

Table of Contents

Coinstack SignOn 소개	1.1
본 문서에 관하여	1.2
전체 조건	1.2.1
문서 정보	1.2.2
사전 정의	1.2.3
저작권 고지	1.2.4
연락처	1.2.5
개요	1.3
OAuth 2.0	1.3.1
개념 및 용어	1.3.1.1
클라이언트(Client)	1.3.1.2
인가 증명(Grant)	1.3.1.3
Coinstack	1.3.2
Coinstack SignOn	1.3.3
주요 모델	1.3.3.1
설치 및 설정	1.4
개요 및 준비사항	1.4.1
Linux에서 설치 및 제거	1.4.2
설치 확인	1.4.3
Coinstack SignOn 시작하기	1.5
도움말 확인	1.5.1
서버	1.5.2
서버 환경 설정	1.5.2.1
서버 코인 충전	1.5.2.2
서버 로그	1.5.2.3
서버 관리	1.5.2.4
클라이언트	1.5.3
사용자	1.5.4
인가 코드 확인	1.5.5
인증 토큰 확인 및 만료	1.5.6
활용	1.6
사용자 인증정보 연동	1.6.1
프로젝트 설정	1.6.1.1
사용자 인증 정보 저장소 구현	1.6.1.2
SignOn 서버와 통합	1.6.1.3
Single Sign On	1.6.2
Servlet 프로젝트 설정	1.6.2.1
Spring 프로젝트 설정	1.6.2.2
리소스 서버 만들기	1.6.2.3
권한 제어	1.6.2.4
시스템 구성	1.6.2.5

사용자 정의 로그인 페이지	1.6.3
환경 구성	1.6.3.1
Nginx 설정	1.6.3.2
Appendix	1.7
오픈 소스 라이센스	1.7.1
설정	1.7.2
명령행 자동 완성	1.7.3
테스트	1.7.4
Postman 테스트	1.7.4.1
기능 테스트	1.7.4.2
부하 테스트	1.7.4.3
성능	1.7.5
알려진 이슈	1.7.6
문자셋 미지정	1.7.6.1

Coinstack SignOn 소개

Coinstack SignOn은 블록체인 기반의 OAuth 2.0 표준을 지원하는 서버와 관리 도구입니다. Coinstack SignOn을 통해 통합 계정, SSO 등의 시스템을 구축할 수 있습니다.

이 문서는 다음과 같은 내용을 기술하고 있습니다.

- OAuth 2.0의 인가 절차
- Coinstack SignOn 서버의 구조
- Coinstack SignOn 서버를 설치하는 방법
- Coinstack SignOn 서버의 디렉터리 및 각 역할
- Coinstack SignOn 관리 명령
- Coinstack SignOn 서버 관리 명령

본 문서에 관하여

Coinstack SignOn에 관한 기본 개념과 설치 방법 그리고 간단한 활용 예제에 대하여 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 전제 조건
- 문서 정보
- 사전 경의
- 저작권 고지
- 연락처

전체 조건

숙지 사항

Coinstack SignOn을 이해하고 사용하기 위해서 다음의 내용을 미리 숙지할 것을 권장합니다.

- OAuth 2.0에 관한 전반적인 이해
- Blockchain에 관한 전반적인 이해
- Coinstack Node의 설치 및 환경 설정

본 문서의 모든 예제와 환경 구성은 Linux 또는 그에 준하는 스타일로 작성되어 있습니다. 따라서 Windows와 같은 환경에서 작업하는 경우 몇 가지 사항을 고려해야 합니다.

예를 들어, 경로 구분자는 Linux에서 "/"를 Windows에서 "\\"로 바꿔서 사용해야 합니다. 또한 환경 변수도 \${VAR} 스타일을 %VAR%로 바꾸어야 합니다.

정의된 항목 및 값

본 문서에서는 다음의 항목에 대해 다음 값을 사용합니다. 개인키와 같은 항목은 다른 값으로 바꿔 사용해야 합니다.

- 서버 포트 - 8080
- 마이너 개인키 - L2F4ENCSqU9Srak3dCd8ySoLHRSvNj7Tm15N9AzEg12TLk9feydV
- 마이너 주소 - 141xLCiRBmAjBxGxRv3YwAvCTkTjYMLWtP
- 관리자 개인키 - KyQxDqRvUYAKnMHN3PvHwQUwpQvtu5Bz2MDARaNBZiTsdTRAaj5e
- 관리자 주소 - 1KkM2NLZEK9SL4sChJVgg45k3godj7Tnix

문서 정보

제목	Coinstack SignOn 안내서
발행일	2018-05-17
소프트웨어 버전	Coinstack SignOn 1.1
문서 버전	1.1

제한 및 한계

OAuth 2.0

본 문서에서 다루는 Coinstack SignOn은 OAuth 2.0 스펙을 준수하지만 OAuth 2.0 스펙에 대하여 자세히 다루지 않습니다. 따라서, 해당 내용은 OAuth 관련 문서를 참고하시기 바랍니다.

Coinstack Node

Coinstack SignOn은 Coinstack Node에 데이터를 쓰거나 읽습니다. Coinstack Node의 설치 및 설정에 관한 내용은 다루지 않습니다. 성능이나 올바른 동작을 위해 필요한 경우에만 그 내용을 언급합니다.

활용 방안

본 문서에서는 Coinstack SignOn을 활용한 예제를 담고 있습니다. 그러나, 실무에 적용 가능한 구체적인 방법이나 관리 및 운영에 관련된 사항은 광범위하므로 구체적으로 다루지 않습니다.

사전 정의

용어

영문	한글
Coinstack	코인스택
Authentication	인증
Authorization	인가
Resource owner	리소스 오너
Client	클라이언트
Access token	액세스 토큰
Endpoint	엔드포인트
Private key	개인키
Public key	공개키

표기 및 의미

표기	의미
\${VAR}	VAR라는 이름의 환경 변수
진하게	출력 또는 화면의 문구를 인용
\$	명령행 프롬프트
#	Root 계정의 명령행 프롬프트
<Ctrl>+C	Ctrl 키를 누른 상태에서 C키를 누름
<USER-INPUT>	사용자 입력
[OPTION]	선택사항

약속된 환경 변수

변수명	의미
INSTALL_PATH	프로그램이 설치된 경로
PROJECT_HOME	개발을 위한 프로젝트 디렉터리

저작권 고지

Copyright Notice

Copyright © 2018 BLOCKO, Inc. All rights reserved.

대한민국 경기도 성남시 분당구 성남대로 331번길 8(정자동) 킨스타워 16층

Restricted Right Legend

All Cointack Software and documents are protected by copyright laws and international convention. Cointack software and documents are made available under the term of the BLOCKOLicense Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of BLOCKO, Inc. Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to BLOCKO trademarks, logos, or any other brand features.

This document is for information purposes only. The company assumes no direct or indirect responsibilities of the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

해당 소프트웨어(Cointack)와 문서들은 저작권법과 국제 협약으로 보호되고 있습니다. 본 문서와 본 문서에서 언급된 Cointack 소프트웨어는 BLOCKO, Inc와의 사용권 계약 하에서만 사용할 수 있으며, 본 문서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 문서의 전체 또는 일부를 BLOCKO, Inc의 사전 서면 동의 없이 어떠한 형식이나 수단(전자, 기계 또는 광학)으로도 전송, 복제, 배포, 2차 가공물 작성 등의 행위를 해서는 안됩니다. 이 문서와 소프트웨어의 사용권 계약은 어떤 경우에도 (등록 여부와 상관없이) 관련된 지적 재산권을 양도하는 것으로 해석되지 않으며, BLOCKO 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다.

이 문서는 오로지 정보 제공만을 목적으로 하며, 이로 인해 계약상의 직접 혹은 간접적 책임을 지지 않으며 문서상의 내용이 법적 또는 상업적인 조건을 보장하지 않습니다. 이 문서에 포함된 정보는 제품 업그레이드나 수정에 따라 예고없이 변경될 수 있습니다. 본사는 이 문서의 내용 상에 오류가 없음을 보장하지 않습니다.

Trademarks

Cointack is registered trademark of BLOCKO, Inc. Other products, titles or services may be registered trademarks of their respective companies

Cointack은 BLOCKO, Inc의 등록 상표입니다. 기타 제품들과 이름 및 서비스는 각 회사에 귀속되어 있습니다.

Open Source Software Notice

각 제품에 사용된 공개 소프트웨어에 관한 정보는 제품 설치 디렉터리의 \${INSTALL_PATH}/doc/licence를 참조하시기 바랍니다.

연락처

BLOCKO, Inc.

331beonkil 8, Bundang-gu

Seongnam-si, Gyeonggi-do, 13558

South Korea

Tel: +82-8016-6253

Fax: +82-8016-6253

Email: support@blocko.io

Web: <https://blocko.io>

개요

Coinstack SignOn에 대한 전반적인 개념을 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- OAuth 2.0
- Coinstack 소개
- Coinstack SignOn의 구조와 동작

OAuth 2.0

OAuth 2.0에 대한 전반적인 개념을 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 개념 및 용어
- 클라이언트
- 인가 증명

개념 및 용어

개념

OAuth 2.0은 애플리케이션이 Facebook, Google, GitHub와 같은 서비스의 사용자 계정에 대한 제한된 액세스 권한을 얻을 수 있게 해주는 인가 프레임워크입니다. 이는 사용자 계정을 호스팅하는 서비스에 사용자 인증을 위임하고 타사 응용 프로그램에 사용자 계정에 대한 액세스 권한을 부여하여 작동합니다. OAuth는 웹 및 데스크톱 응용 프로그램 및 모바일 장치에 대한 인증 흐름을 제공합니다.

Coinstack SignOn은 블록체인을 이용하여 개발되었으며 Facebook, Google, GitHub와 같은 서비스 제공자의 입장에서 OAuth 서비스를 제공할 수 있게 합니다. Coinstack SignOn 플랫폼은 귀사의 계열사, 자회사, 귀사의 고객을 공유하고자 하는 협력사, 신규 육성하는 앱 등에 적용할 수 있습니다. 귀사의 서비스를 이용하는 사용자들에게 다른 서비스에 접근할 기회를 제공하고 귀사의 서비스에 대한 사용을 늘리며 관리할 수 있습니다.

용어

Authentication(인증)

인증은 현재 사용자의 신원을 확인하여 증명이 완료된 사용자에게 리소스 접근을 허용하는 것입니다.

Authorization(인가)

인가는 특정 리소스에 접근할 수 있는 권한을 부여하는 것으로 접근 제어라고 볼 수 있습니다.

Protected Resource

OAuth 절차상에서 접근하려 하는 대상 자원. 웹상의 페이지, 미디어 파일과 같은 것들이 예가 됩니다.

Resource Owner(사용자)

제한된 리소스에 접근을 허용하는 주체로 자신의 계정에 접근하기 위해 **애플리케이션을 인증하는 사용자**입니다. 응용 프로그램의 사용자 계정 액세스 권한은 부여된 권한(예: 읽기 또는 쓰기 권한)의 "범위"로 제한됩니다. 귀사의 서비스에 가입된 각각의 사용자가 리소스 오너가 될 수 있습니다. **사이트를 이용하는 사용자**일 수도 있으며, **앱을 이용하는 사용자**가 될 수도 있습니다.

Authorization Server(인가 서버)

인가 서버는 **리소스 오너의 ID를 검증하는 서버**입니다. 인가 서버는 사용자의 ID를 검증한 이후 액세스 토큰을 응용 프로그램에 발행합니다. 액세스 토큰을 발급받은 클라이언트, 사용자는 해당 액세스 토큰을 리소스에 접근하기 위해 사용합니다.

예를 들어 Facebook OAuth2를 이용하는 경우 인가 서버는 Facebook이 됩니다.

Resource Server (API 서버)

리소스 서버는 제한된 리소스를 제공하는 서버를 의미합니다. 클라이언트와 사용자는 리소스 서버에 접근하기 위해 인가 서버에서 발급받은 액세스 토큰을 사용합니다. 리소스 서버는 액세스 토큰을 검증하여 사용자의 정보에 접근할 권한을 체크합니다.

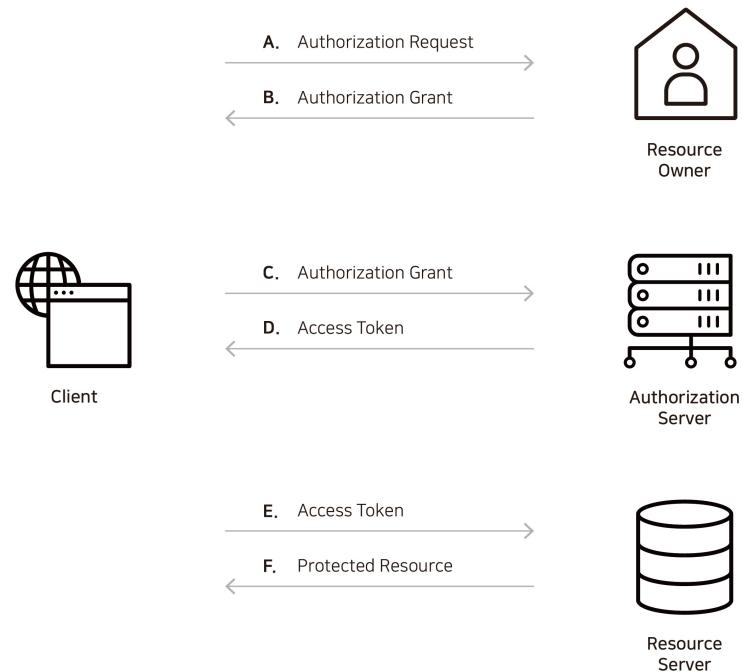
Client(접근 대상)

클라이언트는 사용자의 계정에 액세스하려는 응용 프로그램입니다. 하지만 그 전에 사용자가 권한을 부여받아야 하며 권한의 체크와 승인은 리소스 서버에 의해 확인되어야 합니다.

Protocol Flow

OAuth 2.0 인증 서비스의 흐름은 다음과 같습니다.

Abstract Protocol Flow



- A. 클라이언트는 리소스 오너에게 제한된 리소스의 접근 허가를 요청합니다.
- B. 리소스 오너는 이를 허가합니다.
- C. 클라이언트는 인가 서버에 리소스 오너의 허가를 증명합니다.
- D. 인가 서버는 클라이언트의 증명에 문제가 없음을 증명하는 액세스 토큰(Access token)을 발행합니다.
- E. 액세스 토큰을 리소스 서버에 보내서 리소스를 요청합니다.
- F. 리소스 서버는 리소스를 반환합니다.

클라이언트(Client)

OAuth 2.0의 가장 큰 특징은 다양한 상황에 대해서 사용할 수 있는 인가 프레임워크라는 점입니다. 이를 위해 클라이언트의 종류를 분류하고, 이에 따른 다양한 인가 증명 방법을 수행합니다.

클라이언트의 종류

클라이언트는 크게 클라이언트 상에 자격 증명 수단을 유지해도 안전한지에 따라 다음과 같이 분류됩니다.

confidential

자격 증명을 위한 값을 유지해도 안전한 클라이언트. 자격 증명에 제한적으로 접근할 수 있도록 설계된 애플리케이션.

public

자격 증명을 위한 값을 유지하면 안 되는 클라이언트. 브라우저, 모바일 앱 등.

수용 가능한 클라이언트

OAuth 2.0은 제안 시점에 다음과 같은 클라이언트들을 염두에 두고 설계되었습니다.

Web application

웹서버 상에서 실행되는 웹 애플리케이션은 **confidential** 클라이언트입니다. 리소스 오너는 브라우저(User-agent)를 통해 표현되는 웹 페이지를 통해 클라이언트에 접근할 수 있습니다. 클라이언트 자격 증명뿐만 아니라 발행되는 모든 토큰을 서버에 저장할 수 있습니다.

User-agent-based application

브라우저에서 실행되는 SPA(Single Page Application) 같은 애플리케이션은 **public** 클라이언트입니다. 서버로부터 코드가 다운로드되어서 실행되고 실행 데이터와 결과가 브라우저상에 존재하기 때문에 손쉽게 접근할 수 있습니다.

Native application

사용자의 컴퓨터나 모바일 장치에 설치되어서 단독으로 실행되는 애플리케이션은 **public** 클라이언트입니다. 프로토콜 데이터와 자격 증명 값에 접근할 수 있기 때문에 문제가 됩니다.

인가 증명(Grant)

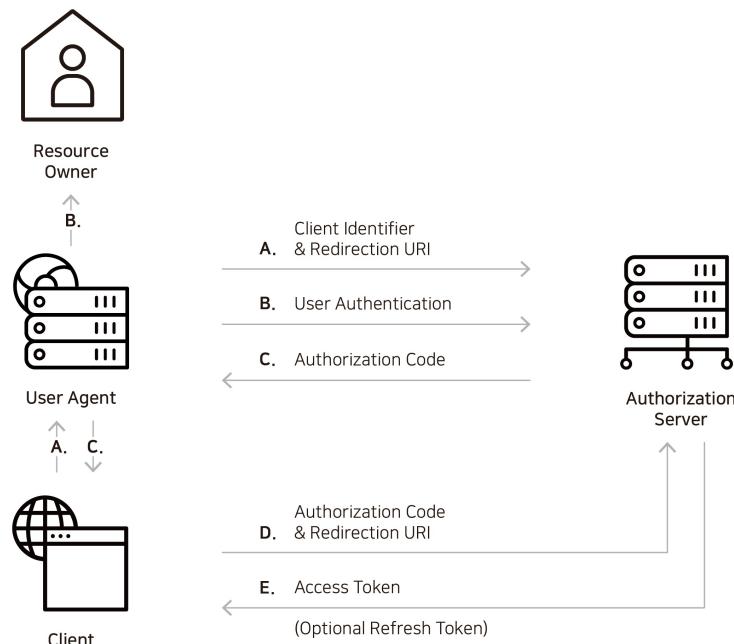
OAuth 2.0의 인가 증명을 위한 좋은 방법을 제안하고 있을 뿐만 아니라 표준안에 포함되어 있지 않은 더 많은 활용예제가 존재합니다.

