비로 이동

- 웹

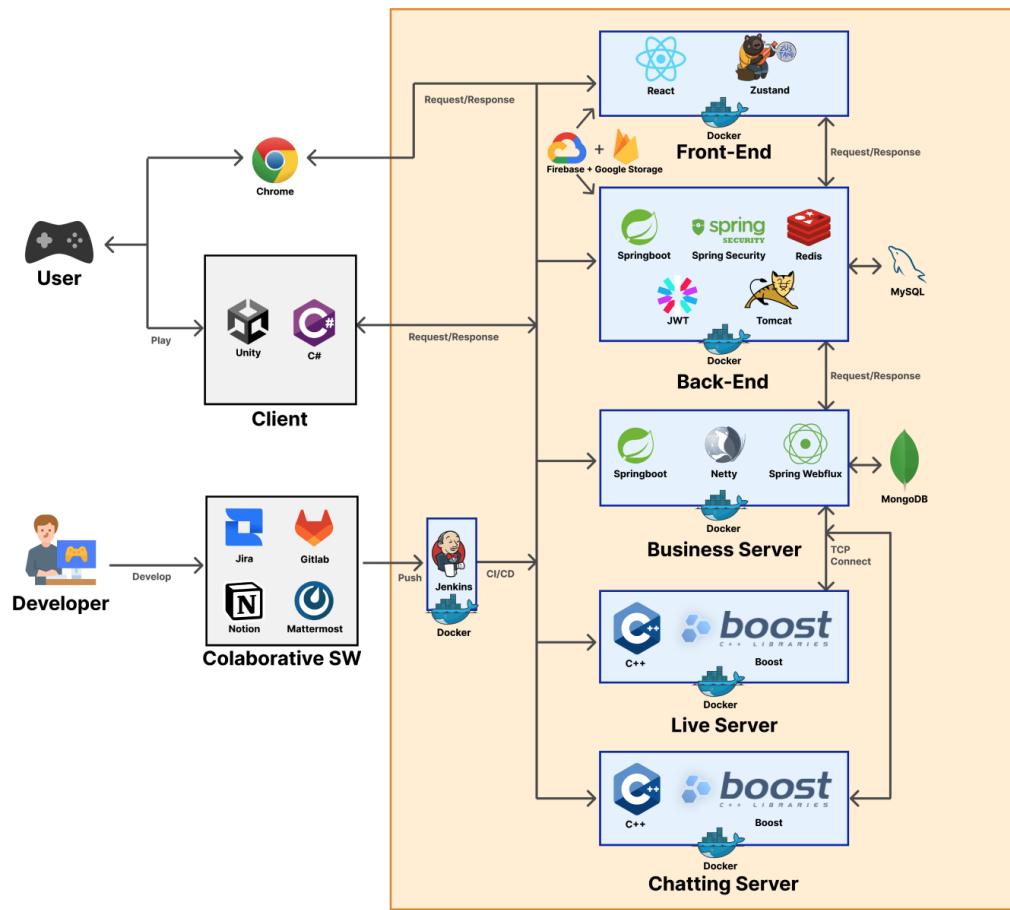
요구사항	기능 정의
랭킹 확인	전체 유저 혹은 특정 유저의 랭킹을 맵 별로 조회
프로필 변경	닉네임과 프로필 이미지를 변경
공지 사항 확인	공지 사항 및 업데이트 사항 조회
커뮤니케이션	자유게시판을 활용한 커뮤니케이션 가능 (글 작성 및 댓글 작성 가능)
게임 다운로드	다운로드 버튼을 통한 게임파일 다운로드

4) 활용 언어 및 주요 기술

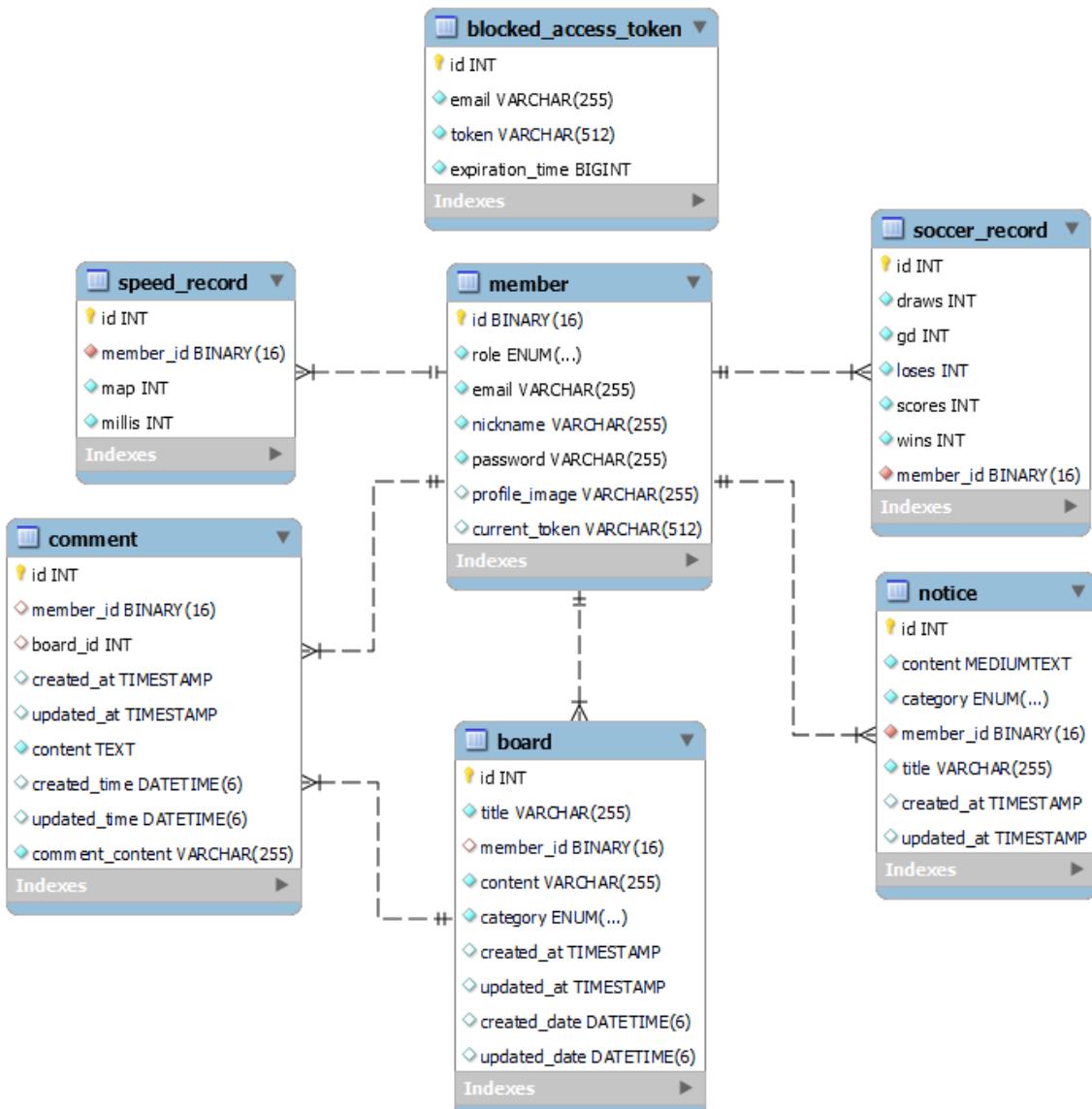
- 활용 언어
 - C++

- C#
- Java
- JavaScript(Typescript)
- 주요 기술
 - Unity
 - Spring Boot
 - React
 - Netty
 - Message Pack
 - Spring Security
 - Docker
 - Jenkins
 - Jwt
 - Redis

