











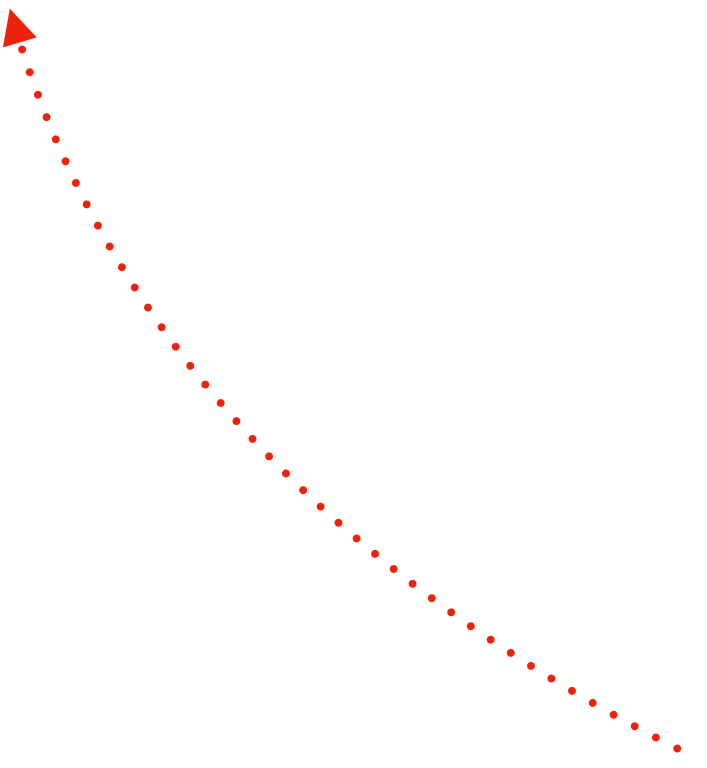
Implicit Grant

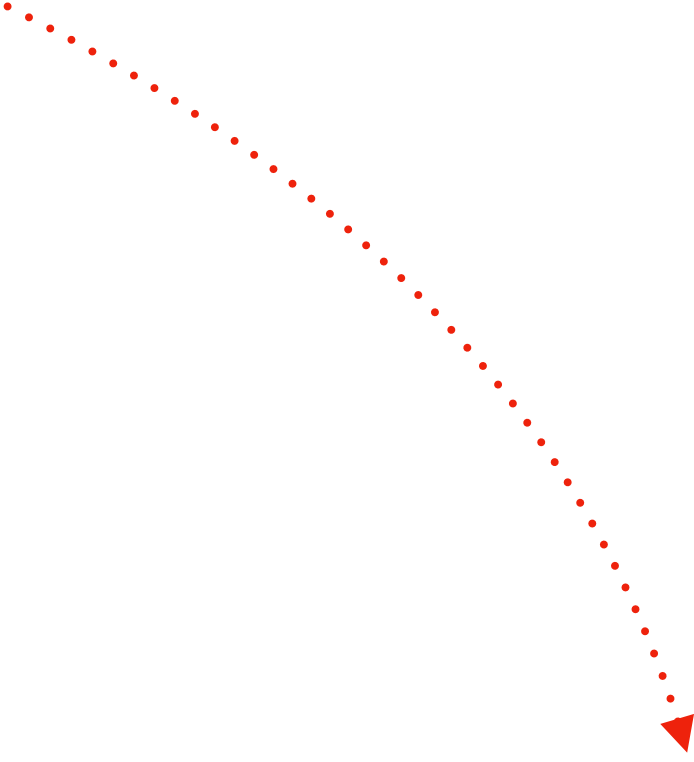






참고 링크: <https://showbugs.github.io/2017-11-16/Auth-%E8%80%B4%E6%97%B7%E6%9D%B6%EAEAE8C>