표준 인가 증명

인가 코드(Authorization Code)

액세스 토큰(Access token)을 얻기 위해 사용되며, **confidential** 클라이언트에 최적화된 방법입니다. Redirection에 기반을 둔 절차이기 때문에 클라이언트는 사용자가 조작하는 프로그램(Browser)과 상호 작용할 수 있어야 합니다.

Authorization Code Flow

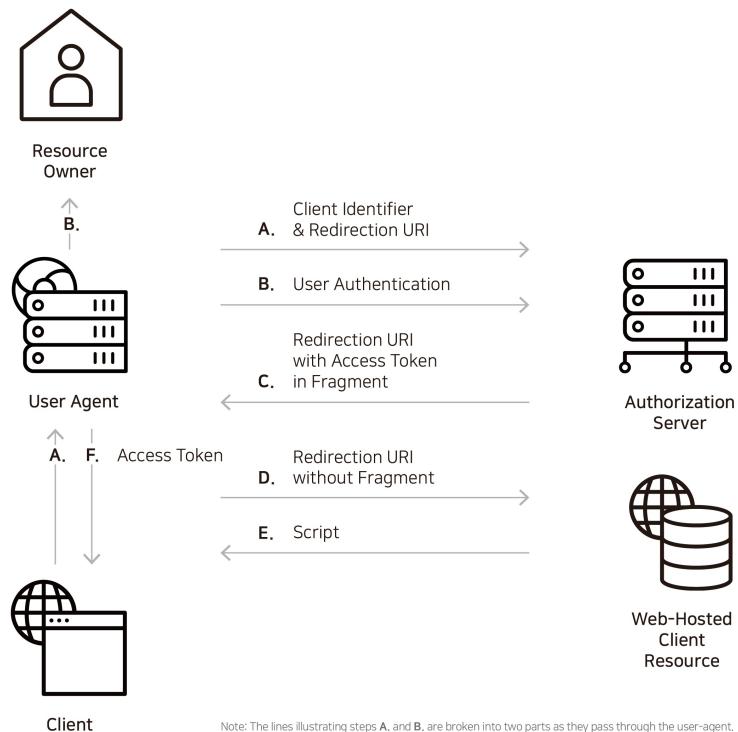


- A. User agent는 클라이언트 ID와 리다이렉트 할 클라이언트의 URI를 인가 서버에 전달합니다.
- B. 사용자 인증을 인가 서버에게 요청합니다.
- C. 인가 서버는 인증에 성공하면 인가 코드를 리다이렉트 URI에 포함해서 리다이렉트합니다.
- D. 인가 코드를 받은 클라이언트는 인가 코드를 A 번의 리다이렉트 URI에 포함해서 액세스 토큰을 요청합니다.
- E. 인가 서버는 액세스 토큰을 발행합니다.

암묵적 증명(Implicit)

인가 코드 방식에서 인가 코드 발행 절차를 생략한 방식입니다. 리프레시 토큰은 발행되지 않으며, **public** 클라이언트에 최적화된 절차입니다.

Implicit Flow

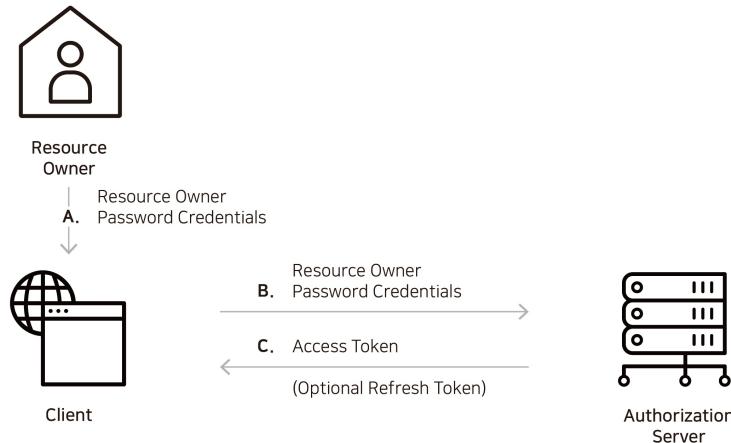


- A. User agent는 클라이언트 ID와 리다이렉트 할 클라이언트의 URI를 인가 서버에 전달합니다.
- B. 사용자 인증을 인가 서버에게 요청합니다.
- C. 인가 서버는 인증에 성공하면 액세스 토큰을 포함한 리다이렉트 URI를 Fragment에 포함해서 리다이렉트합니다.
- D. 액세스 토큰을 받은 User agent는 Fragment를 제거한 URI를 요청합니다.
- E. 액세스 토큰을 추출하기 위한 스크립트를 전달합니다.
- F. User agent는 스크립트를 통해 클라이언트에 액세스 토큰을 전달합니다.

패스워드 방식>Password)

리소스 오너와 클라이언트에 신뢰가 있을 때 사용하는 방식입니다. 장치의 OS나 관리자 권한으로 실행된 애플리케이션, 또는 클라이언트가 인가 서버가 같은 시스템인 경우에 적합합니다.

Resource Owner Password Credentials Flow



A. 리소스 오너는 패스워드 자격 증명을 클라이언트에 전달합니다.

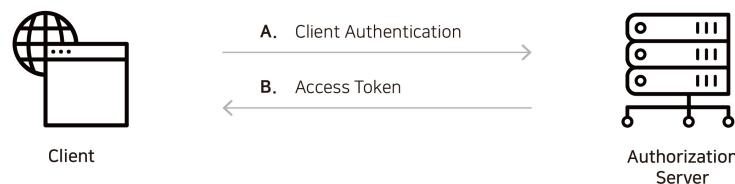
B. 클라이언트는 이 자격 증명을 인가 서버에 전달합니다.

C. 인가 서버는 액세스 토큰을 발행합니다.

클라이언트 자격 증명(Client Credentials)

클라이언트 자격 증명만으로 액세스 토큰을 발행할 수 있는 방식입니다. **confidential** 클라이언트만 사용해야 합니다.

Client Credentials Flow



A. 클라이언트 인증을 인가 서버에 전달합니다.

B. 인가 서버는 액세스 토큰을 발행합니다.

인가 증명 확장

표준안에 포함되지 않은 방식이더라도 grant_type 인자값을 URI로 지정해서 사용할 수 있습니다. 아래의 예시 외에 수많은 확장 예제가 존재합니다.

SAML 2.0 Bearer Assertion Profiles for OAuth 2.0

SAML 2.0 을 사용하여 인증/인가를 하는 표준입니다. 복잡한 규칙의 인증/인가 절차를 적용할 수 있습니다.

자세한 내용은 <https://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-09>를 확인해 보시기 바랍니다.

장치 코드

플레이스테이션이나 YouTube 플레이어 같은 장치상에서 애플리케이션(또는 서비스)으로 접근할 때 사용하는 인가증명 방법입니다.

자세한 내용은 <https://alexabilbie.com/2016/04/oauth-2-device-flow-grant/>를 확인해 보시기 바랍니다.

Cinstack

블록체인은 비트코인을 필두로 한 암호화폐(Cryptocurrency)의 핵심 기술로 시작되었습니다. 이후 비가역성과 투명성, 무결성 등의 장점을 살려 다양한 분야에서 주목받기 시작했으며, 꾸준한 개발을 통해 성능 향상과 확장성이 극대화되고 있습니다.

현재 블록체인 기술은 기존 암호화폐 분야를 넘어, 최고 수준의 데이터 안전성과 신뢰성이 요구되는 금융, 의료, 관공서 등의 분야에서 활용되거나 적용 검토 중입니다. 점점 더 다양한 분야에서 블록체인을 기반으로 한 참신한 서비스들도 하나둘 등장하고 있고, 물류나 보험을 비롯한 여러 이해관계자가 참여하는 복잡한 시장에서 특히 큰 파급력을 선보일 수 있을 거라 기대되고 있습니다.

하지만, 블록체인 기술을 활용할 획기적인 아이디어가 있다고 해도 높은 기술장벽으로 인해 많은 시간과 비용이 들어갔습니다. 현재 오픈 소스로 구현되는 대부분의 블록체인 기술이 암호화폐와 그 생태계를 지탱하기 위해 만들어졌기 때문에 범용성이 떨어지는 문제 또한 있었습니다. 특히, △하드포크 △가격 변동성 △특정 개인 혹은 법인에 대한 종속성을 비롯한 퍼블릭 블록체인의 단점으로 인해 아직 금융 등의 특정 분야에서만 활용되고 있습니다.

블로코는 코인스택을 통해 이러한 단점을 해결하고 더 많은 분야에서 블록체인 기술이 활용될 수 있도록 기존 서비스와 블록체인을 연결해주는 징검다리 역할을 하고자 합니다.

코인스택은 기업과 개인 누구나, 간편하게 블록체인 기반서비스를 개발할 수 있도록 도와주는 플랫폼입니다. 실제로 블로코에서 서비스 중인 [클라우드스탬프](#), [WATCH](#) 모두 코인스택 기반으로 개발되었으며, 현재 제1금융권 은행 및 대형 카드사를 비롯한 대기업, 한국거래소 등 다양한 기업과 기관에서 블록체인 기반 서비스를 제공하기 위해 코인스택을 사용하고 있습니다. 블로코가 무료로 제공하는 클라우드 기반 코인스택 서비스는 과제를 수행하는 학생들, 개인 개발자들이 편리하게 이용하고 있습니다.

코인스택은 많은 개발자에게 친숙한 프로그래밍 언어인 자바, 자바스크립트, HTML5 기반 SDK를 제공하며, HTML5 기반의 [웹플레이그라운드](#), [검색엔진](#), [대시보드](#) 등을 통해 편리하고 익숙한 개발 환경을 제공합니다.

블록체인. 더 이상 복잡하고 장벽 높은 ‘꿈의 기술’이 아닙니다. 블로코가 ‘현실의 기술’로 만들어 나가겠습니다.

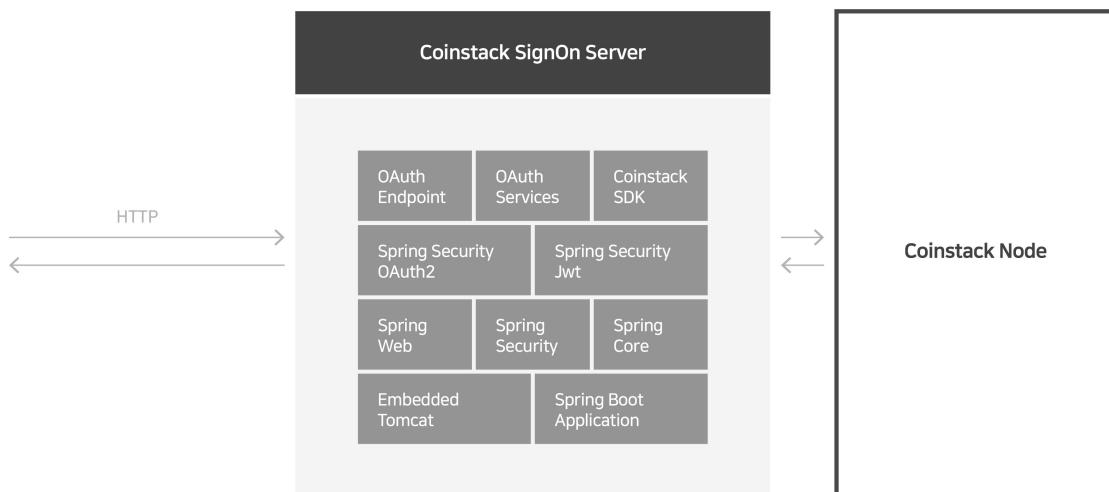
Coinstack SignOn

Coinstack SignOn은 블록체인 기반의 사인-온(Blockchain-based Sign On)의 개념을 구현한 구현체입니다. 사인-온 절차의 이해와 타 시스템과의 연계 편의성을 위해 표준인 OAuth 2.0 프로토콜을 준수하며, 공유된 원장이라는 블록체인의 특성을 잘 활용한 제품입니다. 외부의 침입으로 인해 서버가 노출되어도 서버를 관리하는 개인키만 탈취되지 않으면 시스템을 마음대로 변조할 수 없어 뛰어난 보안성을 가집니다.

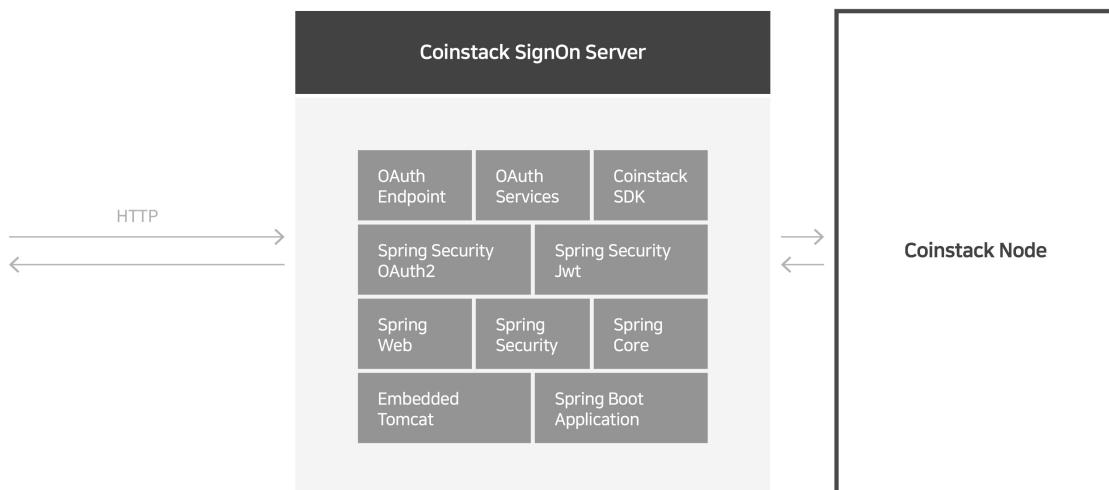
서버 구조

Coinstack SignOn 서버는 Spring Boot 2.0기반으로 구현되어 있습니다. Spring Security OAuth2의 엔드포인트와 서비스들을 활용하고 있습니다.

Server Architecture

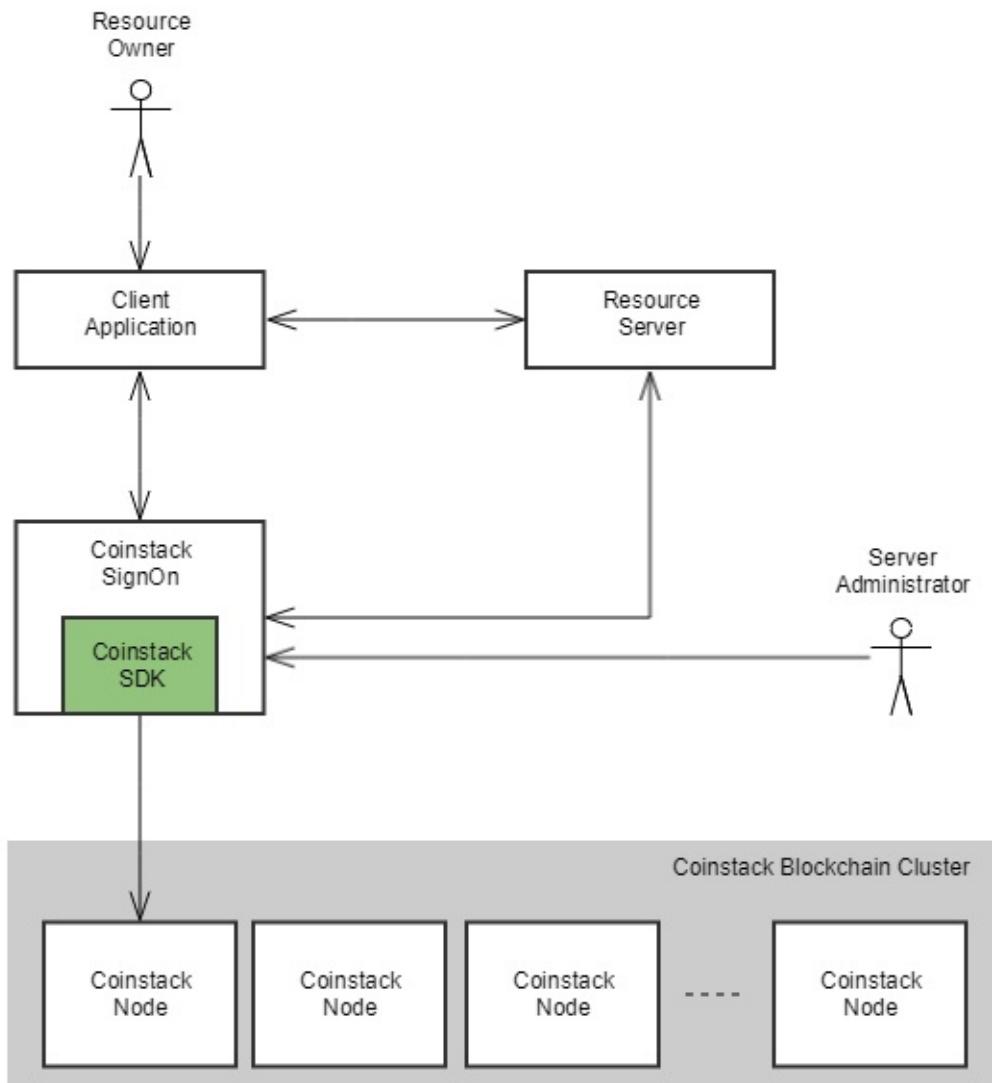


Server Architecture



인증 및 확인 절차

다음은 Coinstack SignOn의 인증 및 확인 절차입니다.



관리자 어드레스

관리자 어드레스는 클라이언트(Client), 사용자(User)를 관리하는 어드레스입니다.