5) 프로젝트 아키텍처



6) ERD



2 FRONT-END

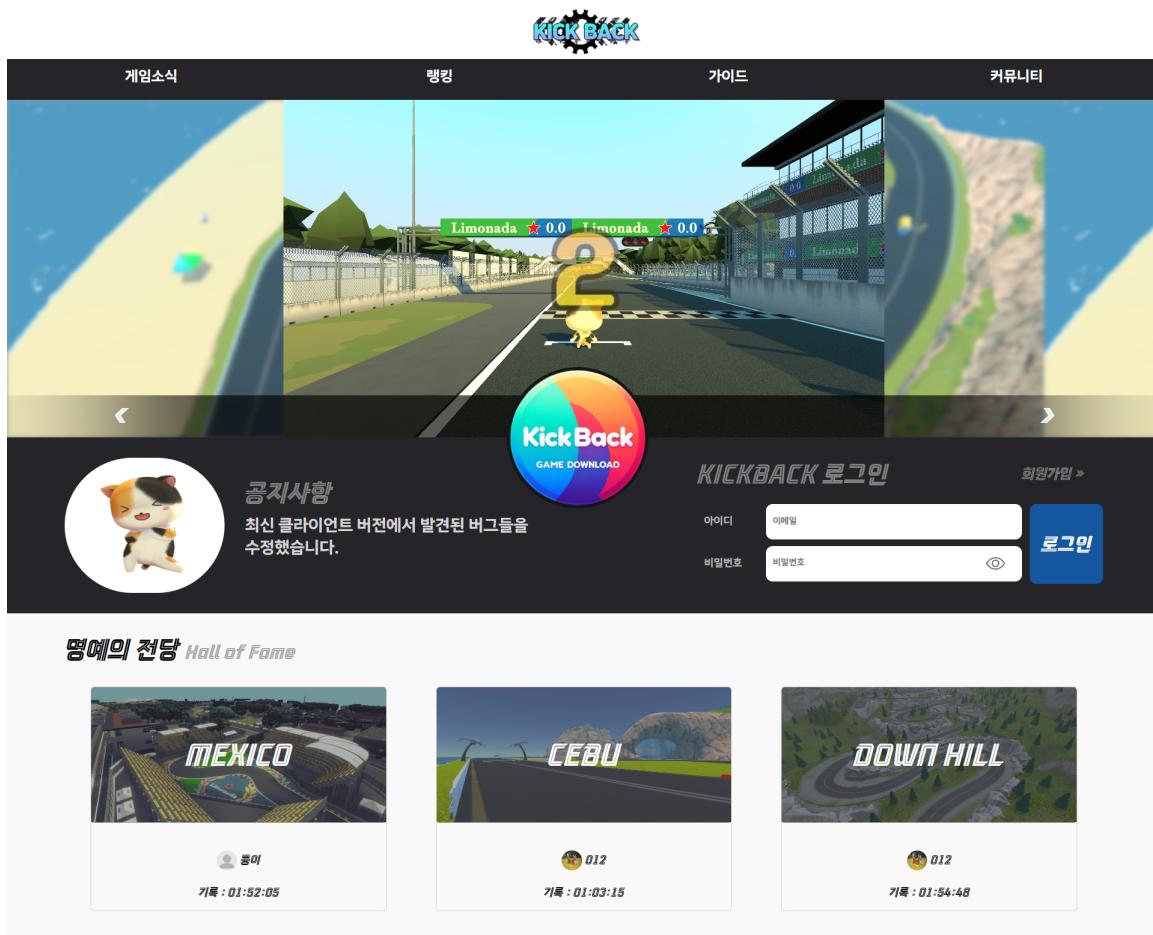
FRONT-END 파일 구조

설명

- 웹사이트에서 사용자는 로그인, 회원가입을 할 수 있습니다.
- 메인페이지 중앙의 다운로드 버튼을 통해 게임파일을 다운로드 할 수 있습니다.
- 마이페이지에서 프로필변경(닉네임, 프로필 이미지)을 할 수 있습니다.
- 랭킹페이지에서 전체유저의 모드별, 맵별 기록 및 랭킹을 확인할 수 있으며 특정유저의 랭킹 및 기록을 조회할 수 있습니다.

- 공지사항 페이지에서 공지사항과 게임 업데이트 사항을 확인할 수 있습니다.
- 자유게시판에서 소통할 수 있는 할수 있는 커뮤니티 서비스를 제공합니다.

▼ 메인 페이지 이미지

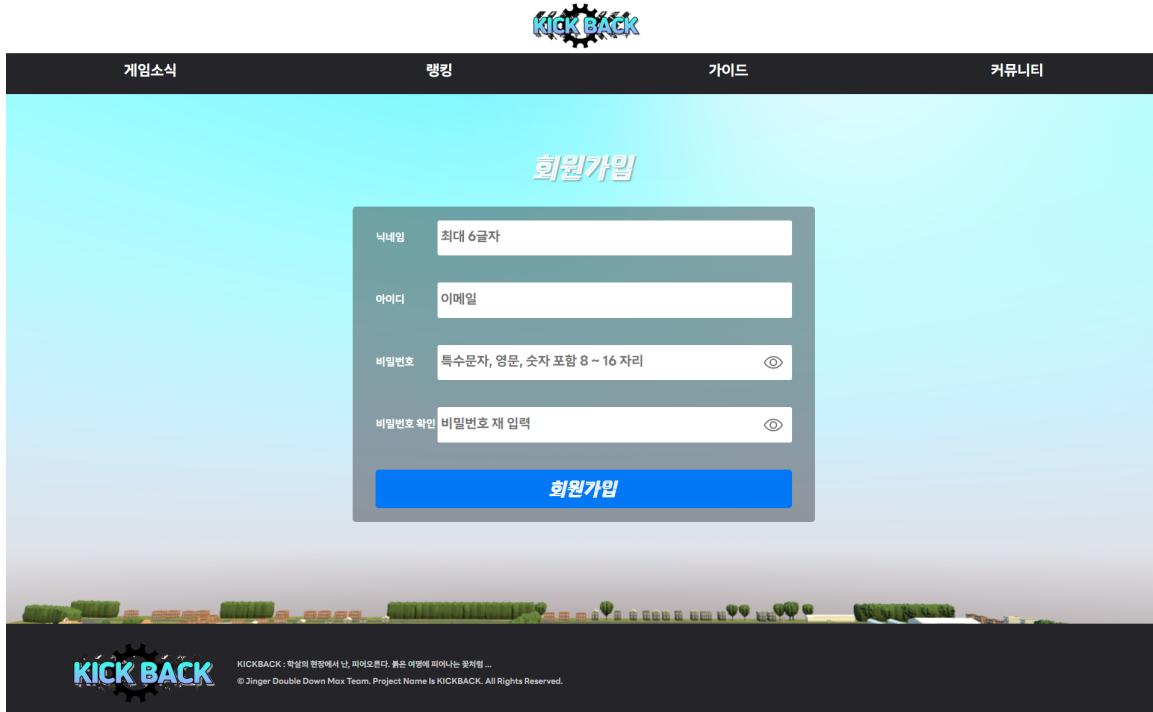


모드 소개 Mode Introduction



KICKBACK : 학살의 현장에서 난, 피어오른다. 뿐은 아영에 피어나는 꽃처럼 ...
© Jinger Double Down Max Team, Project Name is KICKBACK. All Rights Reserved.

▼ 회원가입 페이지 이미지



▼ 랭킹 페이지 이미지

등수	닉네임	기록
		동이 01:52:05
		012 02:02:22
		메룡 03:35:06

3 WEB SERVER

WEB SERVER 파일 구조

주요 기술 및 기능

- **Spring Security와 JWT 토큰을 사용하여 인증 및 인가 시스템을 구현**
 - Redis를 활용하여 refreshToken을 저장하고 빠른 조회를 가능하게 구현
 - 클라이언트의 요청시 서버에서 Jwt Token을 필터링하여 인증/인가를 처리 및 중복된 로그인 상태를 방지
- **Jasypt를 통해 환경 변수를 암호화하여 안전하게 관리**
- **JPA 및 @Query 어노테이션 활용해 유연하게 데이터베이스에 접근**
- **비동기 통신 활용 (async function)**
 - Spring의 `@Async` 어노테이션을 사용하여 게임 기록을 생성 및 업데이트하는 요청을 별도에 스레드 풀에서 비동기적으로 처리
 - 메인 스레드 부담을 감소하여 자원을 효율적으로 활용하고 및 응답시간을 단축
- **클라우드 스토리지 활용**
 - 클라우드 스토리지(Firebase, 구글 클라우드)에 게임 파일, 프로필 이미지를 저장하여, 서버 저장소의 부담 감소