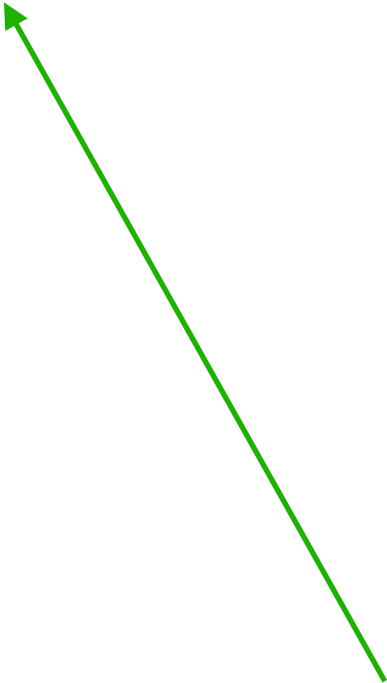


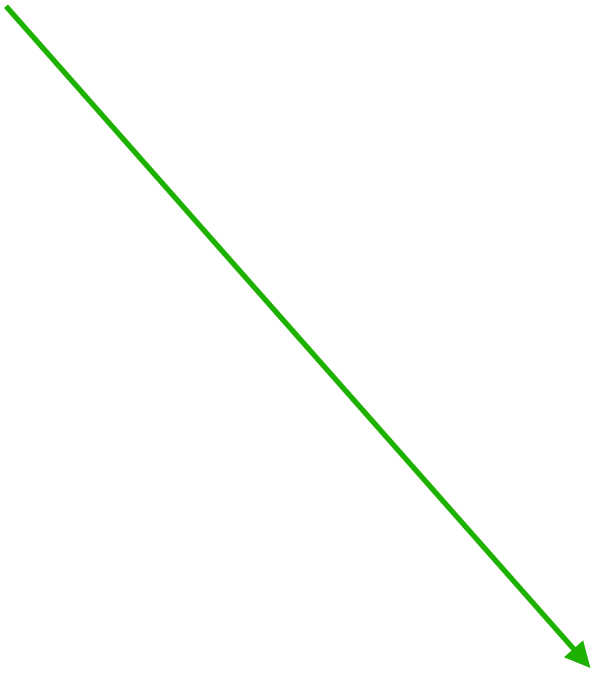
카카오로 로그인하기