Admin Address Transaction Form

	SENDER	RECEIVER	OP_RETURN
1	1AUy...	1AUy...	{"clientId":"trusted", "clientSecret":"3E38BD30361AA8556...", ...}
2	1AUy...	1Kkm...	{"clientId":"trusted", "clientSecret":"3E38BD30361AA8556...", ...}
3	14d4...	1AUy...	{"clientId":"attack", "clientSecret":"62FEF6A25FE97BF52...", ...}
			...
			...
			...
			...

※ 마이닝 어드레스를 사용하는 경우 UTXO가 많이 발생하여 부하가 생길 수 있습니다.

- 사용자 등록 시 관리자 어드레스는 2개의 트랜잭션을 발생시킵니다.
- **수신자가 관리자 어드레스인 트랜잭션**
 - 관리자 어드레스에 클라이언트 정보를 기록함으로써 리스트 관리가 가능합니다.
- **수신자가 clientId로 추출한 어드레스인 트랜잭션**
 - clientId로 추출한 어드레스에 클라이언트 정보를 기록함으로써 개별 관리가 가능합니다.
- 타 유저가 임의의 클라이언트 데이터를 등록한 경우
 - 블록체인에서 데이터를 조회할 때 **Input 값이 관리자 어드레스인 데이터들을 유효하다고 판단하기 때문에 관리자가 아닌 타 어드레스에서 등록한 클라이언트 정보들은 적용되지 않습니다.**

※ 클라이언트의 경우 **수가 많지 않아** 블록체인에서 관리하지만, User의 경우 많게는 수만 개 이상의 데이터를 보유하기 때문에 블록체인으로 따로 관리하지 않습니다. 이에 대한 자세한 사항은 [사용자 인증 연동하기](#)장을 참조합니다.

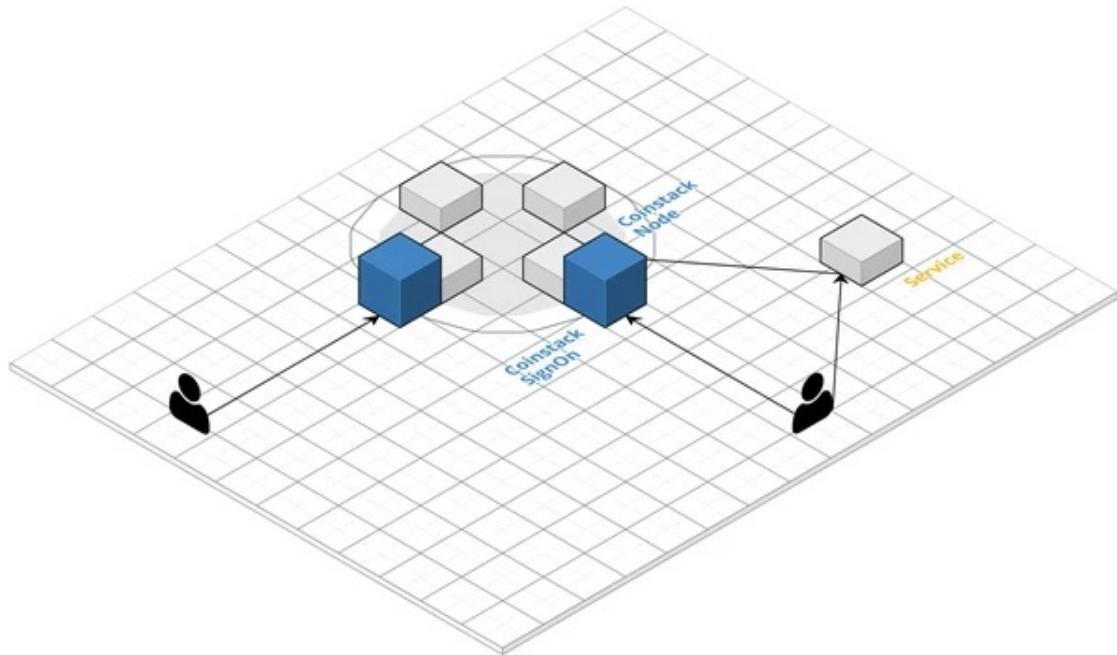
서버 개인키

서버 개인키는 인가 코드(Authorization code), 액세스 토큰(Access token)을 생성하는데 사용되는 개인키입니다. 서버 개인키는 하나의 서버에만 할당해야 하며, 서버는 여러 개인키를 가질 수 있습니다. 즉 서버 vs 개인키는 1:N의 관계를 맺습니다. 인가 코드와 액세스 토큰과 관련된 거래는 서버 어드레스나 관리자 어드레스에서 나온 거래에 대해서만 유효성이 인정됩니다. 따라서, 임의의 공격자가 인가 코드를 발행하거나 발행된 액세스 토큰을 무효화(expire)할 수 없습니다.

※ 마이닝 어드레스를 사용하는 경우 UTXO가 많이 발생하여 부하가 생길 수 있습니다.

서버 피어 주소

여러 Cinstack SignOn로 구성된 시스템이라면, 하나의 서버가 발행한 액세스 토큰을 다른 서버에서도 인지할 수 있어야 합니다. 따라서, 서버별로 개인키들을 따로 지정하는 것과 동시에 이러한 개인키의 주소들을 모두 알고 있어야합니다.



Endpoint

Cinstack SignOn 서버에서 제공하는 OAuth 2.0 인증 서비스의 엔드포인트는 다음과 같습니다.

Path	API 설명	비고
/oauth/authorize	클라이언트의 정보를 받아 접근 권한을 확인해주는 엔드포인트	
/oauth/token	액세스 토큰을 발급해주는 엔드포인트	
/oauth/check_token	액세스 토큰을 확인해주는 엔드포인트	
/oauth/logout	로그아웃해주는 엔드포인트	
/oauth/error	에러 발생 내용을 보여주는 엔드포인트	

/oauth/authorize

클라이언트의 정보를 받아 접근 권한을 확인해주는 엔드포인트

Query Parameters

Parameter	Description	Values
response_type	사용할 권한 부여 처리 과정을 결정 Authorization Code, Implicit 방식에서 사용	String · code · token
grant_type	인가 증명 방식을 결정하기 위해 사용	String · authorization_code · implicit · password · client_credentials
scope	클라이언트에서 사용자에게 허가를 요구하는 범위	
client_id	클라이언트 식별자	
secret	클라이언트 비밀번호	
redirect_uri	SignOn 서버에서 인증/인가 후 반환되는 URI	

Response

Parameter	Description
200	OK
302	Redirection
401	Bad Client Credentials
500	Error

/oauth/token

액세스 토큰을 발급해주는 엔드포인트

Query Parameters

Parameter	Description	Values
grant_type	인가 증명 방식을 결정하기 위해 사용	String · authorization_code · password · client_credentials · refresh_token
redirect_uri	SignOn 서버에서 인증/인가 후 반환되는 URI	
code	인가 증명 방식이 Authorization Code 방식일 경우 사용	
username	인가 증명 방식이 Password 방식일 경우 사용	
password	인가 증명 방식이 Password 방식일 경우 사용	
refresh_token	인가 증명 방식이 Refresh Token 방식일 경우 사용	

Response

Parameter	Description	Values
access_token	액세스 토큰값	
refresh_token	리프레시 토큰값	
scope	클라이언트에서 사용자에게 허가를 요구하는 범위	
token_type	액세스 토큰의 타입	String · bearer
expires_in	액세스 토큰 만료 시간	

/oauth/check_token

액세스 토큰을 확인해주는 엔드포인트

Query Parameters

Parameter	Description	Values
token	확인할 액세스 토큰값	

Response

Parameter	Description
user_name	사용자 식별자
scope	클라이언트에서 사용자에게 허가를 요구하는 범위
authorities	사용자 접근 권한
client_id	클라이언트 식별자

/oauth/logout

로그아웃을 하고 accessToken을 만기하는 엔드포인트

access_token, redirect_uri 인자는 필수적이며 로그아웃이 완료되면 \${REDIRECT_URI}로 리다이렉션됩니다.

Query Parameters

Parameter	Description	Values
access_token	만료할 액세스 토큰값	
redirect_uri	사용 중인 redirect_uri	

/oauth/confirm_access

userApprovalPage로 리다이렉션시켜주는 엔드포인트

사용자가 허가의 승인을 확인하는 POST 요청을 받습니다.

/oauth/error

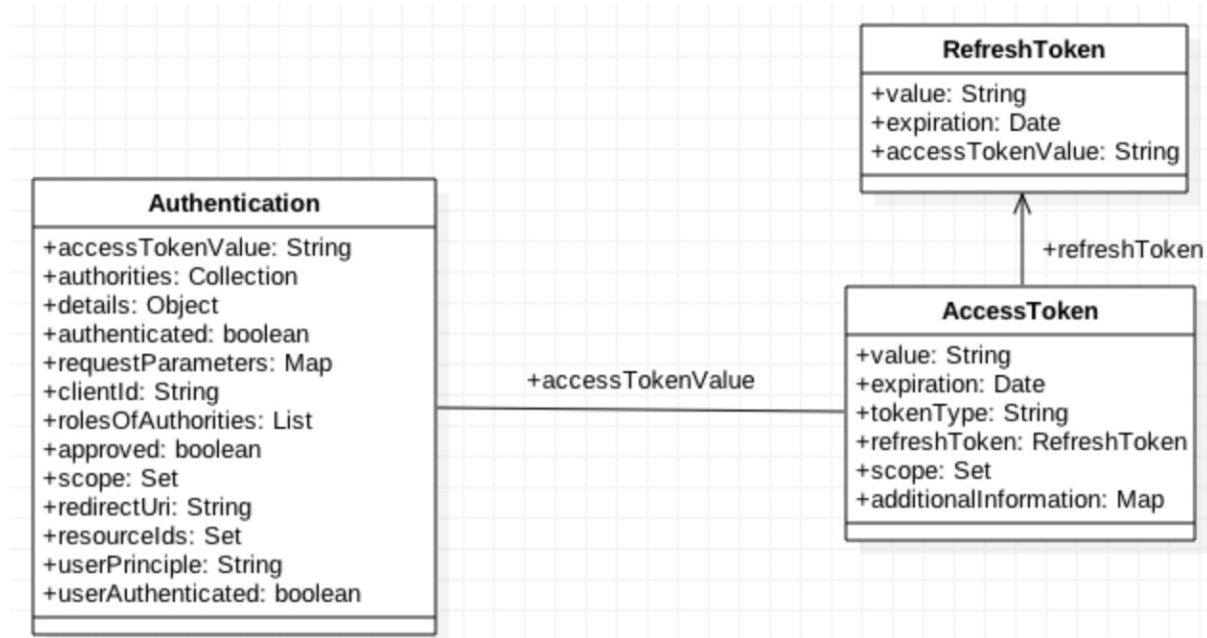
에러 발생 내용을 보여주는 엔드포인트

주요 모델

논리 모델

SignOn에서는 AccessToken과 관련된 모델이 가장 중요하고 빈번하게 사용됩니다.

관련 모델은 다음과 같습니다.



이 모델을 이해하기 위해서 현실에서 일어나는 일을 예로 들어보겠습니다.

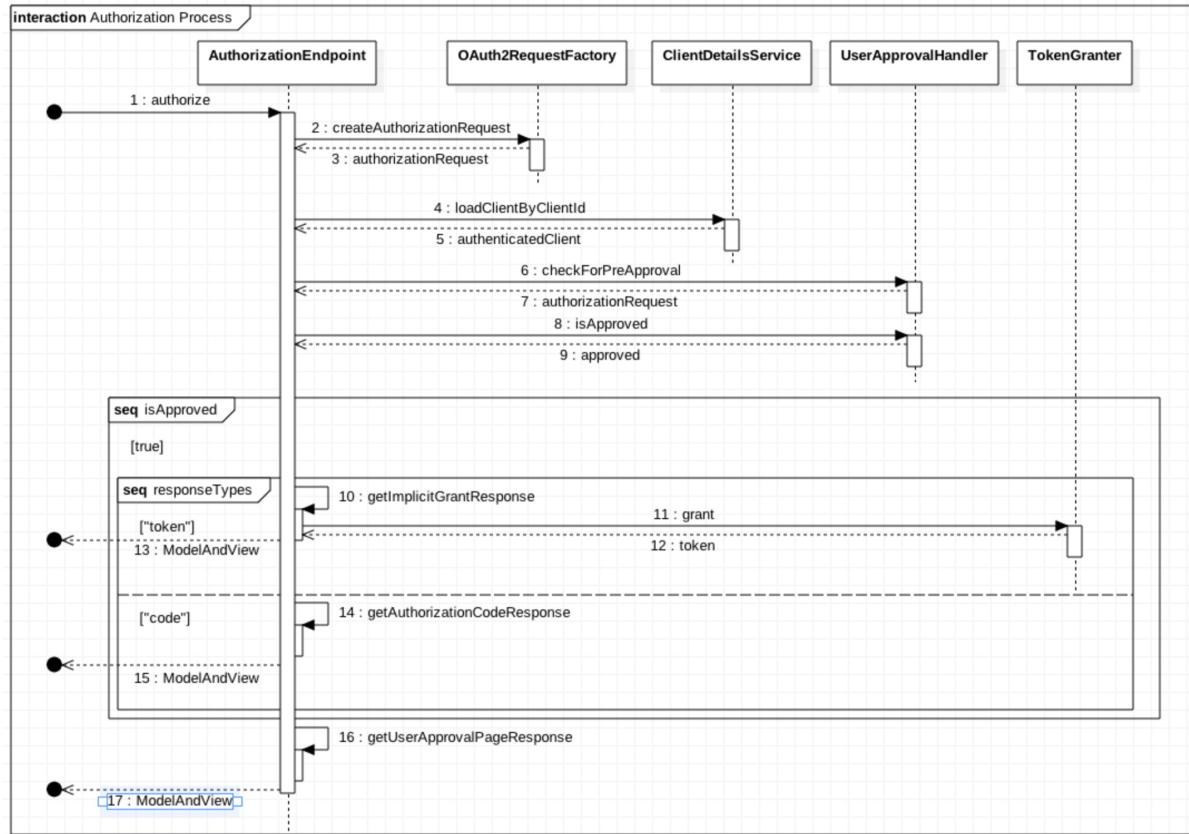
당신이 인터넷으로 영화를 예매하고, 극장에 가서 입장권을 발매하여 입장하는 과정은 우리가 제공하는 사인온행위와 매우 닮아 있습니다. 입장권을 발매하기 위해 당신은 신분증(Authentication)을 제시합니다. 입장권 발매자는 신분증을 확인하고 입장권(Access Token)을 발행하게 됩니다. 극장의 앞에서는 표를 확인하는 직원이 입장통제(Access Control)합니다. 만일, 표를 잃어버렸을 때는 다시 신분증을 가지고 가서 표를 재발행하거나 기존의 표가 동작하지 않도록 하는 작업을 해야 합니다. 이러한 이유로 Authentication 정보에 Access Token의 값이 있어야 합니다.

이 경우, 신분증을 통한 작업의 경우 표를 다시 발행하는 행위인데 이 과정을 좀 더 손쉽게 하기 위한 방법으로 리프레쉬 토큰을 제공합니다. 이것은 신분증보다는 구매 번호 또는 예약 번호에 비유할 수 있습니다. 신분 인증없이 액세스 토큰을 재발행할 수 있도록 합니다.

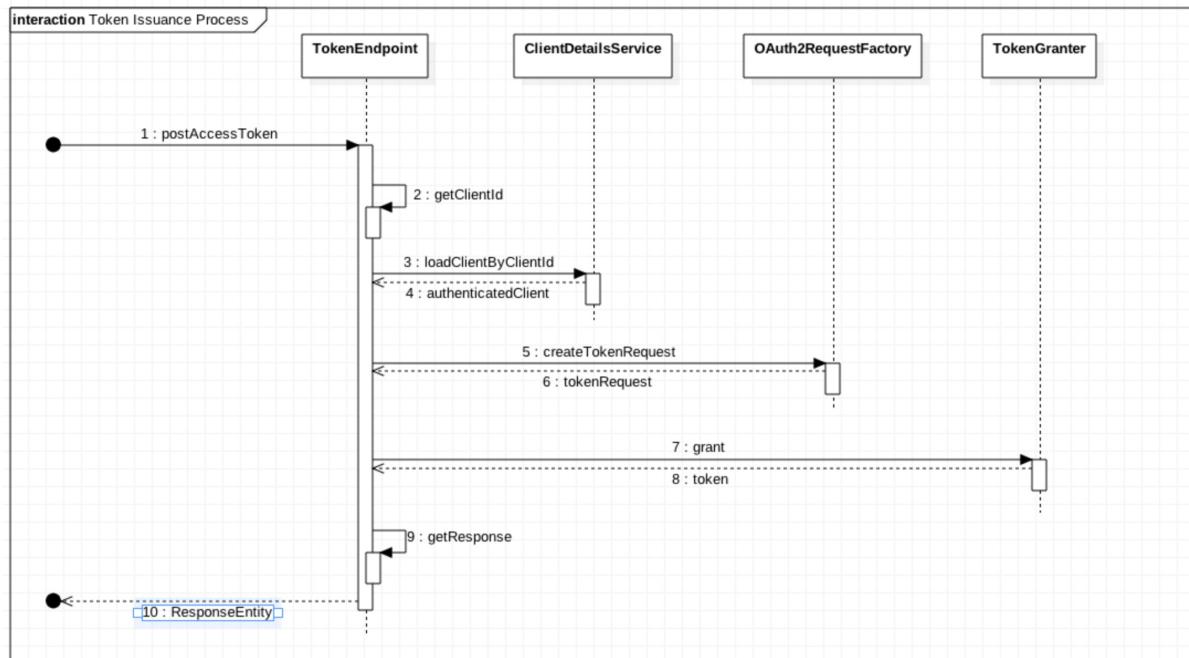
이러한 이유로 액세스 토큰을 발행하기 위해서는 인증 정보와 리프레쉬 토큰을 함께 기록해야 합니다.