Spring Security 와 Jwt 토큰을 사용한 인증/인가

▼ Security Configuration 코드

```
# Security Configuration

@EnableConfigurationProperties(JwtProps.class)
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtTokenProvider jwtTokenProvider;
```

```
private final ObjectMapper objectMapper;
private final BlockedAccessTokenService blockedAccessTokenService;

// 비밀번호 인코딩 방식을 BCrypt로 설정
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}

// HTTP 요청에 대한 보안 구성 정의
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        // CORS 설정을 활성화합니다.
        .cors(cors -> cors.configurationSource(corsConfigurationSource()))
        // 모든 요청에 대해 HTTPS를 요구합니다.
        .requiresChannel(channel -> channel.anyRequest().requiresSecure())
        // 기본 인증을 비활성화합니다.
        .httpBasic(AbstractHttpConfigurer::disable)
        // Clickjacking 공격 방지를 위해 사용되는
        // X-Frame-Options 헤더를 비활성화합니다.
        .headers(header -> header.frameOptions(HeadersConfigurer.FrameOptionsConfig::disable))
        // 모든 요청에 대해 접근을 허용합니다.
        .authorizeHttpRequests(auth -> auth.anyRequest().permitAll())
        // 폼 기반 로그인을 비활성화합니다.
        .formLogin(AbstractHttpConfigurer::disable)
        // 로그아웃 처리를 설정합니다.
        .logout(authz -> authz.logoutUrl("/api/v1/member/logout"))
        .deleteCookies("JSESSIONID")
```

```

        .clearAuthentication(true))
            // JWT 필터를 UsernamePasswordAuthenticationFilter 앞에 추가합니다.
            .addFilterBefore(jwtSecurityFilter(),
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    // JWT 보안 필터를 빈으로 정의합니다.
@Bean
public JwtSecurityFilter jwtSecurityFilter() {
    return new JwtSecurityFilter(jwtTokenProvide
r, objectMapper, blockedAccessTokenService);
}

// CORS 필터를 등록합니다. 외부 도메인에서의 API 요청을 허
용하기 위한 구성입니다.
@Bean
public FilterRegistrationBean<CorsFilter> corsFil
terRegistrationBean() {
    CorsConfigurationSource source = corsConfigur
ationSource();
    FilterRegistrationBean<CorsFilter> filterBean
= new FilterRegistrationBean<>(new CorsFilter(sourc
e));
    // 필터 체인에서의 순서 설정, 숫자가 낮을수록 먼저 실
행.
    filterBean.setOrder(0);
    return filterBean;
}

// CORS 설정을 위한 Bean을 정의합니다. CORS는 외부 도메인
에서 리소스를 안전하게 요청할 수 있게 해주는 메커니즘입니다.
@Bean
public CorsConfigurationSource corsConfigurations
ource() {
    CorsConfiguration config = getCorsConfigurati
on(6 * 60 * 60 * 1000L);

```

```
        UrlBasedCorsConfigurationSource source = new
        UrlBasedCorsConfigurationSource();
        // 애플리케이션의 모든 경로 ("/**")에 대해 CORS 구성
        적용
        source.registerCorsConfiguration("/**", config);
        return source;
    }

    // CORS 구성은 생성합니다. 여기서는 모든 출처, 헤더, 메소드
    를 허용하며, 사전 요청의 Max Age를 설정합니다.
    private CorsConfiguration getCorsConfiguration(long maxAge) {
        CorsConfiguration configuration = new CorsCon
        figuration();

        List<String> allowedOrigins = Arrays.asList(
            "http://localhost:3000",
            "https://k10c209.p.ssafy.io",
            "http://192.168.100.146:3000"
        );

        // 허용된 출처 설정
        configuration.setAllowedOrigins(allowedOrigin
        s);
        // 허용된 HTTP 메서드 설정
        configuration.setAllowedMethods(Arrays.asList
        ("GET", "POST", "PATCH", "DELETE", "PUT"));
        // 자격 증명 허용 설정
        configuration.setAllowCredentials(true);
        // 허용된 헤더 설정
        configuration.setAllowedHeaders(Arrays.asList
        ("Content-Type", "Authorization")));
        // 사전 요청의 최대 캐시 시간 설정
        configuration.setMaxAge(maxAge);

        // 노출할 헤더 설정
        configuration.setExposedHeaders(Arrays.asList
```

```
        ("Authorization", "refreshToken", "accessToken"));  
  
        return configuration;  
    }  
}
```

▼ JwtFilter 코드

사용자의 요청에 대해 Jwt 토큰을 필터링합니다.

```
@Slf4j  
@RequiredArgsConstructor  
public class JwtSecurityFilter extends OncePerRequest  
Filter {  
  
    private final JwtTokenProvider jwtTokenProvider;  
    private final ObjectMapper objectMapper;  
    private final BlockedAccessTokenService blockedAccess  
    tokenService;  
  
    private static final String BEARER_PREFIX = "Bear  
er ";  
  
    // 특정 url들에 대해 필터링을 다르게 처리하도록 하기 위  
    // 한 선언  
    private static final Set<String> EXACT_PATHS = ne  
    w HashSet<>();  
    private static final String RANKING_API_PREFIX =  
    "/api/v1/ranking";  
    private static final String NOTICE_API_PREFIX =  
    "/api/v1/notice";  
    private static final String BOARD_API_PREFIX = "  
api/v1/board";  
  
    static {  
        EXACT_PATHS.add("/api/v1/member/login");  
        EXACT_PATHS.add("/api/v1/member/reissue/acces  
sToken");  
    }
```

```
        EXACT_PATHS.add("/api/v1/member/signup");
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {

        String path = request.getRequestURI();
        String method = request.getMethod();

        // 다음 경로에서는 필터를 실행하지 않음
        if (EXACT_PATHS.contains(path) || path.startsWith(RANKING_API_PREFIX)) {
            filterChain.doFilter(request, response);
            return;
        }
        else if (path.startsWith(NOTICE_API_PREFIX) || path.startsWith(BOARD_API_PREFIX)){
            if (method.equals("GET")) {
                filterChain.doFilter(request, response);
                return;
            }
        }
    }

    String accessToken = resolveTokenFromCookie(request);
    if (StringUtils.hasText(accessToken)) {
        // Blocked된 token인지 유효성 검사
        try {
            MemberLoginActive member = jwtTokenProvider.resolveAccessToken(accessToken);
            if (member == null) {
                log.error("인증된 사용자 정보를 찾을 수 없습니다.");
                throw new JwtException(JwtErrorCode.ACCESS_TOKEN_EXPIRED);
            }
        } catch (Exception e) {
            log.error("액세스 토큰 유효성 검사 중 예상치 못한 오류가 발생했습니다: " + e.getMessage());
            throw new JwtException(JwtErrorCode.ACCESS_TOKEN_INVALID);
        }
    }
}
```

```

de.INVALID_CLAIMS);
        } else {
            log.info("회원 ID : {} - 요청 시도:",
", member.id());
            if (isBlockedAccessToken(member.e
mail(), accessToken)) {
                throw new JwtException(JwtErr
orCode.EXPIRED_TOKEN);
            }
        }
        SecurityContextHolder.getContext()
            .setAuthentication(createAuth
enticationToken(member));
    } catch (JwtException e) {
        SecurityContextHolder.clearContext();
        log.info("JWT 검증 실패: {}", e.getErro
rCode().getErrorMessage());

        // Message 객체를 사용하여 응답을 구성
        Message<Void> errorMessage = Message.
fail(
            e.getErrorCode().getHttpStatu
s().toString(),
            e.getErrorCode().getErrorMess
age()
        );
        // HTTP 응답 설정
        response.setStatus(e.getErrorCode().g
etHttpStatus().value());
        response.setContentType("application/
json; charset=UTF-8");
        // JSON 형태로 변환하여 응답 본문에 작성
        ObjectMapper mapper = new ObjectMappe
r();
        PrintWriter out = response.getWriter
();
        out.print(mapper.writeValueAsString(e
rrorMessage));
    }
}

```

```

        out.flush();
        return;
    }
}

filterChain.doFilter(request, response);
}

private String resolveTokenFromCookie(HttpServletRequest request) {
    // 유니티와의 통신은 웹에서와 달리 쿠키를 활용하기 어렵
    // 때문에 해당 헤더를 통해
    // 인증 및 인가 수행합니다.
    String bearerToken = request.getHeader("Autho
    rization");
    if (StringUtils.hasText(bearerToken)) {
        if (bearerToken.startsWith(BEARER_PREFI
        X)) {
            return bearerToken.substring(7);
        }
        return bearerToken;
    }

    // 쿠키에서 "accessToken" 키의 value를 가져옵니다.
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (Cookie cookie : cookies) {
            if (cookie.getName().equals("accessTo
            ken")) {
                return cookie.getValue();
            }
        }
    }
    return null;
}

// 인증된 사용자 정보를 담는 JwtAuthentication 객체
// 를 생성합니다.
private JwtAuthenticationToken createAuthenticati

```

```

onToken(MemberLoginActive user) {
    return new JwtAuthenticationToken(user, "", 
        List.of(new SimpleGrantedAuthority(user.role().name
        ())));
}

// 전송된 accessToken이 새로운 로그인을 통해 만료되었는지 확인하기 위한 메서드입니다.
private boolean isBlockedAccessToken(String email, String token) {
    List<String> blockedTokens = blockedAccessTokenService.findAllByEmail(email);
    // 각 요소에 대해 equals 메서드를 사용하여 지정된 문자열과 완벽하게 일치하는지 비교합니다.
    return blockedTokens.contains(token);
}
}