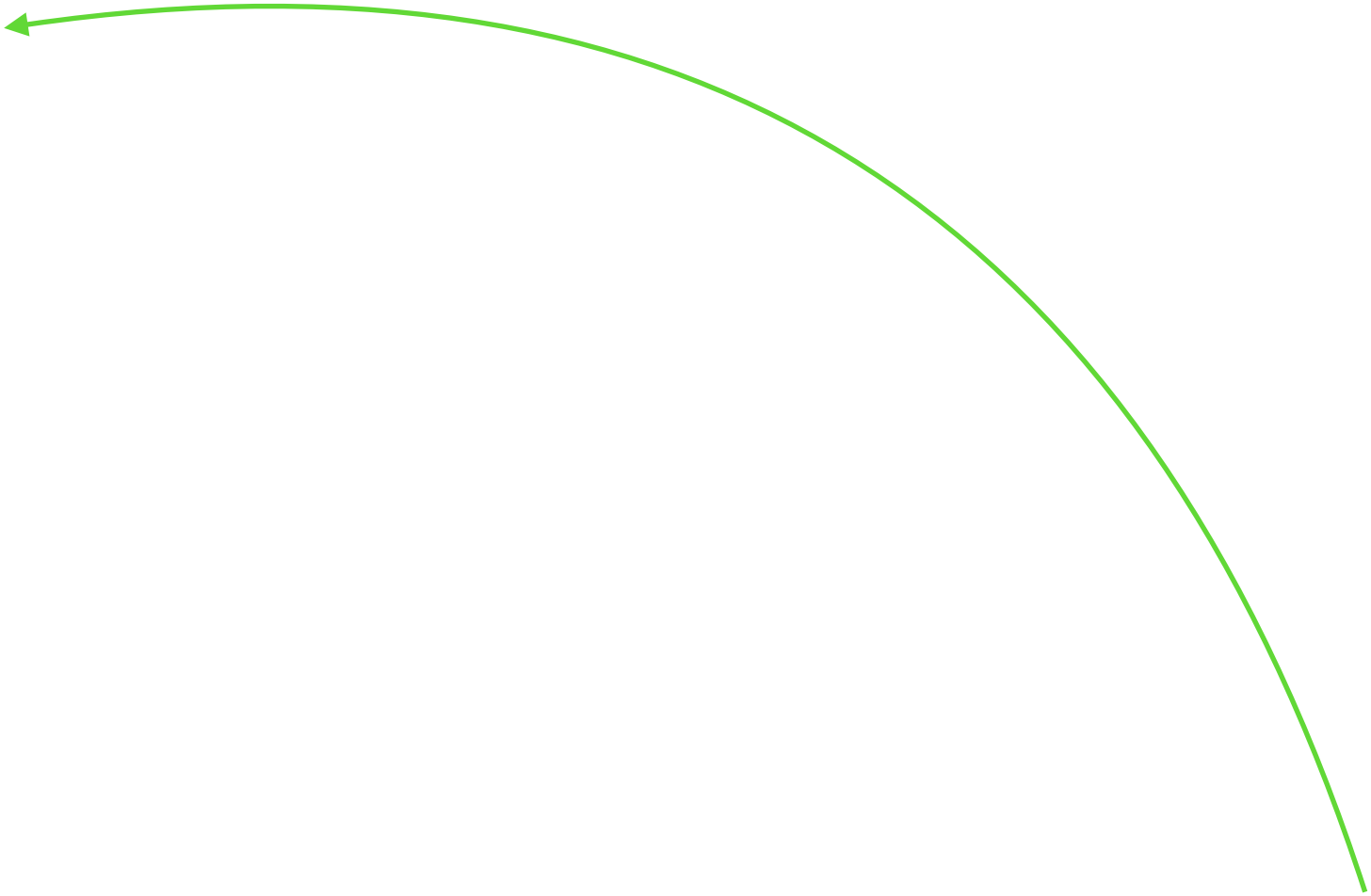


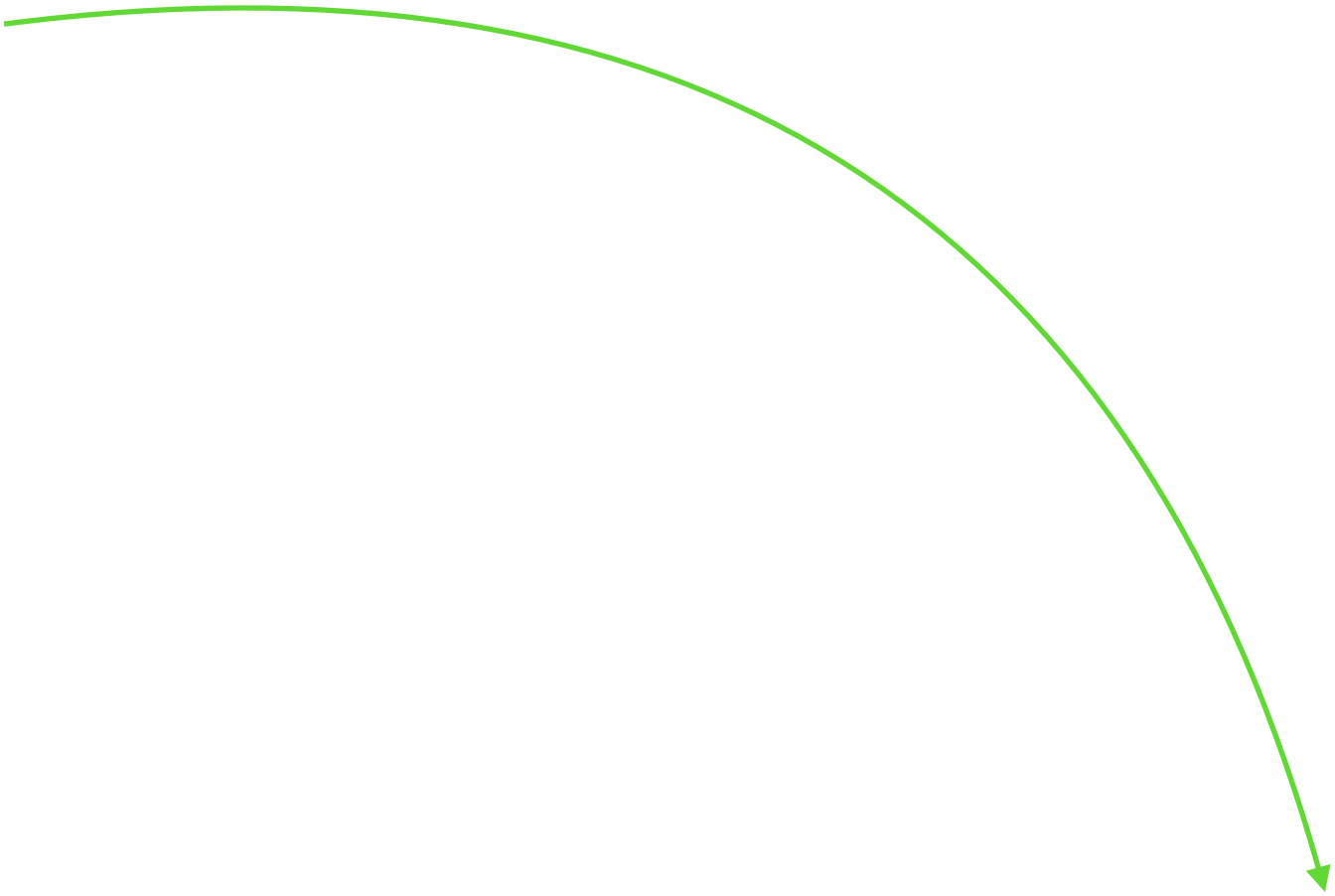