Endpoint별 동작

AuthorizationEndpoint

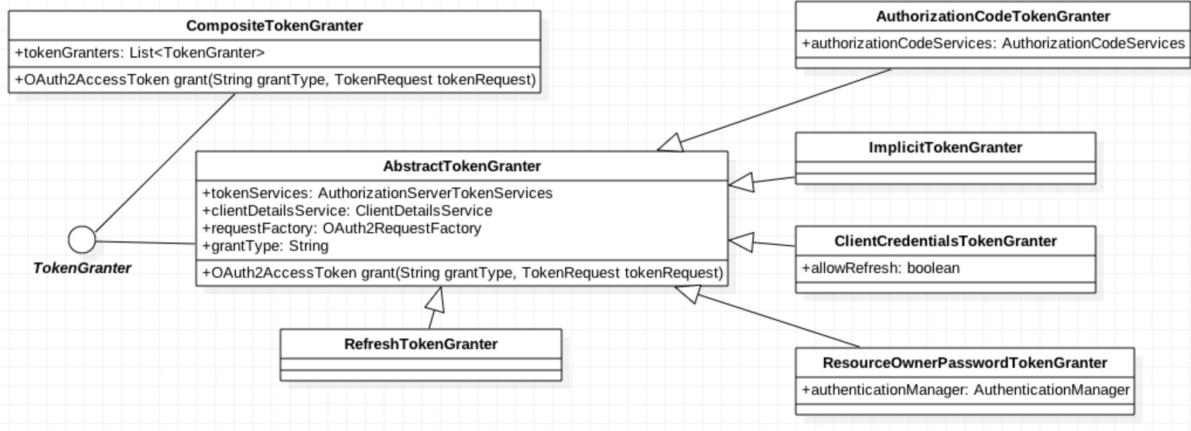


TokenEndpoint



해당 요청들은 최종적으로 모두 TokenGranter를 통해 token을 발행하도록 요청합니다. 그 과정에서 Authorization code를 발행하거나 사용자의 인가를 받기도 하지만 최종 목표는 토큰을 발행받는 것이기 때문에 이는 당연합니다.

TokenGranter



TokenGranter를 구현한 클래스들입니다. 이 클래스에서 사용하는 인터페이스와 그 구현체는 다음과 같습니다.

인터페이스	제공되는 구현체
AuthorizationServerTokenServices	coinstack.signon.server.service.CoinstackTokenService
ClientDetailsService	coinstack.signon.server.service.CoinstackClientService
AuthorizationCodeServices	coinstack.signon.server.service.CoinstackAuthorizationCodeService

설치 및 설정

Coinstack SignOn의 설치 및 설정에 관해 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 개요 및 준비사항
- Linux에서 설치 및 제거
- 설치 확인

개요 및 준비사항

본 장에서는 설치 과정에 대한 전반적인 설명과 설치에 필요한 시스템 환경에 관해서 설명합니다.

개요

Linux 환경에서 Coinstack SignOn을 쉽게 설치하기 위해서 인스톨러를 제공합니다. 인스톨러를 구동하기 위해서는 unzip 명령이 필요합니다. 만일, unzip을 사용할 수 없다면 tar.gz의 형태로 요청하시기 바랍니다.

설치 파일은 다음의 작업을 진행합니다.

- 지정한 위치 또는 미리 정의된 위치에 Coinstack SignOn 구성 파일들을 배치합니다.
- 실행에 필요한 환경 변수를 설정합니다.

시스템 요구사항

하드웨어

최소 1GB의 설치공간 - 로그 파일을 위한 저장공간은 따로 확보되어야 합니다.

최소 2GB의 메모리 - 서버 내의 캐시를 위해 추가적으로 메모리가 있어야 합니다.

Ethernet interface - 클라이언트의 HTTP 요청과 Coinstack Node 연동을 위해 필수적입니다.

운영체제 및 환경

Linux, Windows, Mac 등 - Centos7, Ubuntu 16.04LTS에서 테스트 됩니다.

Bash - 서버 구동 및 정지, 관리 등 다양한 명령이 Bash 기반의 스크립트로 제공됩니다.

Java Runtime Environment v1.8+(혹은 Docker 17.x+) - 프로그램 구동을 위해 필요합니다.

Linux에서 설치 및 제거

설치 배포판

설치 배포판은 다음의 형태로 얻을 수 있습니다.

- *.tar.gz
- *.zip
- *.bin

본 문서에서는 *.bin을 사용하여 설치하는 방법에 대해서 설명합니다. *.zip/*.tar.gz는 *.bin를 사용할 수 없는 제한된 환경에서 설치할 수 있도록 제공되는 파일입니다. 해당 배포판은 다음의 위치에서 얻을 수 있습니다.

버전	URL
v1.1	https://nexus.blocko.io/repository/blocko-product-repository/coinstack-signon-1.1-installer.bin

설치 배포판 가져오기

설치 배포판은 nexus.blocko.io 서버에 저장되어 있습니다. 배포판은 curl이나 wget 명령으로 가져올 수 있습니다.

curl의 경우

```
$ curl -o coinstack-signon-installer.bin \
  https://nexus.blocko.io/repository/blocko-product-repository/coinstack-signon-1.1-installer.bin
```

wget의 경우

```
$ wget -O coinstack-signon-installer.bin \
  https://nexus.blocko.io/repository/blocko-product-repository/coinstack-signon-1.1-installer.bin
```

설치 준비

설치를 위해서는 unzip 명령이 필요합니다. 상황이 여의치 않아 unzip을 사용할 수 없다면, tar.gz 배포판을 준비하는 것이 좋습니다.

unzip 설치

설치 프로그램은 unzip 명령이 사용 가능하지 확인하고 필요하면 unzip 설치를 자동으로 진행하지만, unzip 설치를 위해서는 관리자 권한(sudo)이 필요합니다. 만일, 설치 계정이 관리 권한을 갖지 않는다면, 미리 unzip을 설치해야만 원활히 진행할 수 있습니다.

redhat 계열

```
# yum install -y unzip
```

debian 계열

```
# apt-get install -y unzip
```

그 외 리눅스 시스템에 대해서는 해당 관리자에게 설치 문의/요청하시기 바랍니다.

설치

설치 프로그램 실행하기

프로그램의 설치는 배포판을 실행함으로써 수행됩니다. 따라서, 해당 파일에 실행 권한을 주어야 합니다.

실행 권한 주기

```
$ chmod +x coinstack-signon-installer.bin
```

설치 프로그램 실행

```
$ coinstack-signon-installer.bin
```

명령행 자동완성 스크립트 추가

명령행 자동완성 스크립트는 `~/.bash_completion.d/coinstack-signon.completion` 안에 설정됩니다.

이를 실행되게 하기 위해서는 다음의 명령어를 `~/.bash_completion`안에 추가하여야 합니다.

```
for bcfile in ~/.bash_completion.d/* ; do . $bcfile; done
```

위의 과정이 끝나면 다음의 명령어를 실행하여, bash를 재실행합니다.

```
$ exec bash
```

제거

프로그램의 제거하기 위해서는 해당 디렉토리를 삭제하기만 하면 됩니다.

```
$ rm -rf $INSTALL_PATH
```

설치 확인

정상적으로 설치됐다면 다음과 같은 메시지와 함께, `coinstack-signon-$\{VERSION}`의 형태의 디렉터리가 생성된 것을 확인할 수 있습니다.

설치 디렉터리

설치 디렉터리 목록은 다음과 같습니다.

`coinstack.yaml` 파일은 설치 후에는 존재하지 않으나, 서버 설정에서 생성합니다.

설정 방법은 서버 환경 설정을 참조하시기 바랍니다.

파일명	용도
bin	실행 명령들이 있는 디렉터리
└ coinstack-signon	노드 및 사용자 관리
└ coinstack-function-test	기능 테스트
└ coinstack-load-test	부하 테스트
conf	설정 파일들이 있는 디렉터리
└ coinstack.yaml	노드의 어드레스, 관리자의 어드레스, 서버의 개인키 등이 설정된 파일
└ [jwt.yaml]	사용자가 정의가능한 포맷
└ actuator.yaml	Spring boot actuator 관련 설정
doc	관련 문서를 모아놓은 디렉터리

lib	실행을 위한 라이브러리를 위한 디렉터리
log	실행 로그가 쌓이는 디렉터리

YAML

데이터 직렬화 양식으로 주로 XML과 같이 설정 파일 역할을 담당하는 언어로,

Coinstack SignOn 서버를 구동하기 위해 필요한 설정들을 담고 있습니다.

이 언어에 대한 자세한 사항은 <http://yaml.org/>을 참조하시기 바랍니다.

작성 방법

다음과 같이 들여쓰기에 유의하며 작성합니다.

```
coinstack:
  endpoint: http://localhost:3000

signon:
  admin:
    address: 1KkM2NLZEK9SL4sChJVgg45k3godj7Tnix
  server:
    port: 8080
    privatekeys: |
      L46DA9eQzqEaQGe8bQh3YTpwCaokLsiTEMXvaqbUkfGwY47so24J
```

Coinstack SignOn 시작하기

Coinstack SignOn을 사용해서 OAuth 2.0 시스템을 구축하기 위한 전반적인 개념 및 방법을 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 도움말 확인
- 서버
- 클라이언트
- 사용자
- 인가 코드 확인
- 인증 토큰 확인 및 만료

도움말 확인

Coinstack SignOn은 OAuth 2.0 인증 서비스의 실행 가능한 명령어 목록을 확인할 수 있는 도움말 확인 명령어를 제공합니다.

실행 가능한 명령어들의 자세한 사항은 [Coinstack SignOn 시작하기](#)에 포함되어 있습니다.

실행 가능한 명령어 목록 확인하기

사용자는 실행 가능한 명령어 목록을 확인하기 위해 다음과 같은 명령어를 입력합니다.

```
$ coinstack-signon help
```

명령어를 입력하면 다음과 같은 화면이 출력되며, 실행 가능한 명령어 목록들을 확인할 수 있습니다.

```
8story8ui-MBP:bin 8story8$ ./coinstack-signon help
usage: coinstack-signon <command> [<subcommand>] [<option>*] [<arg>*]

These are various coinstack sign-on cli:
help
    Print the help out - you can see this.

client
    create      Create a client
    list        List clients
    check       Check client and show details
    remove     Remove client

code
    check       Check a authorization code status

coin
    recharge   Recharge coin to server private keys

key
    create     Generate a privatekey

server
    check      Check a server status
    configure  Configure an sign-on server setting
    log        Print log out
    monitor    Monitor server metric
    run        Run an sign-on server
    start      Run an sign-on server as daemon
    stop       Stop a running sign-on server

token
    check      Check an access token
    remove    Remove an access token

user
    create    Create a user
    check     Check user and show details
    remove   Remove user

8story8ui-MBP:bin 8story8$
```

상세 도움말

명령 카테고리별로 상세한 도움말을 제공하고 있습니다. 다음과 같이 명령을 입력해 보시면 더 쉽게 이해할 수 있습니다.

```
$ coinstack-signon help code
```


서버

Coinstack SignOn 서버를 제어하기 위한 전반적인 개념 및 방법을 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 서버 환경 설정
- 서버 코인 충전
- 서버 로그
- 서버 관리

서버 환경 설정

본 장에서는 Cointack SignOn의 환경 설정에 관한 전반적인 내용을 설명합니다.

Cointack SignOn 서버를 설정하는 방법은 다음과 같습니다.

Cointack Node 연동 설정

Cointack SignOn는 Cointack Node에 데이터를 저장하고, 읽어서 인증 및 권한 관련 기능을 수행합니다. 사용자는 Cointack SignOn 서버에서 사용할 Cointack Node와 관련된 정보를 설정해야 합니다.

대화형 명령을 통해 설정하기

Cointack SignOn는 사용자가 손쉽게 Cointack Node와 연동할 수 있도록 대화형 명령 스크립트를 제공합니다. 대화형(interactive) 명령을 통한 설정은 다음과 같은 순서로 진행할 수 있습니다.

명령 실행

다음 명령을 수행하면 대화형 모드를 시작합니다.

```
$ coinstack-signon server configure
```

Cointack Node의 Endpoint 입력

```
Input the coinstack node endpoint [http://localhost:3000]
>
```

서버의 포트 입력

```
Input the server port [8080]
>
```

관리자의 주소 입력

```
Input the administrator's address [1KKM2NLZEK9SL4sChJVgg45k3godj7Tnix]
>
```

서버의 개인키 생성

서버의 개인키는 서버의 성능과 관련되어 있습니다. 병렬 처리를 위해 여러 개의 키를 사용하기 때문에 성능이 높은 서버일수록 더 많은 개인키를 생성해야 합니다.

```
Input the number of private key [1]
>
```

서버의 개인키는 다음과 같은 명령어로도 생성할 수 있습니다.

```
$ coinstack-signon key create
```

구체적인 인자는 help key 명령어를 통해 확인하실 수 있습니다.

```
$ coinstack-signon help key
SYNOPSIS
    coinstack-signon key create [[--log|-l] <loglevel>] [--number|-n] [--address|-a]

DESCRIPTION
    Create and print out a private key randomly.

OPTIONS
```

```
--address, -a
Print out not only private key but also address. You can
separate them with whitespace.

--number, -n
The number of private keys to generate.

--log, -l
To known something wrong, turn on logger. You can
specify log level as next:
error, warn, info, debug, info, custom
The log level specify log configuration file in conf
directory.(ex logback-cli-error.xml) You can customize
any level as your needs.

EXAMPLES
$ coinstack-signon key create -n 20 --address
```

인자	설명
--address, -a	어드레스 출력
--number, -n	개인키 개수
--log, -l	로그

• 설정 파일 생성

```
Process to generate the configuration file? [y/n] y
```

※ 관리자의 주소와 서버의 개인키를 따로 설정하는 이유는 [Coinstack SignOn](#)을 참조합니다.

일괄 처리 명령을 통해 설정하기

Coinstack SignOn은 사용자 입력 없이 서버 설정을 할 수 있는 명령 스크립트를 제공합니다. 일괄 처리(batch) 처리 명령을 통한 설정은 다음과 같은 순서로 진행할 수 있습니다.

설정을 위한 값 지정

다음 명령을 수행하면 디폴트 설정값을 변경할 수 있습니다.

```
$ export NODE_ENDPOINT=${NODE_ENDPOINT}
$ export ADMIN_ADDRESS=${ADMIN_ADDRESS}
$ export SIGNON_SERVER_PORT=${SIGNON_SERVER_PORT}
$ export N_SERVER_PRIVATEKEYS=${N_SERVER_PRIVATEKEYS}
```

설정 환경 변수는 다음과 같습니다.

환경 변수명	의미
NODE_ENDPOINT	연동할 Coinstack Node의 Endpoint
ADMIN_ADDRESS	Coinstack SignOn 관리자의 주소
SIGNON_SERVER_PORT	Coinstack SignOn 서버의 포트
N_SERVER_PRIVATEKEYS	Coinstack SignOn 서버의 개인키의 개수

설정 파일 생성

다음 명령을 수행하면 설정 파일이 자동 생성됩니다.

```
$ coinstack-signon server configure --batch
```

설정 파일을 직접 수정하기

사용자는 설정 파일을 직접 수정해서 설정을 적용할 수 있습니다. 설정은 \${INSTALL_PATH}/conf/coinstack.yaml에 저장됩니다. 설정 파일은 yaml 형식을 따르며, 각 설정 항목에 관련된 내용은 다음과 같습니다.

설정 항목	의미
coinstack.endpoint	연동할 Coinstack Node의 Endpoint
signon.admin.address	Coinstack SignOn 관리자의 주소
signon.server.port	Coinstack SignOn 서버의 포트
signon.server.privatekeys	Coinstack SignOn 서버의 개인키

coinstack.yaml

```
coinstack:
  endpoint: http://localhost:3000

signon:
  admin:
    address: 1KkM2NLZEK9SL4sChJVgg45k3godj7Tnix
  server:
    port: 8080
    privatekeys: |
      ${SERVER_PRIVATEKEY}
      .
```

서버 코인 충전

기본적으로 블록체인에서 각 블록 여러 트랜잭션(거래)들로 구성되며, 트랜잭션(거래)은 입력들과 출력들로 구성되어있습니다.

거래 정보 예제

Inputs	Outputs
Input_0	Output_0
Input_1	Output_1
Input_2	

이러한 거래에서 아직 사용되지 않은 출력은 UTXO(Unspent transaction output)이라고 하며 다른 거래의 입력으로 사용될 수 있습니다. 이미 사용된 출력은 다른 거래에 포함될 수 없고, 만일 다른 두 개 이상의 거래 입력으로 포함된다면 이중 지불(Double spent)이 발생하였다고 합니다. 블록 체인의 거래는 이러한 규칙 아래에서 입력의 주소에 대한 개인키로 거래에 서명함으로써, 무결성을 보장하게 됩니다.

앞서 설명한 것처럼, Cointstack SignOn은 관리자 주소와 서버 주소들을 사용하기 때문에 주소에 코인을 충전해야합니다. 서버의 성능에 따라 다르지만, 동시에 처리되는 요청을 처리하기 위해 서버당 20개 이상의 주소를 확보해야합니다. Cointstack SignOn은 이러한 주소에 코인을 충전하는 번거로움을 덜어주기 위해 코인 충전을 위한 명령을제공합니다.

코인 충전하기

코인을 충전하기 위해서는 다음과 같은 명령을 사용합니다.

```
$ coinstack-signon coin recharge --privatekey ${COIN_SENDER_PRIVATEKEY}
```

이 명령은 설정에 있는 Admin과 Server Privatekey, Peer Address에 1코인을 충전해 줍니다. 명령을 실행하면,충전한 주소들을 출력해 줍니다.

만일, 더 많은 코인을 충전하기를 원한다면, 다음과 같이 인자로 충전할 코인량을 지정할 수 있습니다.

```
$ coinstack-signon coin recharge --privatekey ${COIN_SENDER_PRIVATEKEY} 30
```

자주 발생하는 문제

코인 부족

송금을 하려고 하는 주소에 충분한 코인이 없다면 다음과 같은 예외가 발생합니다.