```

중복 로그인 방지

login method

```

public LoginResponse login(LoginRequest loginRequest) {
    ...

    // 현재 member 인스턴스의 currentToken attribute 값이 존재할 때
    // 기존의 발급 받은 accessToken을 blocked_access_token 테이블에 저장하고
    // Jwt 필터에서 이를 조회하여 중복 로그인 상태를 방지합니다.
    if (member.getCurrentToken() != null){
        blockedAccessTokenService.save(member.getEmail(),
            member.getCurrentToken());
    }
    LoginResponse loginResponse = jwtTokenService.issue
        AndSaveTokens(member);
    member.setCurrentToken(loginResponse.jwtToken().acces
        ssToken());
}

```

```
        return loginResponse;
    }
```

BlockedAccessTokenService.java

```
@Service
@RequiredArgsConstructor
public class BlockedAccessTokenService {
    private final BlockedAccessTokenRepository blockedAccessTokenRepository;

    @Value("${jwt.accessExpiration}")
    private Duration BLOCKING_DURATION;

    /**
     * 블록된 접근 토큰을 저장합니다.
     *
     * @param email 토큰과 연관된 사용자 이메일
     * @param token 저장할 블록된 토큰
     */
    @Transactional
    public void save(String email, String token) {
        // 토큰의 만료 시간을 계산합니다.
        long expirationTime = Instant.now().plus(BLOCKING_DURATION).toEpochMilli();
        // 해당 이메일에 이미 토큰이 발급되었으므로 추가확인은 하지
        // 않습니다.
        BlockedAccessToken blockedAccessToken = BlockedAccessToken.builder()
            .email(email)
            .token(token)
            .expirationTime(expirationTime)
            .build();
        // 블록된 접근 토큰을 저장합니다.
        blockedAccessTokenRepository.save(blockedAccessToken);
    }
}
```

```

    /**
     * 주어진 이메일에 대한 모든 블록된 접근 토큰을 찾습니다.
     *
     * @param email 블록된 토큰을 찾을 사용자 이메일
     * @return 유효한 블록된 토큰 목록
     */
    @Transactional
    public List<String> findAllByEmail(String email){
        // 현재 시간을 밀리초로 가져옵니다.
        long now = Instant.now().toEpochMilli();
        // 만료된 토큰을 삭제합니다.
        blockedAccessTokenRepository.deleteExpiredTokens
        ByEmail(now, email);
        // 주어진 이메일에 대한 모든 블록된 토큰을 찾습니다.
        List<BlockedAccessToken> blockedTokens
        = blockedAccessTokenRepository.findAllByEmail(em
ail);

        // Blocked 되었지만 아직 만료되지 않은 토큰 필
        터링하여 반환합니다.
        return blockedTokens.stream()
            .filter(token -> token.getExpirationTime
() > now)
            .map(BlockedAccessToken::getToken)
            .toList();
    }

}

```

4 BUSINESS SERVER

BUSINESS SERVER 파일 구조

HOW TO

Netty

- 자바 네트워크 애플리케이션 프레임워크
- 비동기 이벤트 기반 네트워크 응용프로그램 프레임워크

Reactor Pattern으로 비동기 Non-Blocking TCP 서버 구현

ChannelPipeline에 Handler 등록

- LineBasedFrameDecoder
- 사용자 정의 ChannelHandlerAdapter

메서드	설명
handlerAdded	client와 최초로 연결되었을 경우 - logging
handlerRemoved	client와의 연결이 끊겼을 경우 - 연결이 끊어진 session 삭제 - logging
exceptionCaught	예외 처리 - logging

데이터 처리 방법

- **MessagePack(네트워크 통신 데이터 직렬화 라이브러리)** 사용
 - 송신 데이터 **MessageBufferPacker**

```
MessageBufferPacker packer = MessagePack.newDefaultBuilder()
    .arrayHeader(2)
    .string("userList")
    .string(String.valueOf(channel.getSessions().size()));

byte[] bytes;
packer.packArrayHeader(2);      // 배열의 Header 지정
packer.packString("userList"); // 문자열 - packString
packer.packString(String.valueOf(channel.getSessions().size()));
bytes = packer.toByteArray();  // 바이트 배열로 변환후 송
```

- 수신 데이터 **MessageUnpacker**

```
MessageUnpacker unpacker = MessagePack.newBuilder()
    .arrayHeader(2)
    .string("userList");
    .string(String.valueOf(channel.getSessions().size()));
```

```

int arrayLength = unpacker.unpackArrayHeader();
// 배열 Header unpacking
Type type = Type.findIndex(unpacker.unpackInt());
// unpackInt - 탑입 반환
String userName = unpacker.unpackString();
// 문자열 - unpackString

```

- Type 지정 **ENUM**

```

public enum Type {
    LIVESERVER( idx: 0 ), // 최초 연결 (라이브 서버)
    CLIENT( idx: 1 ), // 최초 연결 (유저)
    CREATE( idx: 2 ), // 방 생성
    JOIN( idx: 3 ), // 방 참가
    LEVAE( idx: 4 ), // 방 나가기
    READY( idx: 5 ), // 게임 준비
    START( idx: 6 ), // 게임 시작
    ITEM( idx: 7 ), // 아이템 사용
    END( idx: 8 ), // 게임 끝
    Map( idx: 9 ), // 맵 바꾸기
    POSITION( idx: 10 ),
    SPOSITION( idx: 11 ),
    TEAMCHANGE( idx: 12 ), // 팀 바꾸기
    CHARCHANGE( idx: 13 ); // 캐릭터 바꾸기

    private final int idx;

    Type(int idx) { this.idx = idx; }

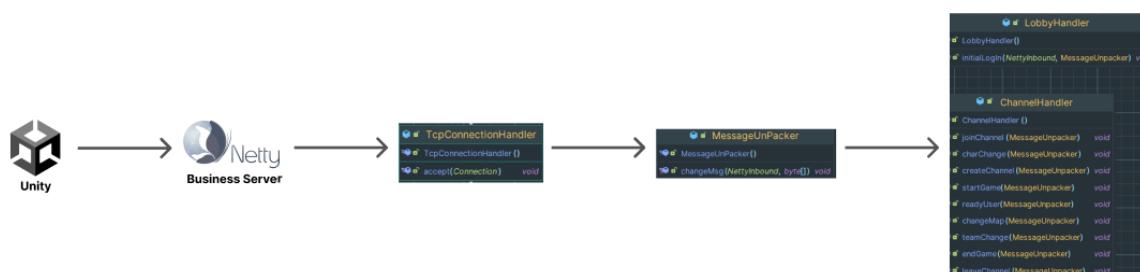
    public static Type findByIndex(int index) {
        for (Type type : Type.values()) {
            if (type.idx == index) {
                return type;
            }
        }
        throw new IllegalArgumentException("Unknown index: " + index);
    }
}

```

CLIENT - BUSINESS SERVER 데이터 흐름

① LOBBY & CHANNEL

- Unity 로그인 후 Business Server 최초접속
- Business Server에서 Type 확인 후 case 별 handler 호출
- 데이터 확인 후 각 Channel에 Broadcasting



5 CHATTING SERVER

CHATTING SERVER 파일 구조

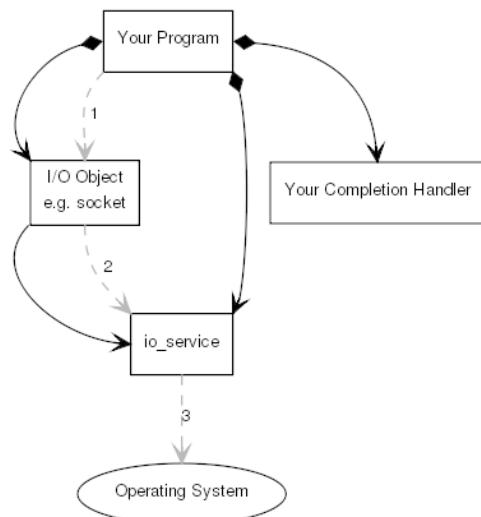


Boost.Asio 라이브러리

- Boost.Asio는 크로스 플랫폼을 지원하는 라이브러리로, Linux 배포 환경과 Windows 개발 환경 모두에서 별도의 조치 없이 정상적으로 작동합니다. 이 라이브러리는 비동기 및 동기 작업을 모두 수행할 수 있는 기능을 제공합니다.

비동기 데이터 처리

- Boost.Asio를 이용한 비동기 방식의 소켓 통신 프로그래밍은 데이터를 효율적으로 처리할 수 있게 해줍니다. 아래 그림은 Boost.Asio의 비동기방식 소켓 통신 프로그래밍의 알고리즘을 나타냅니다.