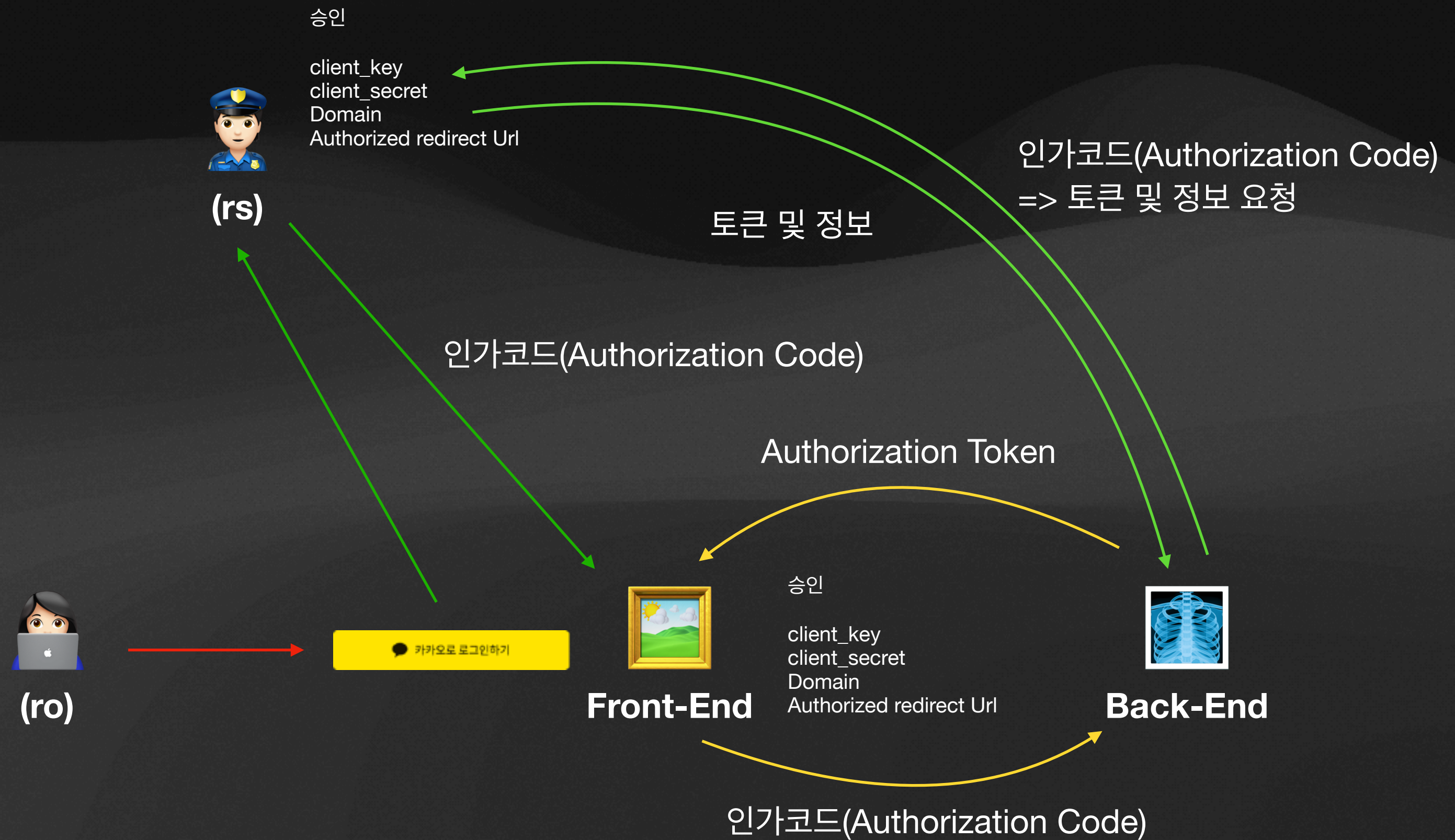


Social Login

OAuth 2.0 [3-legged-auth]