Insufficient fund

```
io.blocko.coinstack.exception.InsufficientFundException: Insufficient fund
```

```
at io.blocko.coinstack.AbstractTransactionBuilder.preBuild\(\AbstractTransactionBuilder.java:283\)
at io.blocko.coinstack.TransactionBuilder.buildTransaction\(\TransactionBuilder.java:25\)
at coinstack.signon.command.RechargeCoin.execute\(\RechargeCoin.java:54\)
at coinstack.signon.exec.AbstractLauncher.run\(\AbstractLauncher.java:80\)
at org.springframework.boot.SpringApplication.callRunner\(\SpringApplication.java:781\)
at org.springframework.boot.SpringApplication.callRunners\(\SpringApplication.java:771\)
at org.springframework.boot.SpringApplication.run\(\SpringApplication.java:335\)
at org.springframework.boot.builder.SpringApplicationBuilder.run\(\SpringApplicationBuilder.java:137\)
```

```
at coinstack.signon.exec.RechargeCoinLauncher.main\RechargeCoinLauncher.java:30\
```

Fail to execute the command

수수료 부족

블럭 체인에 데이터를 기록하기 위해서는 수수료를 지급해야 합니다. 이 수수료는 트랜잭션에 기록되는 데이터의 양에 따라 일정 이상의 금액을 지정해야 합니다. 만일 여러 인풋과 여러 아웃풋이 트랜잭션에 포함된다면 트랜잭션의 크기가 커지게 됩니다. 만일 데이터 기록량에 필요한 수수료보다 적은 수수료를 지정하면, 다음과 같은 예외를 발생시키며 이 트랜잭션을 거부합니다.

The request could not be processed. :

```
transactionfd42522745408006eb75702cb3725a1aa8a9a8c2c2691627e4c11a4d63f2e436 has 10000fees which is  
under the required amount of 59255
```

```
io.blocko.coinstack.exception.CoinStackException: The request could not be processed. :transaction
```

```
fd42522745408006eb75702cb3725a1aa8a9a8c2c2691627e4c11a4d63f2e436has 10000 fees which is under the  
required amount of 59255
```

```
at io.blocko.coinstack.util.JsonToModel.parseError\JsonToModel.java:29\)  
at io.blocko.coinstack.backendadaptor.CoreBackEndAdaptor.processError\CoreBackEndAdaptor.java:180\)  
at io.blocko.coinstack.backendadaptor.CoreBackEndAdaptor.requestPost\CoreBackEndAdaptor.java:144\)  
at io.blocko.coinstack.backendadaptor.CoreBackEndAdaptor.sendTransaction\CoreBackEndAdaptor.java:419\)  
at io.blocko.coinstack.CoinStackClient.sendTransaction\CoinStackClient.java:916\)  
at coinstack.signon.command.RechargeCoin.execute\RechargeCoin.java:55\)  
at coinstack.signon.exec.AbstractLauncher.run\AbstractLauncher.java:80\)  
at org.springframework.boot.SpringApplication.callRunner\SpringApplication.java:781\)  
at org.springframework.boot.SpringApplication.callRunners\SpringApplication.java:771\)  
at org.springframework.boot.SpringApplication.run\SpringApplication.java:335\)  
at org.springframework.boot.builder.SpringApplicationBuilder.run\SpringApplicationBuilder.java:137\)  
at coinstack.signon.exec.RechargeCoinLauncher.main\RechargeCoinLauncher.java:30\)
```

Fail to execute the command

서버 로그

서버에서 발생하는 다양한 이벤트와 정보들은 로깅이라는 형태로 기록할 수 있습니다. 하지만, 모든 이벤트와 정보를 남기게 되면 서버 성능이 저하되고, 로그를 적게 남기면 문제가 발생했을 때 원인을 파악하기 어렵습니다. 따라서, 개발 및 테스트 환경과 운영 환경처럼 서로 다른 로그 정책을 적용해야 합니다.

로그 확인하기

서버 로그를 확인하는 명령은 다음과 같습니다.

```
$ coinstack-signon server log
```

만약, 다음과 같은 문구가 출력된다면. 서버가 실행 중인지 확인하시고 다시 시도해주시기 바랍니다.

```
No process detected
```

로그 확인을 그만 두고 싶으시면, +C 를 통해 현재 상태에서 빠져나갈 수 있습니다.

해당 로그 파일들은 \${INSTALL_PATH}/log 디렉터리 아래 **server-YYYY-mm-dd.log** 형식으로 저장되어집니다.

Logback

Coinstack SignOn는 자바로 만들어져 있으며, Logback이라는 로거를 사용하고 있습니다. 서버에서는 \${INSTALL_PATH}/conf/logback-server.xml 파일을 통해 설정을 변경할 수 있습니다. 기본적으로 제공되는 로그 설정은 다음과 같습니다.

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>%d{HH:mm} %highlight(%-5level) %cyan(%logger{15}) - %msg%n</Pattern>
        </layout>
    </appender>

    <appender name="DAILY" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- encoders are assigned the type
            ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- daily rollover -->
            <fileNamePattern>${COINSTACK_SIGNON_LOG}/server-%d{yyyy-MM-dd}.log</fileNamePattern>
            <maxHistory>200</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="WARN">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="DAILY" />
    </root>

    <!-- Exception notification for services -->
    <logger name="org.springframework.security.oauth2.provider.endpoint" level="info" />

    <!-- Keep silent for warning and error logging because it is already expected -->
    <logger name="org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter" level="none" />
    <logger name="org.springframework.context.annotation.ConfigurationClassPostProcessor" level="error" />
</configuration>
```

본 장에서는 다음과 같은 내용을 변경하는 방법을 설명합니다.

- 로그 레벨
- 로그 메시지 포맷

로그 레벨

로그를 설정하는 대표적인 방법으로 로그의 레벨을 조정하는 방법이 있습니다. 로그 레벨의 종류는 다음과 같습니다.

- ERROR
- WARN
- INFO
- DEBUG
- TRACE

INFO레벨로 지정하면, ERROR, WARN, INFO 레벨의 메시지가 출력되며, TRACE레벨로 지정하면 ERROR,WARN, INFO, DEBUG, TRACE레벨의 메시지가 출력됩니다. 기본적으로 INFO 레벨로 지정되어 있습니다.

로그 메시지 포맷

로그 메시지에 추가적으로 필요한 정보가 있거나 필요없는 정보는 삭제할 수 있습니다. 로그 메시지의 형태는 layout에 의해서 처리됩니다. 본 문서에서는 가장 많이 사용되는 PatternLayout을 이용한 메시지 포맷에 대해서 설명합니다.

Pattern

패턴은 로그 메시지를 출력하는 형태를 지정합니다. 패턴은 특정한 동작을 지정하는 %로 시작하는 지시어와 상수 문자열로 나뉩니다. 지시어의 종류는 다음과 같습니다.

지시어	의미
c{length} / lo{length} / logger{length}	메시지를 출력한 로거의 이름
C{length} / class{length}	로그를 출력한 클래스의 이름
contextName / cn	로그 컨텍스트의 이름
d{pattern} / date{pattern} / d{pattern, timezone} / date{pattern,timezone}	로그 메시지를 남긴 일시
F / file	로그를 남긴 자바 파일 이름
caller{depth} / caller{depthStart..depthEnd} / caller{depth,evaluator-1, ..., evaluator-n} / caller{depthStart..depthEnd,evaluator-1, ...evaluator-n}	로그 메시지를 남긴 시점의 callstack 정보
L / line	로그를 남긴 자바 파일의 라인 번호
m / msg / message	로그 메시지
M / method	로그를 남긴 메서드
n	줄바꿈 문자.(\\n or \\r\\n)
p / le / level	로그 레벨
r / relative	서버가 시작한 이후 로그를 남긴시간
t / thread	로그를 남긴 스레드의 이름
X{key:-default} / mdc{key:-default}	MDC에 저장된 "key"에 대한 값
ex{depth} / exception{depth} / throwable{depth} / ex{depth,evaluator-1, ..., evaluator-n} / exception{depth, evaluator-1, ..., evaluator-n} / throwable{depth, evaluator-1, ..., evaluator-n}	로그에 부가적으로 담겨있는 예외
xEx{depth} / xException{depth} / xThrowable{depth} / xEx{depth, evaluator-1, ..., evaluator-n} / xException{depth,evaluator-1, ..., evaluator-n} / xThrowable{depth, evaluator-1, ..., evaluator-n}	ex와 동일한 정보인데, 추가적으로 클래스의 패키징 정보를 가지고 있습니다.
nopex / nopexception	예외 정보를 출력하지 않음
marker	로거를 남길 때, 지정한 마커
	"key"속성에 대한 값. 속성은 설정 파일

property{key}	이나 JVM 옵션을 통해 전달되거나 OS 환경 변수를 참조합니다.
replace(p){r, t}	p 문자열에서 정규표현식 r을 찾아서 t로 치환
rEx{depth} / rootException{depth} / rEx{depth, evaluator-1, ..., evaluator-n} / rootException{depth, evaluator-1, ..., evaluator-n}	로그의 예외 정보에서 가장 원인이 되는 예외

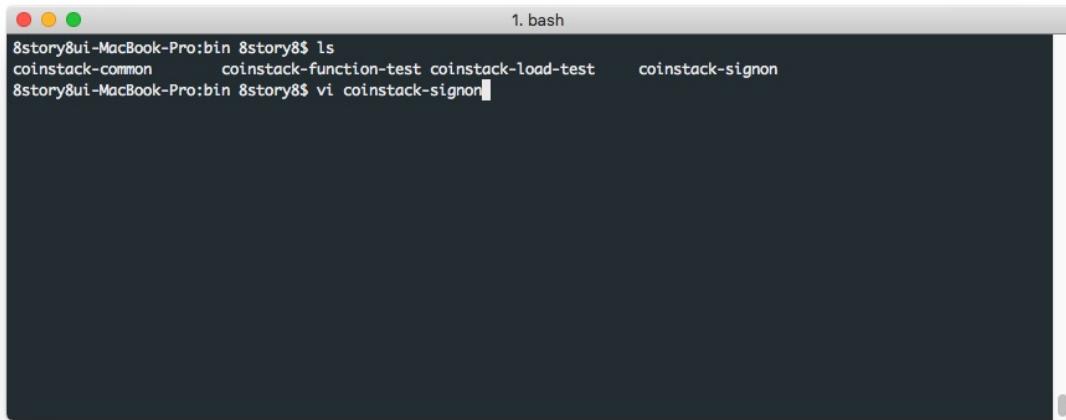
좀 더 상세한 내용을 위해서는 <https://logback.qos.ch/manual/layouts.html>를 참조하시기 바랍니다.

서버 관리

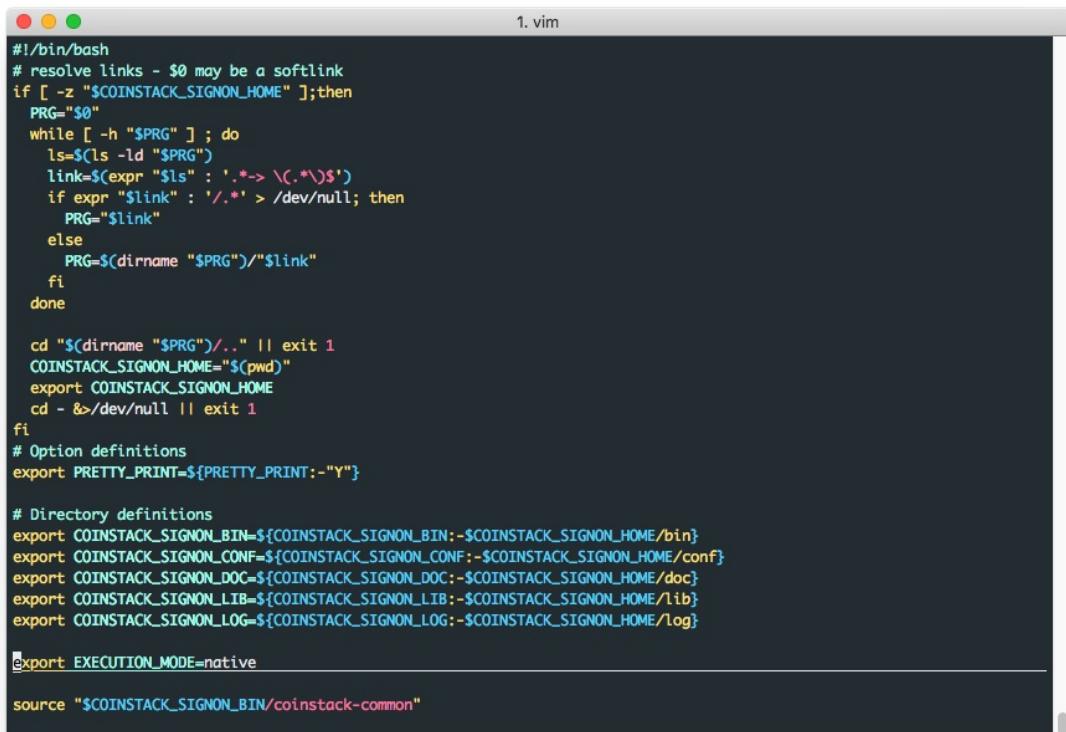
Coinstack SignOn 서버는 직접 컴퓨터에 설치하여 구동할 수 있는 **Native 모드**와 특정 환경에 상관없이 구동할 수 있는 **Docker 모드**로 관리할 수 있습니다.

서버 구동 모드

Coinstack SignOn 서버는 다음 화면처럼 native, docker 값으로 구동 모드를 설정할 수 있습니다. 기본 설정은 **Native 모드**입니다.



```
8story8ui-MacBook-Pro:bin 8story8$ ls
coinstack-common      coinstack-function-test coinstack-load-test      coinstack-signon
8story8ui-MacBook-Pro:bin 8story8$ vi coinstack-signon
```



```
#!/bin/bash
# resolve links - $0 may be a softlink
if [ -z "$COINSTACK_SIGNON_HOME" ];then
    PRG="$0"
    while [ -h "$PRG" ] ; do
        ls=$(ls -ld "$PRG")
        link=$(expr "$ls" : '.*-> \(.*\)$')
        if expr "$link" : '/.*' > /dev/null; then
            PRG="$link"
        else
            PRG=$(dirname "$PRG")/"$link"
        fi
    done
    cd "$(dirname "$PRG")/.." || exit 1
    COINSTACK_SIGNON_HOME=$(pwd)
    export COINSTACK_SIGNON_HOME
    cd - &>/dev/null || exit 1
fi
# Option definitions
export PRETTY_PRINT=${PRETTY_PRINT:-"Y"}

# Directory definitions
export COINSTACK_SIGNON_BIN=${COINSTACK_SIGNON_BIN:-$COINSTACK_SIGNON_HOME/bin}
export COINSTACK_SIGNON_CONF=${COINSTACK_SIGNON_CONF:-$COINSTACK_SIGNON_HOME/conf}
export COINSTACK_SIGNON_DOC=${COINSTACK_SIGNON_DOC:-$COINSTACK_SIGNON_HOME/doc}
export COINSTACK_SIGNON_LIB=${COINSTACK_SIGNON_LIB:-$COINSTACK_SIGNON_HOME/lib}
export COINSTACK_SIGNON_LOG=${COINSTACK_SIGNON_LOG:-$COINSTACK_SIGNON_HOME/log}

export EXECUTION_MODE=native

source "$COINSTACK_SIGNON_BIN/coinstack-common"
```

서버 실행

Coinstack SignOn 서버는 다음의 명령어를 통해 실행 가능합니다.

일반 실행

Docker 모드의 경우 백그라운드 process로 유지하기 때문에 run 명령을 지원하지 않습니다.

```
$ coinstack-signon server run
```

데몬 실행

run으로 서버를 실행하는 경우 run 명령이 끝나게 되면 프로세스가 소멸하지만 데몬 구동은 명령이 끝나도 백그라운드 process가 유지됩니다.

데몬 실행을 하는 방법은 run에 --daemon 옵션을 추가하거나 server start라는 명령어를 사용하시면 됩니다.

```
$ coinstack-signon server run --daemon
```

```
$ coinstack-signon server start
```

구동이 완료되면 아래와 같은 문구가 나옵니다.

```
Server started
```

프로필 설정

프로필을 사용하면 미리 설정해 놓은 환경설정 파일들을 반영할 수 있습니다.

프로필은 다음의 명령어를 통해 사용 가능합니다.

Docker 모드의 경우 start 명령으로 프로필 설정을 지원합니다.

```
$ coinstack-signon server start --profile test
$ coinstack-signon server run --profile test
```

서버 종료

Coinstack SignOn 서버는 다음의 명령어를 통해 종료 가능합니다.

```
// Gracefully shutdown
$ coinstack-signon server stop

// Forcefully shutdown
$ coinstack-signon server stop --force
```

종료가 잘 되면 하단의 문구가 출력됩니다.

```
Server stopped
```

서버 재실행

Coinstack SignOn 서버는 다음의 명령어를 통해 재실행 가능합니다.

```
// Gracefully server restart
$ coinstack-signon server restart

// Forcefully server restart
$ coinstack-signon server restart --force
```

서버 상태 확인

서버가 정상적으로 서비스를 제공하는지 HTTP 요청을 통해 확인할 수 있습니다.

다음의 명령어를 통해 상태 확인이 가능합니다.

```
$ coinstack-signon server check
```

- 서버가 부팅 중인 경우

```
Server is booting
```

- 서버가 구동 중인 경우

```
Server is running
```

- 서버가 구동 중이지 않은 경우

```
Server stopped
```

서버 현황

서버의 OAuth 2.0 서비스 현황을 도표 및 그래프를 통해 실시간으로 확인할 수 있습니다.

다음의 명령어를 통해 사용 가능합니다.

```
$ coinstack-signon server monitor
```

만약 다음과 같은 문구가 출력된다면, 서버가 실행 중인지 확인하시고 다시 시도해주시기 바랍니다.