MessagePack

- MessagePack은 데이터 압축 라이브러리로, JSON 형식의 데이터를 압축할 경우 원본 데이터의 67% 크기로 줄일 수 있어 데이터 송수신에 있어서 효율적입니다.

```
JSON 27 bytes
{channelIndex: 0, message: "테스트 입니다."}

MessagePack (hex) 18 bytes 67 %
82 a7 63 6f 6d 70 61 63 74 c3 a6 73 63 68 65 6d 61 00
```

데이터 관리

1. 채널 관리

- 모든 채널을 관리하기 위해 **Channel_List**를 싱글톤으로 생성합니다.

2. 채널

- 채널은 여러 세션들로 구성되며, 같은 채널 내의 세션들은 서로 채팅이 가능합니다.

3. 세션

- 각 세션은 **endpoint**를 갖고 있어 서버와 데이터를 송수신할 수 있습니다.

서버 동작 시나리오

1. 클라이언트 최초 접속

- 클라이언트가 최초로 접속하면 **Channel_List**의 0번 채널(로비)에 세션 정보를 저장합니다.

2. 채널 이동

- ENUM**을 사용하여 수신한 데이터를 파싱하고 원하는 채널로 세션 정보를 이동시킵니다.

3. 채팅

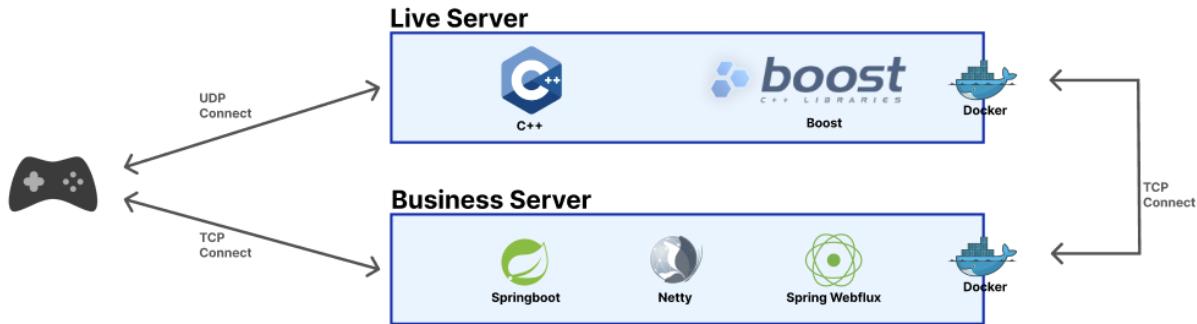
- ENUM**을 이용하여 수신한 데이터를 파싱하고 현재 채널에 접속한 세션들에게 메시지를 전송합니다.

4. 종료

- 클라이언트가 종료될 때, 세션에 저장된 현재 채널 정보를 이용하여 채널에서 세션 정보를 제거합니다. 로비 채널을 제외한 다른 채널에 현재 세션이 존재하지 않을 경우, 해당 채널도 제거됩니다.

6 LIVE SERVER

💡 HOW TO



Boost.Asio 라이브러리

- Boost.Asio는 크로스 플랫폼을 지원하는 라이브러리로, Linux 배포 환경과 Windows 개발 환경 모두에서 별도의 조치 없이 정상적으로 작동합니다. 이 라이브러리는 비동기 및 동기 작업을 모두 수행할 수 있는 기능을 제공합니다.

MessagePack

- MessagePack은 데이터 압축 라이브러리로, JSON 형식의 데이터를 압축할 경우 원본 데이터의 67% 크기로 줄일 수 있어 데이터 송수신에 있어서 효율적입니다.

멀티쓰레드

1. 쓰레드 구성

- TCP Connection Thread: 비즈니스 서버로 부터 데이터를 수신 및 처리하기 위한 단일 쓰레드
- UDP Receiver Thread: 클라이언트로 부터 데이터를 수신하기 위한 단일 쓰레드
- Worker Thread: 수신한 데이터를 처리하기 위한 쓰레드 풀

2. Message Queue & Mutex

- Message Queue: UDP Receiver 쓰레드가 수신한 데이터를 Worker Thread가 데이터를 처리할 수 있도록 임시로 저장하는 역할
- Mutex: 공유된 자원의 데이터 혹은 임계영역에 여러개의 쓰레드가 동시에 접근하는 것을 막아주는 역할

UDP / TCP

1. UDP

- 비연결형 서비스로 데이터그램 방식
- TCP에 비해 빠른 속도
- 게임 플레이어의 위치 정보를 빠르게 송수신하는 데 사용

2. TCP

- 연결 지향 방식으로 패킷 교환 방식을 사용
- 높은 신뢰성을 보장
- 게임 시작/종료 메시지와 같이 손실이 일어나서는 안 되는 데이터를 송수신하는 데 사용

서버 동작 시나리오

1. 비즈니스 서버에 연결

- 비즈니스 서버에 Live Server 식별 메시지를 전송하여 연결

2. 게임 시작

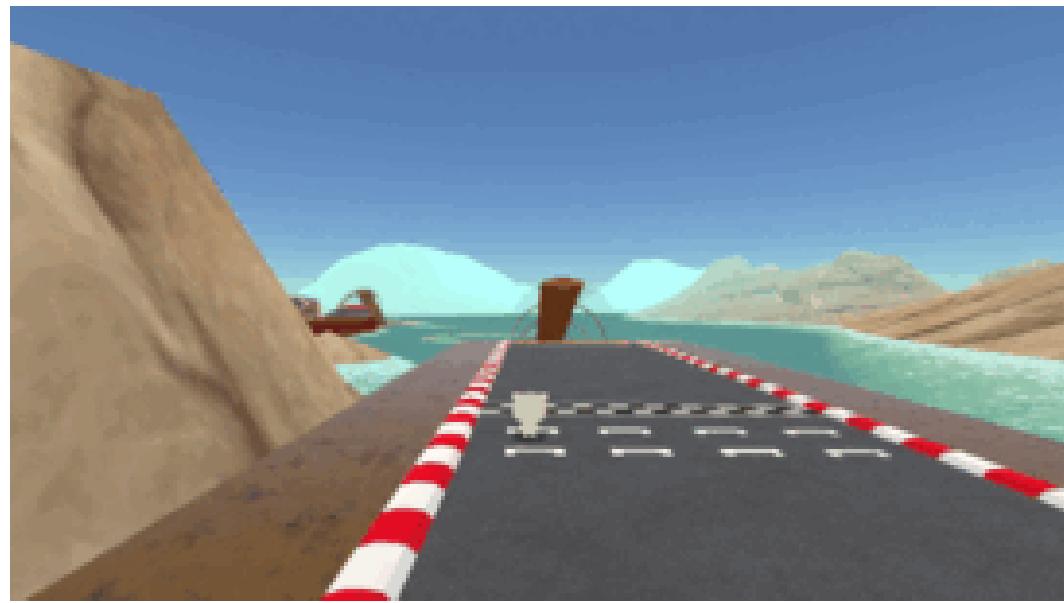
- 비즈니스 서버로부터 게임 시작 식별 메시지를 수신
- 채널 생성 및 클라이언트 연결 대기
- 클라이언트 연결시 각 클라이언트의 위치 데이터를 송수신

3. 게임 종료

- 비즈니스 서버로부터 게임 종료 식별 메시지를 수신
- 일정 대기시간을 거친 뒤 채널 제거

7 UNITY

- PC 버전
 - 인트로

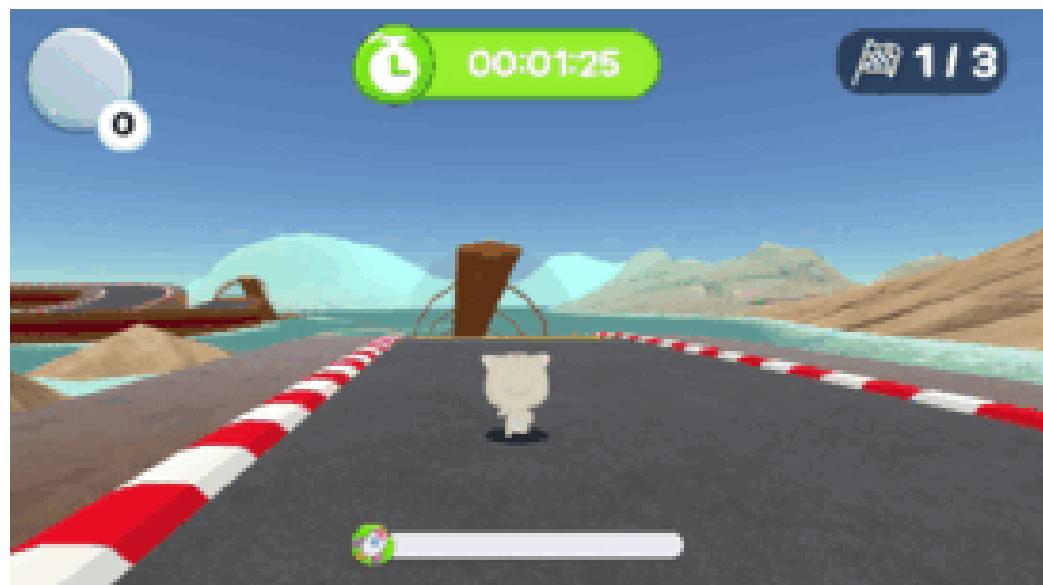


- 게임 시작 시 Main Camera의 Transform을 변환 시켜 게임 인트로 및 카운트 다운 구현
- 인트로 및 카운트 다운 진행 중 움직이지 못하도록 제한
 - 게임 플레이