- ✓ Authorization Code Grant
- ✓ Implicit Grant
- ✓ Password Credentials Grant
- ✓ Client Credentials Grant

-  **Client**
 - 서비스의 제작자 및 제공하는 '나'
-  **Resource owner(ro)**
 - 서비스를 사용하기 원하는 'user'
-  **Resource Server(rs)**
 - user의 정보를 가지고 있는 'provider'



Social Login Process



Front-End

```
you, 4 hours ago | 2 authors (you and others)
@ApiTags('Auth')
@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}
  @ApiOperation({
    summary: '카카오 로그인',
    description: '카카오 로그인을 하는 API입니다.',
  })
  @Post('kakao')
  async kakaoLogin(@Body() body, @Res() res) {
    try {
      const { payload } = body;
      console.log(body);
      if (!payload) {
        throw new BadRequestException('카카오정보가 없습니다.');
```

```
    }
    const kakao = await this.authService.kakaoLogin(payload);
    console.log('카카오 컨트롤러', kakao);
    const token = await this.authService.kakaoUser(kakao);
    const { AccessToken, RefreshToken } = token;
    res.setHeader('Authorization', AccessToken);
    res.setHeader('RefreshToken', RefreshToken);
    console.log(token);
    if (!kakao.id) {
      throw new BadRequestException('카카오 정보가 없습니다.');
```

```
    }
    const kakaoKey = '1b6507f790effacebec0df34314f133'; // 이부분은 REST_API KEY
    const kakaoTokenUrl = 'https://kauth.kakao.com/oauth/token';
    const kakaoUserInfoUrl = 'https://kapi.kakao.com/v2/user/me';
    const body = {
      grant_type: 'authorization_code',
      client_id: kakaoKey,
      redirect_uri: 'http://localhost:3000/oauth',
      code,
    };
    const headers = {
      'Content-Type': 'application/x-www-form-urlencoded;charset=utf-8',
    };
    try {
      const response = await axios({
        method: 'POST',
        url: kakaoTokenUrl,
        timeout: 30000,
        headers,
        data: qs.stringify(body),
      });
      if (response.status === 200) {
        const headerUserInfo = {
          'Content-Type': 'application/x-www-form-urlencoded;charset=utf-8',
          Authorization: 'Bearer ' + response.data.access_token,
        };
        const responseUserInfo = await axios({
          method: 'GET',
          url: kakaoUserInfoUrl,
          timeout: 30000,
          headers: headerUserInfo,
        });
        if (responseUserInfo.status === 200) {
          console.log('리스폰스 서비스 유저인포', responseUserInfo);
          return responseUserInfo.data;
        } else {
          throw new UnauthorizedException();
        }
      } else {
        throw new UnauthorizedException();
      }
    } catch (error) {
      throw new UnauthorizedException();
    }
  }
}
```

```
async kakaoUser(kakao: KakaoDataDto) {
  const { id, properties, kakao_account } = kakao;
  const { email } = kakao_account;
  const kakaoUser = new UserEntity();
  kakaoUser.password = String(id);
  kakaoUser.name = 'kakao';
  kakaoUser.email = email;
  kakaoUser.nickname = email.split('@')[0];
  kakaoUser.profileImg = properties.profile_image
    ? properties.profile_image
    : process.env.DEFAULT_IMG_URL;

  const existUser: UserEntity = await this.userRepository.findOneBy({
    email,
  });

  if (!existUser) {
    console.log('신규유저', kakaoUser);
    const newUser = await this.userRepository.insert(kakaoUser);
    const tokenId: number = newUser.identifiers[0].id;
    const token = await this.createToken({ tokenId });
    return token;
  }

  const ExistUser = existUser;
  console.log('기존유저', existUser);
  const tokenId: number = ExistUser.id;
  const token = await this.createToken({ tokenId });
  return token;
}
```

```
async createToken(req: string | object): Promise<tokenType> {
  const payload = req;

  const accessToken = this.jwtService.sign(payload, {
    expiresIn: '10m',
    secret: process.env.JWT_SECRET,
  });

  const refreshToken = this.jwtService.sign(payload, {
    expiresIn: '7d',
    secret: process.env.JWT_SECRET,
  });

  return {
    AccessToken: `Bearer ${accessToken}`,
    RefreshToken: `Bearer ${refreshToken}`,
  };
}
```