```
No process detected
```

정상적으로 명령어가 실행되면 다음과 같은 화면을 확인할 수 있습니다.

이와 같이 화면이 출력되는 이유는 OAuth 2.0 서비스를 이용하지 않았기 때문입니다.

따라서 **기능 테스트**, **부하 테스트** 등의 OAuth 2.0 서비스를 이용하면, 다음과 같이 서버 현황을 확인할 수 있습니다.

확인할 수 있는 리소스의 종류는 다음과 같습니다.

리소스 종류	의미
Authorization Issuance/Consumption	인가코드 발행 / 소비 개수
Token Issuance/Revocation/Refresh	액세스 토큰 발행 / 무효 / 리프레시 개수

Cache Hit Rate	캐시 적중률
Coinstack Read/Write	서버와 코인스택 노드 간의 상호작용 개수(Read / Write)
History Client/Authorization/Token	Client/Authorization/Token의 History 그래프

클라이언트

OAuth 2.0 서비스를 사용하기 위해서는 클라이언트를 등록해야 합니다. 이 클라이언트는 Cointstack SignOn 서버의 인증 서비스를 사용할 수 있습니다. 자세한 내용은 [OAuth 2.0](#)을 참조하시기 바랍니다.

관리자

클라이언트를 등록하기 위해서는 관리자의 개인키가 필요합니다. 관리자가 아니면 클라이언트를 등록, 삭제할 수 없습니다.

클라이언트 등록

클라이언트는 CLI(Command Line Interface)를 이용하여 등록할 수 있습니다.

클라이언트를 등록하는 명령은 다음과 같으며, 클라이언트 정보는 JSON 형태의 표준 입력으로 전달합니다.

사용자 입력으로 등록하기

사용자 입력을 통한 클라이언트 등록은 다음과 같은 명령으로 가능합니다.

```
$ coinstack-signon client create \
  --privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "clientId": "${CLIENT_ID}",
  "clientSecret": "${CLIENT_SECRET}",
  "authorizedGrantTypes": ["${GRANT_TYPE}", ...],
  "scopes": ["${SCOPES}", ...],
  "registeredRedirectUris": ["${REDIRECT_URI}", ...],
  "authorities": ["${AUTHORITY}", ...],
  "autoApproves": ["${SCOPE}", ...],
  "resourceIds": ["${RESOURCE_ID}", ...],
  "accessTokenValidity": "${ACCESS_TOKEN_VALIDITY}",
  "refreshTokenValidity": "${REFRESH_TOKEN_VALIDITY}",
  "description": "${DESCRIPTION}",
  "additionalInformation": {"${ADDITIONAL_INFO_KEY}": "${ADDITIONAL_INFO_VALUE}"}
}
EOF
```

파일로부터 등록하기

파일을 통한 클라이언트 등록은 다음 절차에 따라 진행합니다.

- 클라이언트 정보 파일인 \${CLIENT_INFO_FILE}을 생성하여 등록할 클라이언트 정보를 저장합니다.

```
[{
  "clientId": "${CLIENT_ID}",
  "clientSecret": "${CLIENT_SECRET}",
  "authorizedGrantTypes": ["${GRANT_TYPE}", ...],
  "scopes": ["${SCOPES}", ...],
  "registeredRedirectUris": ["${REDIRECT_URI}", ...],
  "authorities": ["${AUTHORITY}", ...],
  "autoApproves": ["${SCOPE}", ...],
  "resourceIds": ["${RESOURCE_ID}", ...],
  "accessTokenValidity": "${ACCESS_TOKEN_VALIDITY}",
  "refreshTokenValidity": "${REFRESH_TOKEN_VALIDITY}",
  "description": "${DESCRIPTION}",
  "additionalInformation": {"${ADDITIONAL_INFO_KEY}": "${ADDITIONAL_INFO_VALUE}"}
}, ...]
```

- 클라이언트 정보 파일을 통해 클라이언트 정보를 등록합니다.

```
$ coinstack-signon client create \
  --privatekey ${ADMIN_PRIVATEKEY} \
  --file ${CLIENT_INFO_FILE}
```

클라이언트 속성

이름	설명	필수 유무	예제
clientId	클라이언트를 구분하는 인식값, 최대 영문 20자	필수	"clientId": "trusted"
clientSecret	클라이언트 자격 증명을 위한 문자열, 패스워드		"clientSecret": "secret"
authorizedGrantTypes	인가 방법		"authorizedGrantTypes": ["authorization_code"]
scopes	접근 범위		"scopes": ["read", "write"]
registeredRedirectUris	인증 후 화면을 전환할 대상 URI		"registeredRedirectUris": [" http://resource.blocko.io:9000 "]
authorities	인증 후 얻게 되는 권한		"authorities": ["USER", "TRUSTED_CLIENT"]
autoApproves	자동 인가 여부, scopes에 종속		"autoApproves": ["read", "write"]
resourceIds	접근할 자원		"resourceIds": ["blocko"]
accessTokenValidity	Access token 유효 시간, 단위는 초		"accessTokenValidity": "600"
refreshTokenValidity	Refresh token 유효 시간, 단위는 초		"refreshTokenValidity": "600"
description	클라이언트에 대한 설명		"description": "This is a client"
additionalInformation	시스템별 추가 정보		"additionalInformation" :{"OS":"CentOS"}

클라이언트 조회

등록된 클라이언트는 단일/목록으로 조회할 수 있으며, 다음 명령을 통해 확인할 수 있습니다.

클라이언트 단일 조회

```
$ coinstack-signon client check ${CLIENT_ID}
```

조회가 완료되면 아래와 같은 문구가 나옵니다.

```
AccessTokenValidity      ${ACCESS_TOKEN_VALIDITY}
AdditionalInformation    ${${ADDITIONAL_INFO_KEY}=${${ADDITIONAL_INFO_VALUE}}}
Authorities             ${AUTHORITY}

.
.

AuthorizedGrantTypes    ${GRANT_TYPE}

.
.

AutoApproves            ${SCOPE}

.
.

ClientId                ${CLIENT_ID}
ClientSecret             ${CLIENT_SECRET}
Description              ${DESCRIPTION}
RefreshTokenValidity     ${REFRESH_TOKEN_VALIDITY}
RegisteredRedirectUris   ${REDIRECT_URI}

.
.

ResourceIds             ${RESOURCE_ID}
```

```
Scopes          ${SCOPE}
.
.
```

※ ClientSecret은 SHA256으로 암호화되어 유출되지 않습니다.

조회할 수 없는 클라이언트를 확인했다면 아래와 같은 문구가 나옵니다.

```
The client not found.
```

클라이언트 목록 조회

등록된 클라이언트 목록은 다음 명령을 통해 확인할 수 있습니다.

```
$ coinstack-signon client list \
--privatekey ${ADMIN_PRIVATEKEY}
```

조회가 완료되면 다음과 같은 문구가 나옵니다.

```
Client ID      Description
${CLIENT_ID}   ${DESCRIPTION}
.
.
.
```

※ \${ }가 20자가 넘어갈 경우 ... 으로 처리합니다.

등록한 클라이언트가 정상적으로 조회되지 않는다면 다음을 의심해 볼 수 있습니다.

클라이언트 등록 실패

클라이언트 등록 실행 중 어떤 문제(버그나 잘못된 사용법)로 인해 클라이언트 등록에 실패할 수 있습니다. 보통의 경우, 로그 레벨을 지정해서 동작에 관련된 로그를 남기고 이를 통해 문제를 확인할 수 있습니다.

로그 레벨을 조정하기 위한 파라미터는 --log입니다.

로그 레벨을 trace로 바꾸기 위한 명령은 두 가지 방식이 있습니다.

사용자 입력으로 로그 확인

```
$ coinstack-signon client create \
--log trace \
--privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "clientId": "${CLIENT_ID}",
  "clientSecret": "${CLIENT_SECRET}",
  "authorizedGrantTypes": ["${GRANT_TYPE}", ...],
  "scopes": ["${SCOPES}", ...],
  "registeredRedirectUris": ["${REDIRECT_URI}", ...],
  "authorities": ["${AUTHORITY}", ...],
  "autoApproves": ["${SCOPE}", ...],
  "resourceIds": ["${RESOURCE_ID}", ...],
  "accessTokenValidity": "${ACCESS_TOKEN_VALIDITY}",
  "refreshTokenValidity": "${REFRESH_TOKEN_VALIDITY}",
  "description": "${DESCRIPTION}",
  "additionalInformation": {"${ADDITIONAL_INFO_KEY}": "${ADDITIONAL_INFO_VALUE}"}
}
```

파일로부터 로그 확인

```
$ cat ${CLIENT_INFO_FILE} | \
coinstack-signon client create \
--log trace \
--privatekey ${ADMIN_PRIVATEKEY}
```

블록에 트랜잭션 미포함

Coinstack Node의 마이닝 주기와 같은 설정 때문에 즉각적으로 블록이 생성되지 않을 수 있습니다. 따라서, 노드의 로그나 설정을 통해 확인해 보는 것이 좋습니다. Node의 설정은 본 문서에서 다루지 않습니다.

클라이언트 삭제

사용하지 않는 클라이언트는 삭제하는 것이 더 높은 보안 수준을 유지합니다. 기존의 클라이언트를 삭제하기 위해서는 클라이언트의 ID를 알고 있어야 합니다. 필요하다면 클라이언트를 조회하도록 합니다.

클라이언트를 삭제하기 위한 명령은 다음과 같습니다.

```
$ coinstack-signon client remove \
  --privatekey ${ADMIN_PRIVATEKEY} \
  ${CLIENT_ID}
```

사용자

OAuth 2.0 서비스를 사용하기 위해서는 사용자를 등록해야 합니다. 이 사용자는 Coinstack SignOn 서버의 인증서비스를 사용할 수 있습니다.

자세한 내용은 [OAuth 2.0](#)을 참조하시기 바랍니다.

관리자

사용자를 등록하기 위해서는 관리자의 개인키가 필요합니다. 관리자가 아니면 사용자를 등록, 삭제할 수 없습니다.

사용자 등록

사용자는 CLI(Command Line Interface)를 이용하여 등록할 수 있습니다.

사용자를 등록하는 명령은 다음과 같습니다.

사용자 입력으로 등록하기

사용자 입력을 통한 사용자 등록은 다음과 같은 명령으로 가능합니다.

```
$ coinstack-signon user create \
  --privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "username": "${USER_NAME}",
  "password": "${PASSWORD}",
  "authorities": ["${AUTHORITY}", ...]
}
EOF
```

파일로부터 등록하기

파일을 통한 사용자 등록은 다음 절차에 따라 진행합니다.

- 사용자 정보 파일인 \${USER_INFO_FILE}을 생성하여 등록할 사용자 정보를 저장합니다.

```
[{
  "username": "${USER_NAME}",
  "password": "${PASSWORD}",
  "authorities": ["${AUTHORITY}", ...]
}, ...]
```

- 사용자 정보 파일을 통해 사용자 정보를 등록합니다.

```
$ coinstack-signon user create \
  --privatekey ${ADMIN_PRIVATEKEY} \
  --file ${USER_INFO_FILE}
```

사용자 속성

이름	설명	필수 유무	예제
username	사용자 아이디	필수	"username":"user"
password	사용자 비밀번호	필수	"password":"password"
authorities	인증 후 얻게 되는 권한들	필수	"authorities":["USER", "TRUSTED_CLIENT"]

사용자 조회

등록된 사용자는 다음 명령을 통해 확인할 수 있습니다.

```
$ coinstack-signon user check ${USER_NAME}
```

조회가 완료되면 아래와 같은 문구가 나옵니다.

Authorities	\${AUTHORITY}
 .	
Password	\${PASSWORD}
Username	\${USER_NAME}

※ Password는 SHA256으로 암호화되어 유출되지 않습니다.

조회할 수 없는 사용자를 확인했다면 아래와 같은 문구가 나옵니다.

```
The user not found.
```

등록한 사용자가 정상적으로 조회되지 않는다면 다음을 의심해볼 수 있습니다.

사용자 등록 실패

사용자 등록 실행 중 어떤 문제(버그나 잘못된 사용법)로 인해 사용자 등록에 실패할 수 있습니다. 보통의 경우, 로그레벨을 지정해서 동작에 관련된 로그를 남기고 이를 통해 문제를 확인할 수 있습니다.

로그 레벨을 조정하기 위한 파라미터는 `--log`입니다.

로그 레벨을 trace로 바꾸기 위한 명령은 두 가지 방식이 있습니다.

사용자 입력으로 로그 확인

```
$ coinstack-signon user create \
  --log trace \
  --privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "username": "${USER_NAME}",
  "password": "${PASSWORD}",
  "authorities": ["${AUTHORITY}", ...]
}
```

파일로부터 로그 확인

```
$ cat ${USER_INFO_FILE} | \
coinstack-signon user create \
  --log trace \
  --privatekey ${ADMIN_PRIVATEKEY}
```

블록에 트랜잭션 미포함

Coinstack Node의 마이닝 주기와 같은 설정 때문에 즉각적으로 블록이 생성되지 않을 수 있습니다. 따라서, 노드의 로그나 설정을 통해 확인해보는 것이 좋습니다. Node의 설정은 본 문서에서 다루지 않습니다.

사용자 삭제

사용하지 않는 사용자는 삭제하는 것이 더 높은 보안 수준을 유지합니다. 기존의 사용자를 삭제하기 위해서는 사용자의 ID인 `${USER_NAME}`을 알고 있어야 합니다. 필요하다면 사용자를 조회하도록 합니다.

사용자를 삭제하기 위한 명령은 다음과 같습니다.

```
$ coinstack-signon user remove \
  --privatekey ${ADMIN_PRIVATEKEY} \
  ${USER_NAME}
```


인가 코드 확인

인가 코드는 액세스 토큰을 발급받기 위해 서버로부터 발급받은 값입니다.

인가 코드 발행

인가 코드는 다음의 명령을 통해 발행 가능합니다.

```
$ coinstack-signon code create --client ${CLIENT_ID} --user ${USER_NAME} --password ${PASSWORD} --endpoint ${ENDPOINT}
```

명령어 실행 후 다음과 같이 출력된다면, 클라이언트 아이디 또는 사용자 정보가 유효하지 않아 발생한 오류입니다.

```
Check your clientId or username, password.
```

만약, Coinstack SignOn 서버가 꺼져있다면 다음과 같은 결과를 확인할 수 있습니다.

```
Connection refused. Check server endpoint.
```

인가 코드 정보 조회

발급받은 인가 코드에 관한 정보는 다음의 명령어를 통해 확인할 수 있습니다.

```
$ coinstack-signon code check ${AUTHORIZATION_CODE}
```

조회가 잘 되면 다음과 같은 결과를 확인할 수 있습니다.

```
Authentication
  AccessTokenValue
  Authorities      ${AUTHORITY}

  .
  .

  ClientId        ${CLIENT_ID}
  Details
  RedirectUri     ${REDIRECT_URI}
  RequestParameters ${PARAMETERS}
  ResourceIds
  RolesOfAuthorities
  Scope           ${SCOPE}

  .
  .

  UserPrinciple   ${USER_PRINCIPLE}

Value          ${AUTH_CODE}
```

유효하지 않은 인가 코드 정보를 조회하면 다음과 같은 결과를 확인할 수 있습니다.

```
The code not found.
```

인증 토큰 확인 및 만료

토큰은 리소스 서버에 리소스를 요청하기 위해 인가 서버로부터 발급받은 값입니다.

액세스 토큰 발행

액세스 토큰은 [인가증명\(Grant\)](#)에서 설명한 4가지 방식을 통해 발급 가능합니다.

```
// Authorization code
$ coinstack-signon token create --type code --client ${CLIENT_ID} --secret ${CLIENT_SECRET} --endpoint ${ENDPOINT} --redirect ${REDIRECT_URI} --code ${AUTHORIZATION_CODE}

// Implicit
$ coinstack-signon token create --type implicit --user ${USERNAME} --password ${PASSWORD} --client ${CLIENT_ID} --secret ${CLIENT_SECRET} --endpoint ${ENDPOINT}

// Resource Owner Password Credentials
$ coinstack-signon token create --type password --user ${USERNAME} --password ${PASSWORD} --client ${CLIENT_ID} --secret ${CLIENT_SECRET} --endpoint ${ENDPOINT}

// Client Credentials
$ coinstack-signon token create --type credentials --client ${CLIENT_ID} --secret ${CLIENT_SECRET} --endpoint ${ENDPOINT}
```

명령어 실행 후 다음과 같이 출력된다면, 클라이언트 아이디 또는 사용자 정보가 유효하지 않아 발생한 오류입니다.

```
Check your clientId or username, password.
```

만약, 다음과 같은 문구가 출력된다면 유효하지 않은 인가 코드를 입력하여 발생한 오류입니다.

```
The code not found.
```

만약, Coinstack SignOn 서버가 꺼져있다면 다음과 같은 결과를 확인할 수 있습니다.

```
Connection refused. Check server endpoint.
```

액세스 토큰 정보 조회

발급받은 액세스 토큰에 관한 정보는 다음의 명령어를 통해 확인할 수 있습니다.

```
// Direct access to the block chain
$ coinstack-signon token check ${ACCESS_TOKEN}

// Use endpoint
$ coinstack-signon token check --client ${CLIENT_ID} --secret ${CLIENT_SECRET} --endpoint ${ENDPOINT} ${ACCESS_TOKEN}
```

조회가 잘 되면 다음과 같은 결과를 확인할 수 있습니다.

AdditionalInformation	\${{ADDITIONAL_INFO_KEY}}=\${{ADDITIONAL_INFO_VALUE}}
Expiration	\${{DATE}}
RefreshToken	\${{REFRESH_TOKEN_VALUE}}
Scope	\${{SCOPE}}
 .	
 .	
TokenType	\${{TOKEN_TYPE}}
Value	\${{ACCESS_TOKEN_VALUE}}

명령어 실행 후 다음과 같이 출력된다면, 클라이언트 아이디 또는 사용자 정보가 유효하지 않아 발생한 오류입니다.

```
Check your clientId or username, password.
```

만약, 다음과 같은 문구가 출력된다면 유효하지 않은 토큰을 입력하여 발생한 오류입니다.