- 키보드 버튼 클릭에 따라 Horizontal, Vertical 값을 Raw 형태로 받아 -1, 1로 입력값으로 받음
- Rigidbody를 통해 AddForce 등을 통해 Transform 변화를 통해 움직임 구현

- Left Shift를 눌렀을 때 Drift 발동, Player의 회전 속도를 상승 시켜 빠른 회전을 통해 드리프트 효과 구현

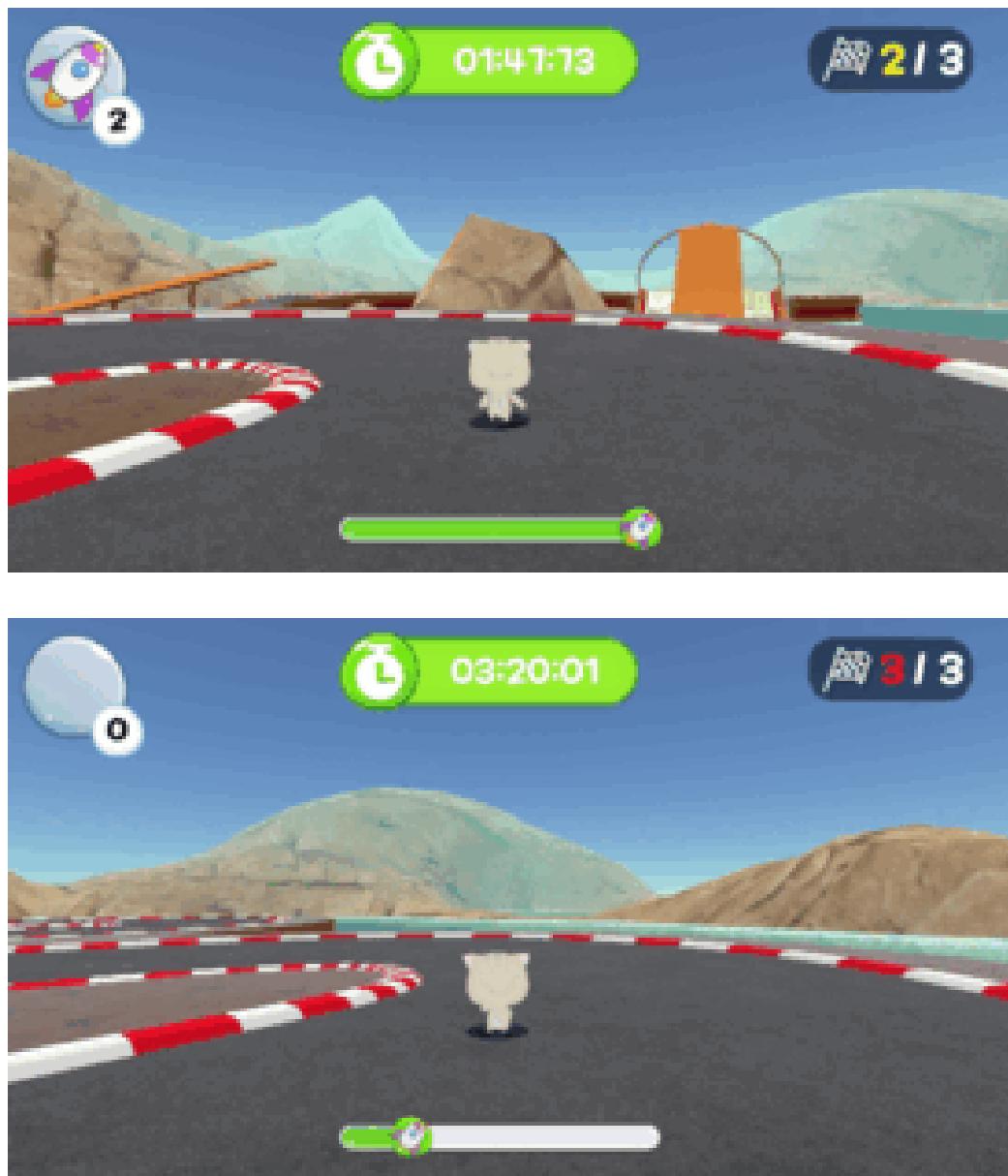


- 드리프트 시 부스터 게이지 충전
- 드리프트 시 TrailRender 생성을 통해 스키드 마크 구현
- 부스터 최대 2개 까지 충전
- 부스터 사용 동안 중첩 부스터 사용 제한
- 부스터 사용 시 ParticleSystem 재생, 불꽃 효과 및 스피드 라인을 통해 속도 감 있는 부스터 구현
- OnTrigger 이벤트를 통해 부스터 패드를 밟았을 시에도 부스터 발동

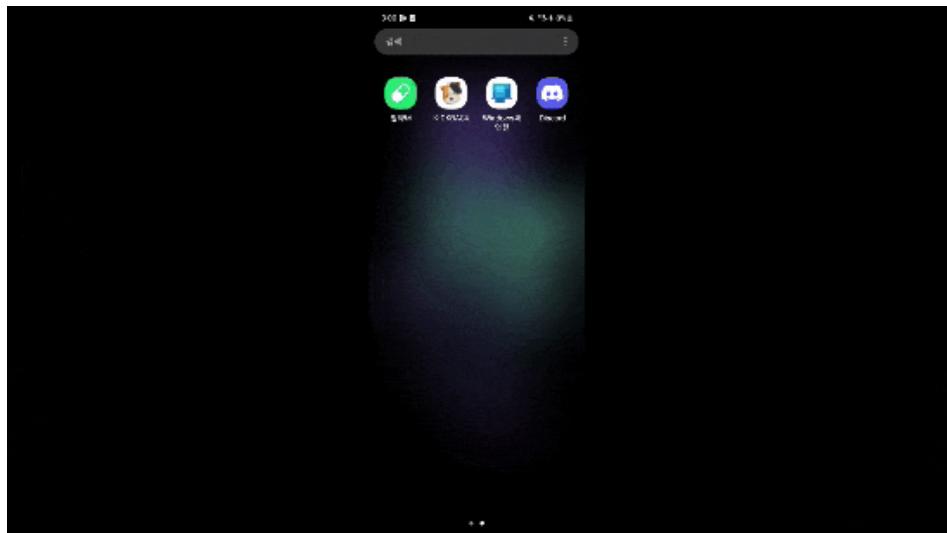


- RayCast를 통해 땅 감지 및 점프 기능 구현





- 레벨 디자인에 투명한 체크 포인트 배치
 - 체크 포인트를 지날 때 리스폰 Transform 초기화
 - 맵에 끼거나 맵 밖으로 나가졌을 때 데드존에 떨어졌을 때 리스폰
 - 각 체크 포인트 및 데드존은 OnTrigger 이벤트를 통해 초기화
 - 체크 포인트를 순서대로 통과했을 때 랩 측정
 - 올바르게 체크 포인트를 통과하지 못할 경우 경고 메세지 출력 후 강제 리스폰
 - 각 맵 별 정해진 랩을 모두 통과 했을 시 게임 종료
- 안드로이드 버전
 - 앱 다운로드 후 실행

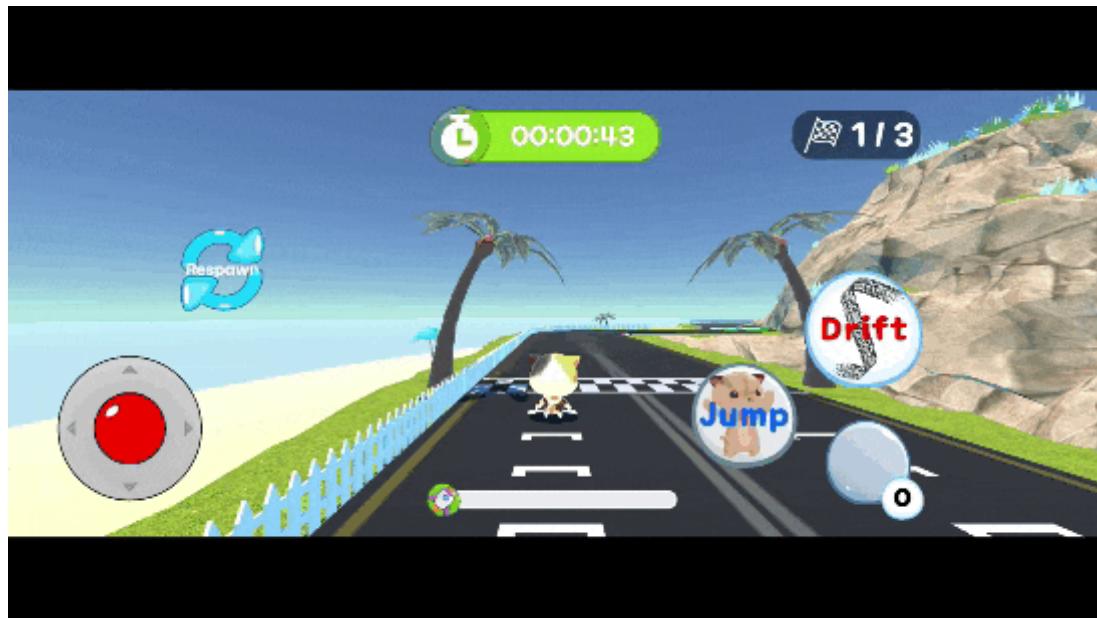


- 인트로

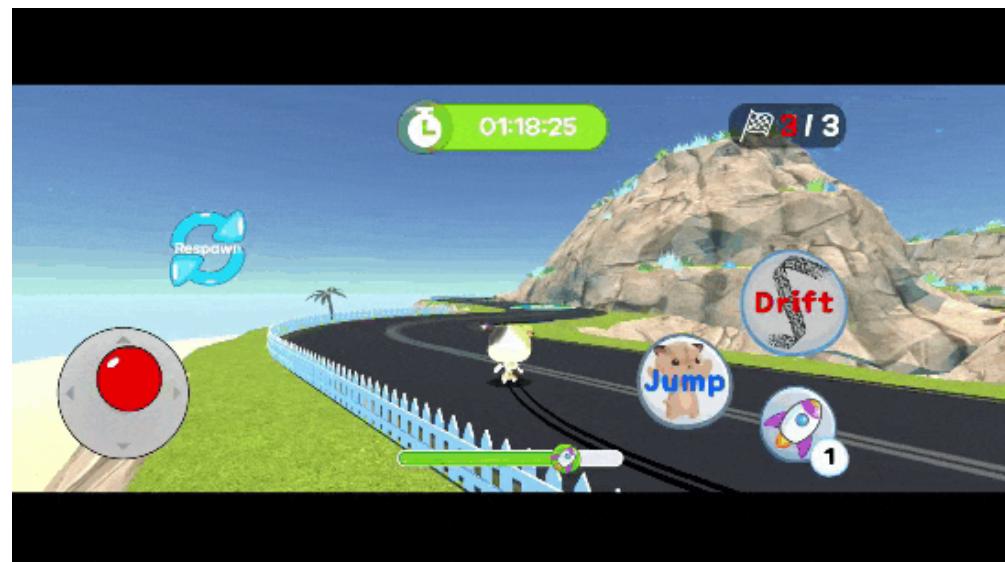


- 게임 시작 시 Main Camera의 Transform을 변환 시켜 게임 인트로 및 카운트 다운 구현
- 인트로 및 카운트 다운 진행 중 움직이지 못하도록 제한

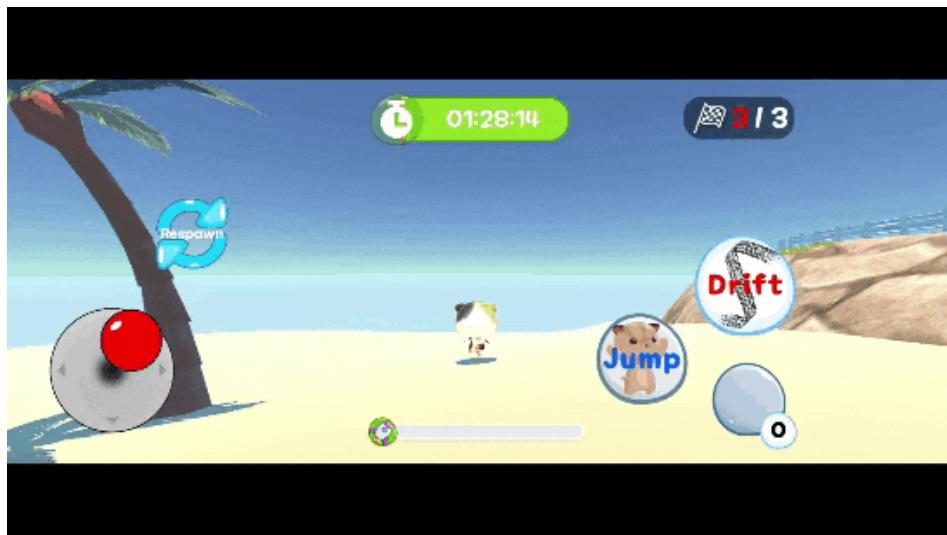
- 게임 플레이



- 조이스틱의 움직임에 따라 Horizontal, Vertical 값을 Raw 형태로 받아 -1, 1로 입력값으로 받음
- Rigidbody를 통해 AddForce 등을 통해 Transform 변화를 통해 움직임 구현
- 각각의 기능들을 OnClick 혹은 Event Trigger를 통해 버튼에 담아 버튼 클릭 시 기능이 구현
- Drift를 눌렀을 때 Player의 회전 속도를 상승 시켜 빠른 회전을 통해 드리프트 효과 구현
- 드리프트 시 부스터 게이지 충전
- 드리프트 시 TrailRender 생성을 통해 스키드 마크 구현

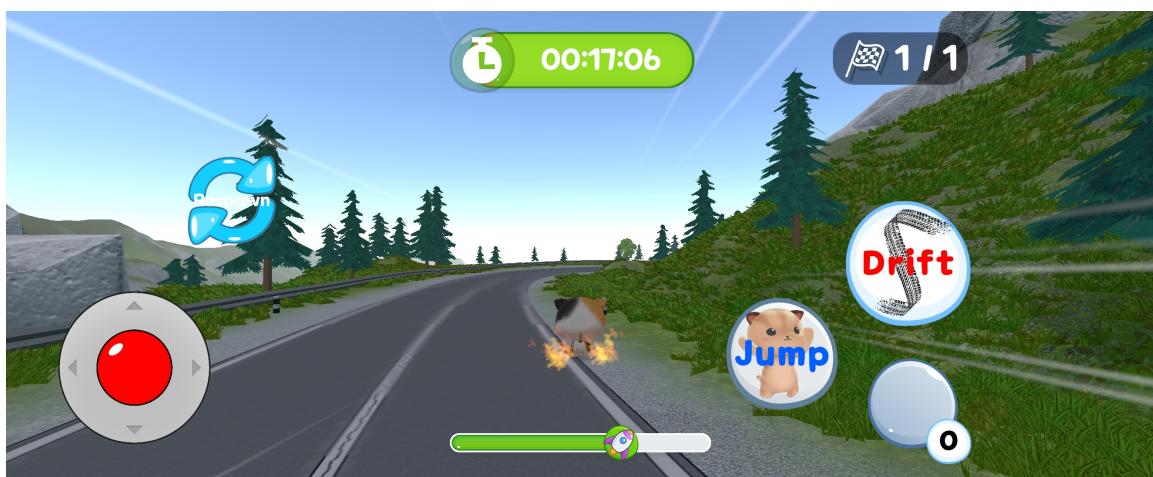
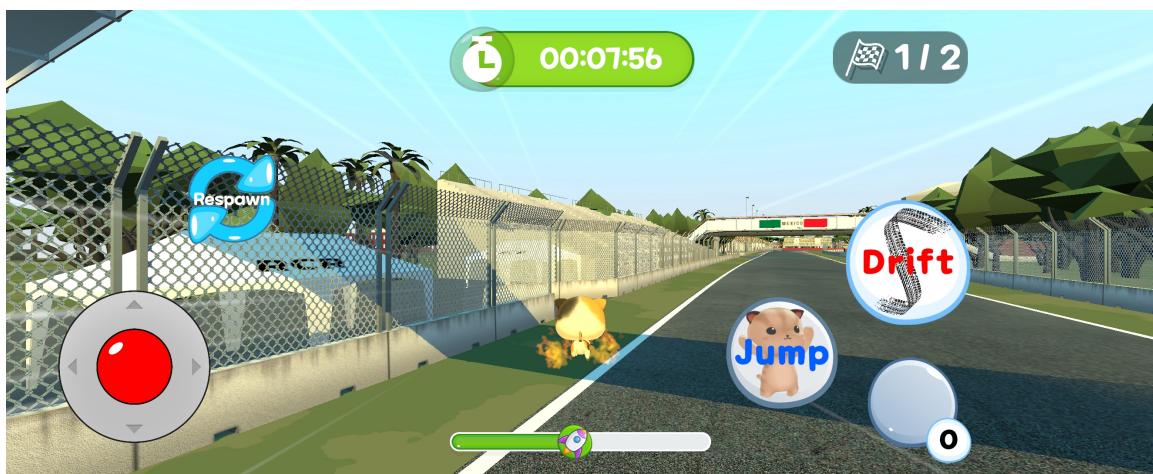