```
The token not found.
```

만약, Coinstack SignOn 서버가 꺼져있다면 다음과 같은 결과를 확인할 수 있습니다.

```
Connection refused. Check server endpoint.
```

리프레시 토큰으로 재발행

발급받은 액세스 토큰과 함께 제공되어지는 리프레시 토큰을 이용하여 만료되기 전인 액세스 토큰을 다음의 명령어를 통해 재발행할 수 있습니다.

```
// Direct access to the block chain  
$ coinstack-signon token refresh --privatekey ${ADMIN_PRIVATEKEY} ${REFRESH_TOKEN}  
  
// Use endpoint  
$ coinstack-signon token refresh --client ${CLIENT_ID} --secret ${CLIENT_SECRET} --endpoint ${ENDPOINT} ${REFRESH_TOKEN}
```

명령어 실행 후 다음과 같이 출력된다면, 클라이언트 아이디 또는 사용자 정보가 유효하지 않아 발생한 오류입니다.

```
Check your clientId or username, password.
```

만약, 다음과 같은 문구가 출력된다면 유효하지 않은 토큰을 입력하여 발생한 오류입니다.

```
The token not found.
```

만약, Coinstack SignOn 서버가 꺼져있다면 다음과 같은 결과를 확인할 수 있습니다.

```
Connection refused. Check server endpoint.
```

액세스 토큰 무효화

발급받은 토큰을 명시적으로 무효화하는 것은 다음의 명령어를 통해 할 수 있습니다.

```
$ coinstack-signon token remove \  
--privatekey ${ADMIN_PRIVATEKEY} \  
${ACCESS_TOKEN}
```

발급받은 액세스 토큰을 무효화하게 되면 같이 발급받은 리프레시 토큰도 무효화 되게 됩니다.

활용

Coinstack SignOn을 활용하는 방법들을 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 사용자 인증 연동하기
- Single Sign On
- 사용자 경의 로그인 페이지

사용자 인증정보 연동

Coinstack SignOn과 연동 중인 사용자 인증 정보 저장소를 변경하는 예제를 제공합니다. 본 예제에서는 간단하게 한 명의 사용자만이 존재하는 사용자 인증 정보 저장소를 구현하여 Coinstack SignOn과 사용자 인증 정보를 연동하는 방법을 보여줍니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

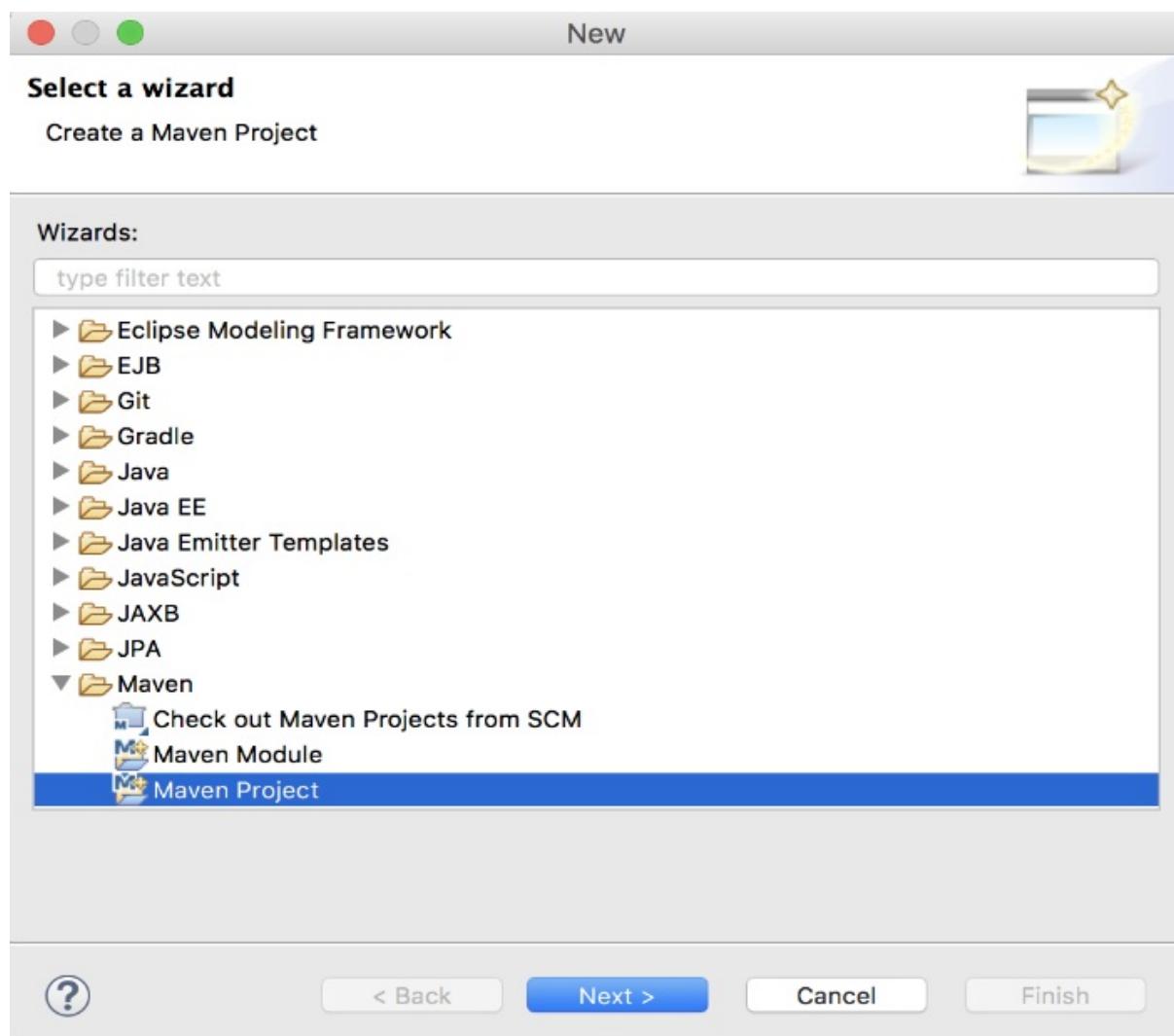
- 프로젝트 설정
- 사용자 인증 정보 저장소 구현
- SignOn 서버와 통합

프로젝트 설정

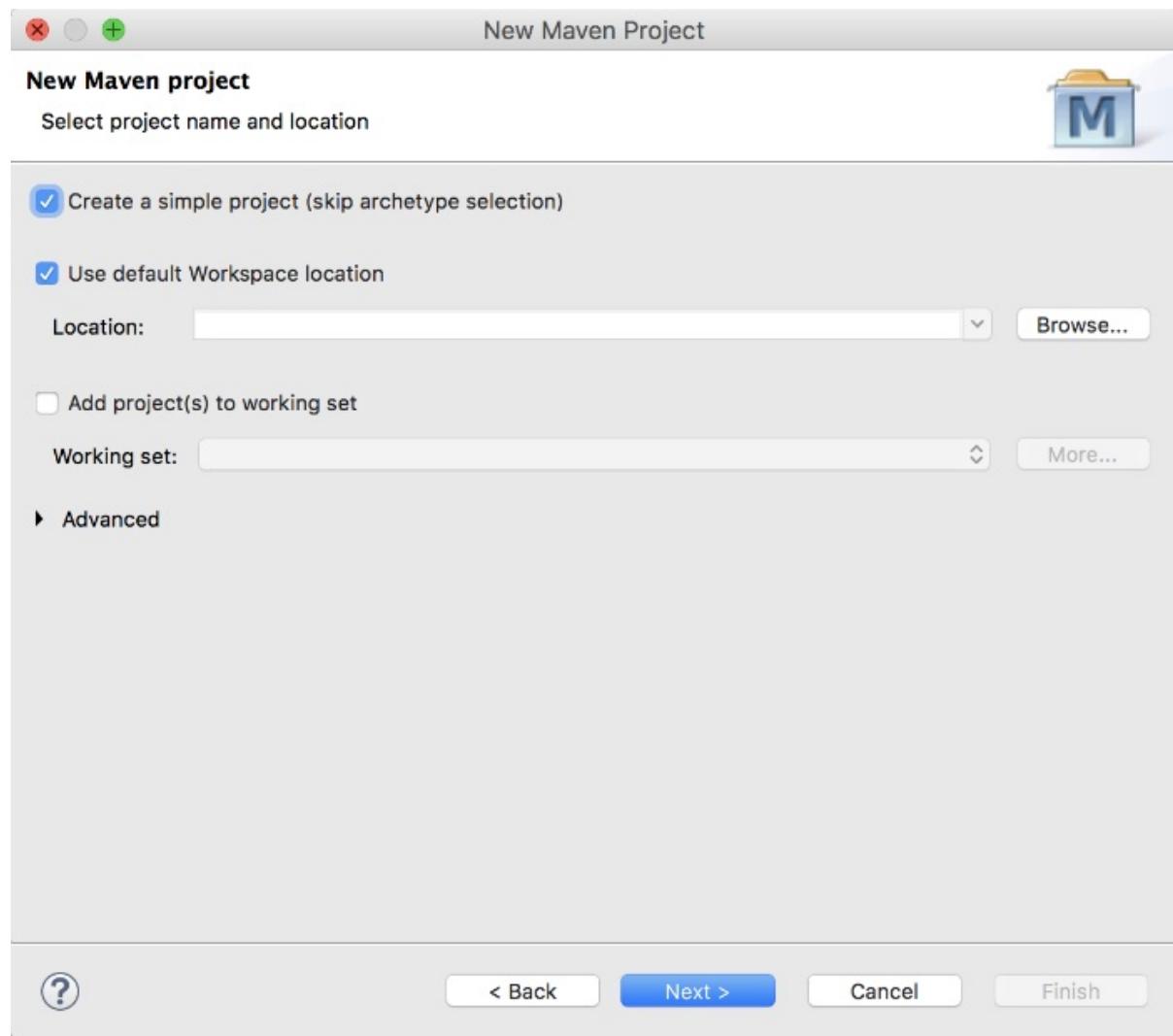
본 장에서는 Eclipse와 Maven을 이용하여 개발 환경을 구성하는 방법에 관하여 설명합니다.

Eclipse에서 Maven 프로젝트 생성하기

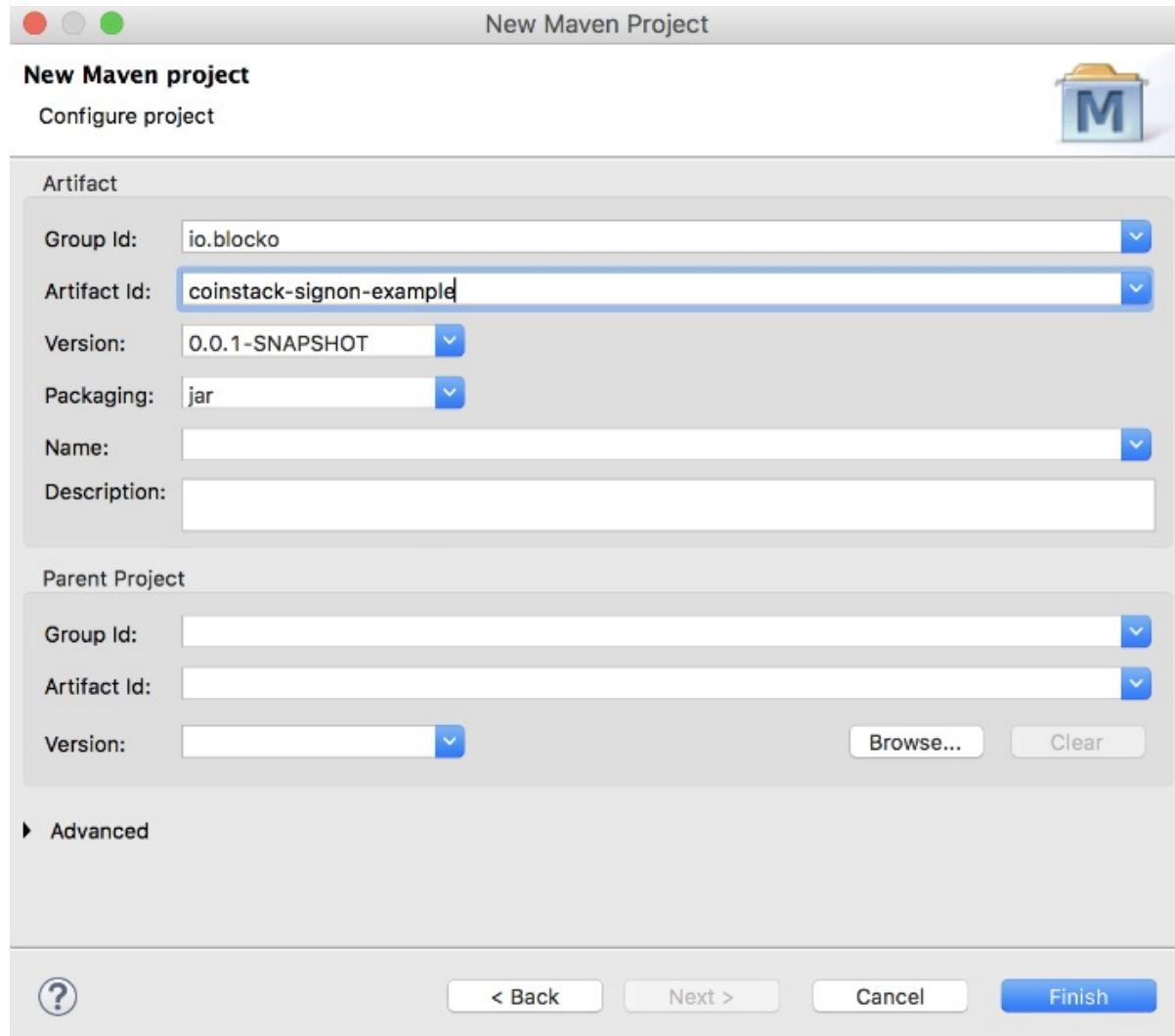
메뉴에서 File -> New를 선택하고, 리스트에서 **Maven Project**를 선택하고 다음으로 진행합니다.



Create a simple project를 체크하고, 다음 단계로 진행합니다.



<group-id>와 <artifact-id>를 입력하고, 과정을 종료하면 프로젝트가 생성됩니다.



pom.xml 설정하기

개발을 진행하기 위해서는 Maven 저장소에서 라이브러리들을 가져와야 합니다. BLOCKO, Inc는 개발에 필요한 모든 라이브러리를 인터넷을 통해 제공합니다. 인터넷이 단절된 환경에서 개발하기 위해서는 해당 라이브러리를 직접다운로드받아서 프로젝트를 구성할 수 있습니다. 관련 라이브러리를 프로젝트에 포함하기 위해서는 \${PROJECT_HOME}/pom.xml 파일을 수정해야 합니다.

라이브러리 추가

pom.xml에서는 라이브러리를 가져오기 위해서 Maven 저장소 정보를 입력하고, 필요한 라이브러리를 의존성에 정의합니다.

```

...
<properties>
    <project.build.sourceEncoding>utf-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>utf-8</project.reporting.outputEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<repositories>
    <repository>
        <id>blocko-maven-release-repository</id>
        <url>https://nexus.blocko.io/repository/blocko-maven-release-repository/</url>
    </repository>
</repositories>
<dependencies>
    <dependency>
        <groupId>io.blocko</groupId>
        <artifactId>coinstack-signon-user</artifactId>
        <version>1.0</version>
    </dependency>

```

```

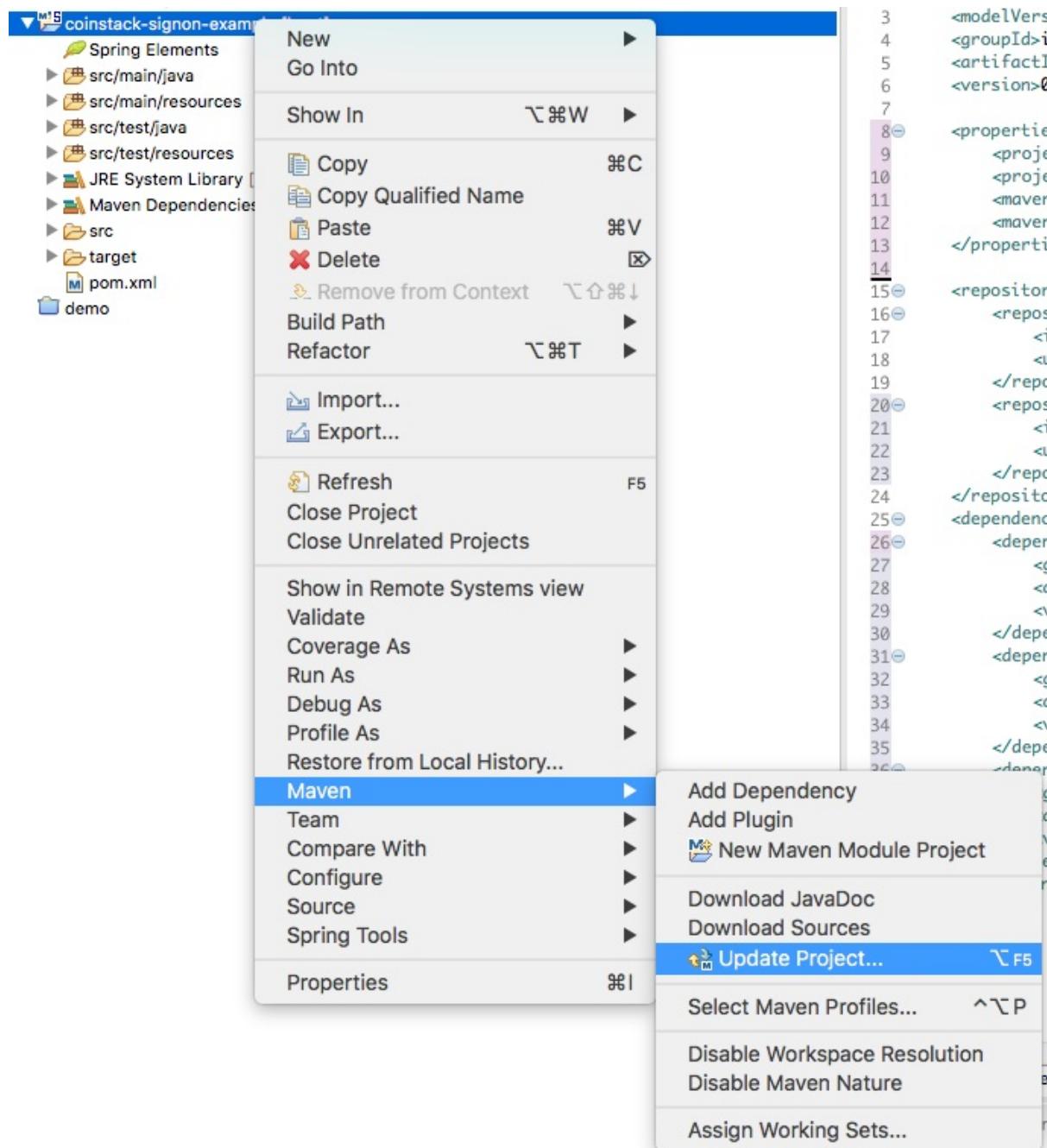
</dependency>
<dependency>
    <groupId>io.blocko</groupId>
    <artifactId>coinstack-signon-common</artifactId>
    <version>1.0</version>
</dependency>
<dependency>
    <groupId>io.blocko</groupId>
    <artifactId>coinstack-spring</artifactId>
    <version>0.4</version>
</dependency>