- 부스터 최대 2개 까지 충전
- 부스터 사용 동안 중첩 부스터 사용 제한
- 부스터 사용 시 ParticleSystem 재생, 불꽃 효과 및 스피드 라인을 통해 속도감 있는 부스터 구현
- OnTrigger 이벤트를 통해 부스터 패드를 밟았을 시에도 부스터 발동



- 다양한 장애물 배치를 통해 색다른 레이싱 구현

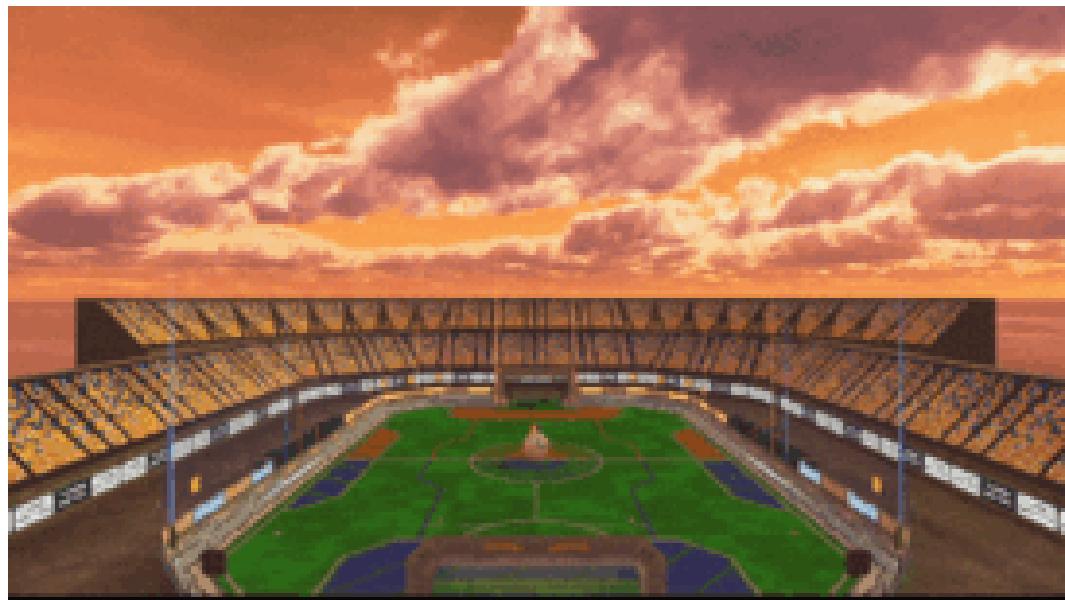
- RayCast를 통해 땅 감지 및 점프 기능 구현
- 레벨 디자인에 투명한 체크 포인트 배치
- 체크 포인트를 지날 때 리스폰 Transform 초기화
- 맵에 끼거나 맵 밖으로 나가졌을 때 데드존에 떨어졌을 때 리스폰
- 각 체크 포인트 및 데드존은 OnTrigger 이벤트를 통해 초기화
- 체크 포인트를 모두 통과했을 때 랩 측정
- 각 맵 별 정해진 랩을 모두 통과 했을 시 게임 종료



- 다양한 맵 구성
- 비탈길에서 Physics Raycast를 통해 지면과 붙어 있는지 확인 후 자연스러운 움직임 구현

- 축구 모드

- 인트로



- 게임 시작 시 Main Camera의 Transform을 변환 시켜 게임 인트로 및 카운트 다운 구현
 - 인트로 및 카운트 다운 진행 중 움직이지 못하도록 제한

- 움직임





- 드리프트 시 부스터 게이지 충전
- 드리프트 시 TrailRender 생성을 통해 스키드 마크 구현
- 부스터 최대 2개 까지 충전
- 부스터 사용 동안 중첩 부스터 사용 제한
- 부스터 사용 시 ParticleSystem 재생, 불꽃 효과 및 스피드 라인을 통해 속도 감 있는 부스터 구현
- 골 넣기



- OnTrigger를 통해서 골대에 공이 들어가게 될 경우 골로 체크 후 점수 측정

- 골득실과 승리, 무승부, 패배를 통해 점수 획득 및 랭킹 측정
- 랭킹 페이지

랭킹 시스템

- 맵 별 랩타임 기록



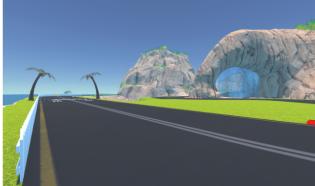
MEXICO CEBU DOWNHILL BORYEONG

닉네임을 검색해보세요!

검색

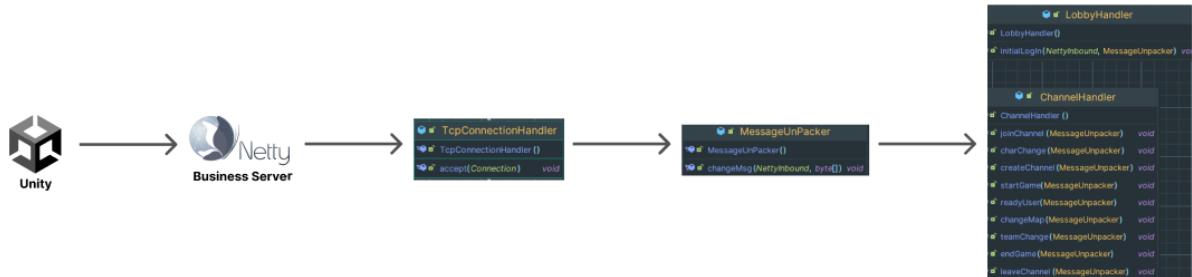


닉네임 : 012
랭킹 : 1
기록 : 01:03:15



등수	닉네임	기록
		012 01:03:15
		qq 01:04:25

- 게임 종료 후 자신이 기록한 랭 타임은 HTTP 프로토콜 통신을 통해 requestUrl에 PUT 요청 보냄
- 갱신된 랭 타임은 웹페이지 랭킹 페이지에서 확인



8 API 명세서

[Kickback API 명세서 링크](#)

9 Convention

태그 이름	설명
Feat	새로운 기능을 추가할 경우
Fix	버그를 고친 경우
Design	CSS 등 사용자 UI 디자인 변경
!BREAKING CHANGE	커다란 API 변경의 경우
!HOTFIX	급하게 치명적인 버그를 고쳐야하는 경우
Style	코드 포맷 변경, 세미 콜론 누락, 코드 수정이 없는 경우
Refactor	프로덕션 코드 리팩토링
Comment	필요한 주석 추가 및 변경
Docs	문서를 수정한 경우
Test	테스트 추가, 테스트 리팩토링(프로덕션 코드 변경 X)
Chore	빌드 태스크 업데이트, 패키지 매니저를 설정하는 경우(프로덕션 코드 변경 X)
Rename	파일 혹은 폴더명을 수정하거나 옮기는 작업만인 경우
Remove	파일을 삭제하는 작업만 수행한 경우

👩 Branch종류

- master: 최종 배포 branch
- develop: 다음 출시 버전을 개발하는 branch (merge 대상 branch)
- release: 배포 branch
- test: CI/CD 및 기타 테스트용 branch

👤 커밋 브랜치

- 개인 브랜치(영문 이름) -> Merge Request → develop 브랜치 -> 개발완료 -> master 브랜치

👨‍👩‍👧 팀 커밋 컨벤션

- <타입>[적용범위]: <설명>
 - ex) ssafy브랜치에 Feat[Common]: README.md update

- 적용범위
 1. Common: 공통 작업
 2. Live: 라이브 서버
 3. Business: 비즈니스 서버
 4. Auth: 웹 및 인증 서버
 5. Chat: 채팅 서버
 6. Web: 웹 프론트엔드
 7. Unity: 유니티

10 팀원 별 담당 역할 및 업무

이름	담당 역할 및 업무
김선욱(팀장)	Unity 개발
김영일	Buisness Server, Chatting Server, Web Front-End 개발
서민주	Web Server, Web Front-End 개발
이향우	Live Server 개발, CICD 담당
장동재	Live Server, Business Server 개발
조현호	Auth(Web) Server 개발