</dependencies>
...

```

프로젝트 업데이트

pom.xml에 정의한 라이브러리들을 Maven 저장소로부터 가져오려면 해당 프로젝트를 우클릭 -> Maven -> Update Project...를 선택합니다.



선택을 완료하면, Maven 저장소로부터 자동적으로 로컬 저장소로 라이브러리들이 다운로드되어집니다.

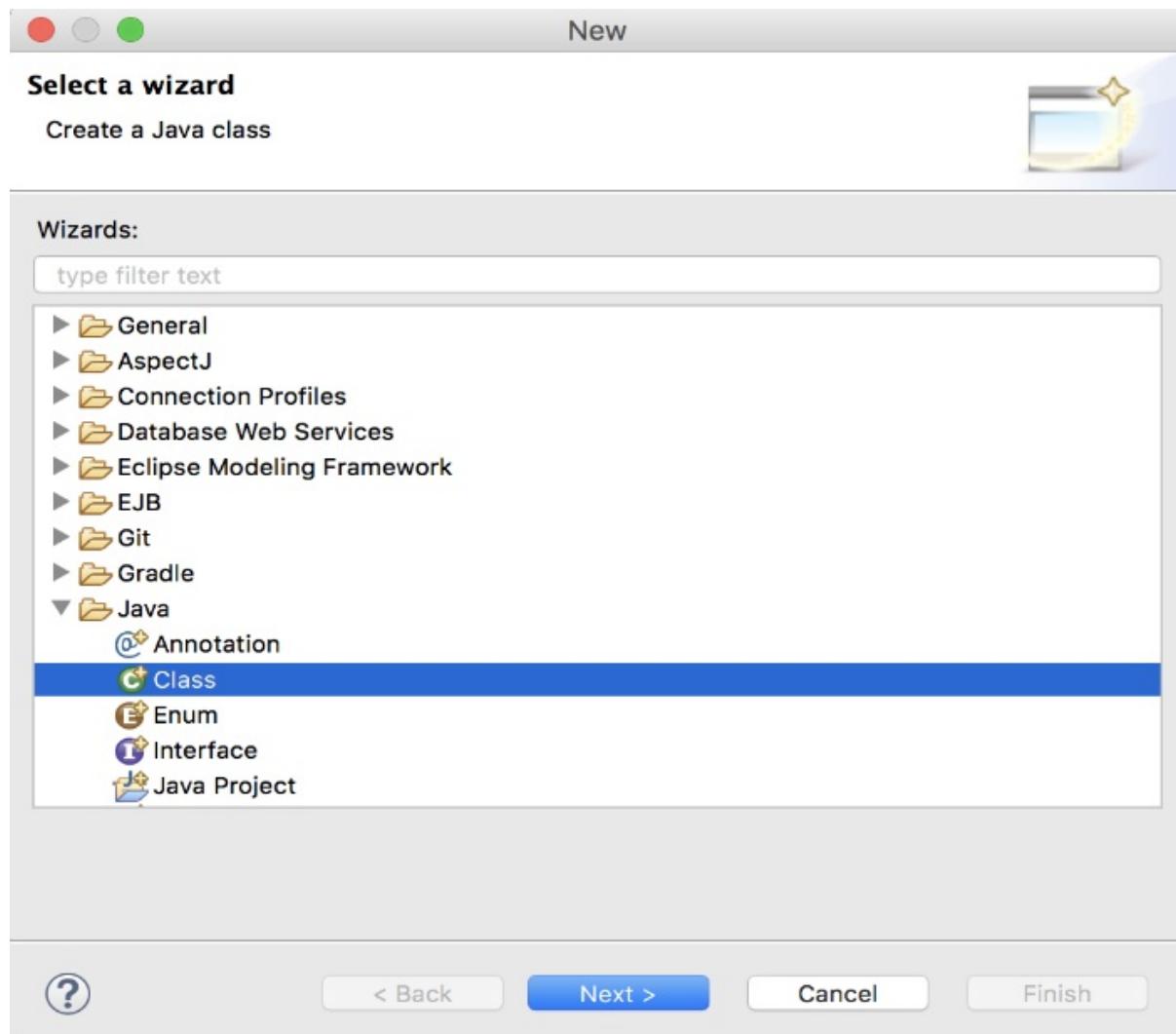
사용자 인증 정보 저장소 구현

Maven 저장소를 통해 받은 coinstack-signon-user.jar 파일 안에는 사용자 인증 정보 저장소 변경에 대비하여 coinstack.signon.repository 패키지 안에 UserDetailsRepository 인터페이스를 제공합니다.

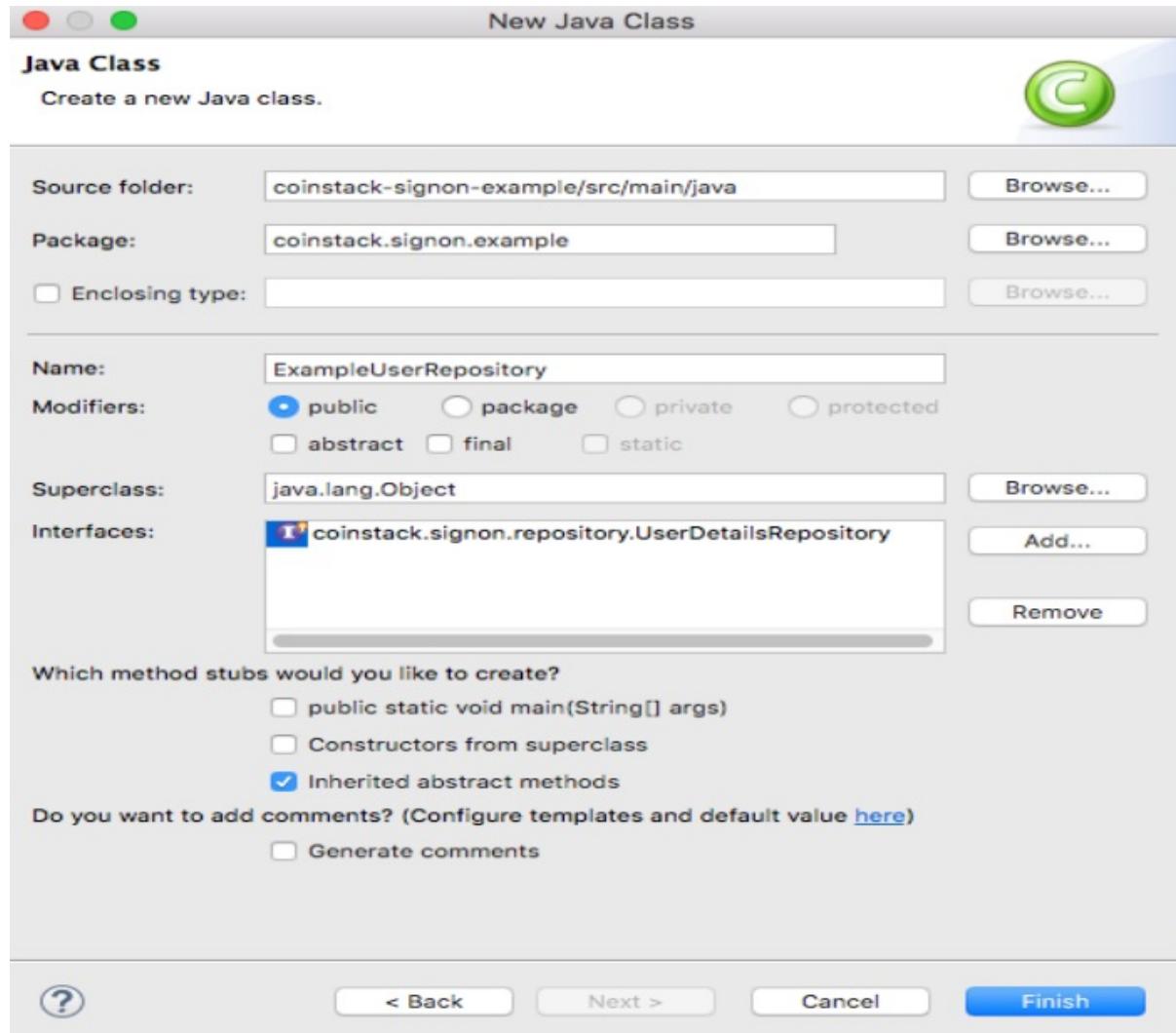
본 예제는 UserDetailsRepository 인터페이스를 이용한 사용자 인증 정보 저장소 구현에 대하여 설명합니다.

사용자 인증 정보 저장소 클래스 생성

해당 프로젝트를 선택하고 메뉴에서 File > New 그리고 Other..을 선택합니다. 그 후, 새 창이 나오면 목록에서 Java Class를 선택합니다.



사용자 인증 정보 저장소로 사용할 클래스의 이름을 입력하고, coinstack.signon.repository.UserDetailsRepository를 인터페이스로 추가합니다.



UserDetailsRepository를 이용한 사용자 인증 정보 저장소 구현

UserDetailsRepository를 통해 ExampleUserRepository에는 다음과 같은 delete, findByUsername, save메소드가 강제되고, 이를 통해 Cointstack SignOn과 호환 가능한 사용자 인증 정보 저장소를 구현할 수 있습니다.

예제에서는 **test-pass**라는 패스워드로 인증을 하고, **USER**라는 역할(권한)을 가진 **test-user**라는 사용자가 저장된 간단한 사용자 인증 정보 저장소를 구현합니다. 요건에 따라 LDAP, Active Directory 또는 기타 인증 서버로부터 사용자 정보를 가져오는 것을 구현함으로써 연동할 수 있습니다.

ExampleUserRepository.java

```
package coinstack.signon.example;

import static java.util.Arrays.asList;

import coinstack.signon.model.User;
import coinstack.util.Sha256PasswordEncoder;
import java.util.Optional;

public class ExampleUserRepository implements UserDetailsRepository {

    @Override
    public void delete(String username){
        ...
    }

    @Override
    public Optional<User> findByUsername(final String username) {

```

```
if ("test-user".equals(username)) {
    return Optional.ofNullable(new User(username, new Sha256PasswordEncoder().encode("test-pass"), asList("USER")));
}
return Optional.empty();
}

@Override
public void save(User user){
    ...
}
```

본 예제는 클래스 하나로 구성되어 있으나 여러 클래스로 구성된 모듈이어도 상관없습니다.

SignOn 서버와 통합

개발된 모듈 빌드하기

Maven을 실행함으로써 모듈을 빌드해서 archive(본 예제에서는 coinstack-signon-example-\${version}.jar) 파일을 생성할 수 있습니다.

```
$ mvn clean package
```

빌드된 archive 파일은 \${PROJECT_HOME}/target에서 찾을 수 있습니다.

모듈 배치(deploy)하기

사용자 모듈을 배치(deploy)하는 과정은 따로 없습니다. 라이브러리를 \${INSTALL_PATH}/lib에 위치시키는 것으로 배치는 끝납니다. 만일 서버가 동작 중이라면, 자동으로 반영되는 것이 아니므로 서버를 재시작해야 합니다.

모듈 로드 설정

서버가 모듈을 로드하기 위해서는 로드할 클래스의 이름을 알아야 합니다. 설정을 위해 \${INSTALL_PATH}/conf에 위치해있는 coinstack.yaml파일에 추가합니다. 추가가 완료되면 서버를 재시작하여 모듈을 적용시킵니다.

coinstack.yaml

```
signon:
  server:
    user-repository: coinstack.signon.example.ExampleUserRepository
```

동작 확인

로그인 페이지

브라우저의 주소창에 [http://\\${SERVER_HOSTNAME}:\\${SERVER_PORT}/welcome.html](http://${SERVER_HOSTNAME}:${SERVER_PORT}/welcome.html)를 입력하면, 설정에 의해 로그인 페이지로 리다이렉트되면서 다음과 같은 화면을 볼 수 있습니다.

Login with Username and Password

User:	<input type="text"/>
Password:	<input type="password"/>
<input type="button" value="Login"/>	

User와 Password에 사용자에서 등록한 사용자를 입력합니다. 해당 사용자와 패스워드는 기본 모듈의 사용자 정보 저장소에 등록된 사용자 정보입니다. 즉, 새로운 모듈을 적용하였으므로, 다음과 같이 로그인에 실패해야 합니다.

Your login attempt was not successful, try again.

Reason: none

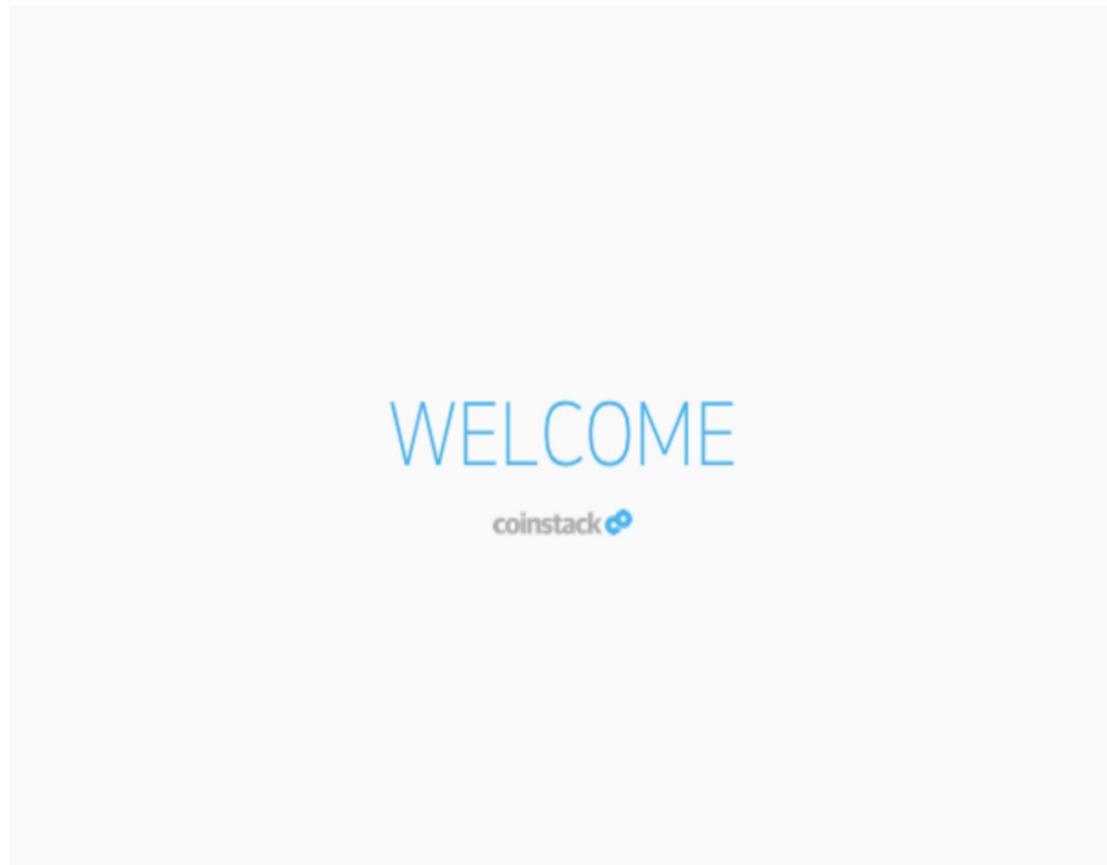
Login with Username and Password

User:

Password:

이제 다시 새로 만든 모듈의 사용자 정보로 로그인을 시도합니다.

User에 , Password에 를 입력하면 다음과 같은 페이지를 볼 수 있습니다.



만일 다음과 같은 페이지를 보게 된다면, URL이 올바르지 않은 경우입니다. 하지만, 인증을 통과한 것이기 때문에 올바르게 테스트한 것입니다. 아래 이미지는 [http://\\${SERVER_HOSTNAME}:\\${SERVER_PORT}/](http://${SERVER_HOSTNAME}:${SERVER_PORT}/)로 접근한 경우 보여지는 페이지입니다.

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Tue Feb 20 14:06:34 JST 2018

There was an unexpected error (type=Not Found, status=404).
No message available

Single Sign On

통합 인증(Single Sign On; SSO)을 이용한 통합 인증 시스템 구성에 관한 예제를 수록하고 있습니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 여러 개의 리소스 서버에 대한 액세스 토큰 공유 방법
- 사용자 역할(권한)에 따른 리소스(페이지) 접근 제어 방법

Servlet 프로젝트 설정

프로젝트 설정

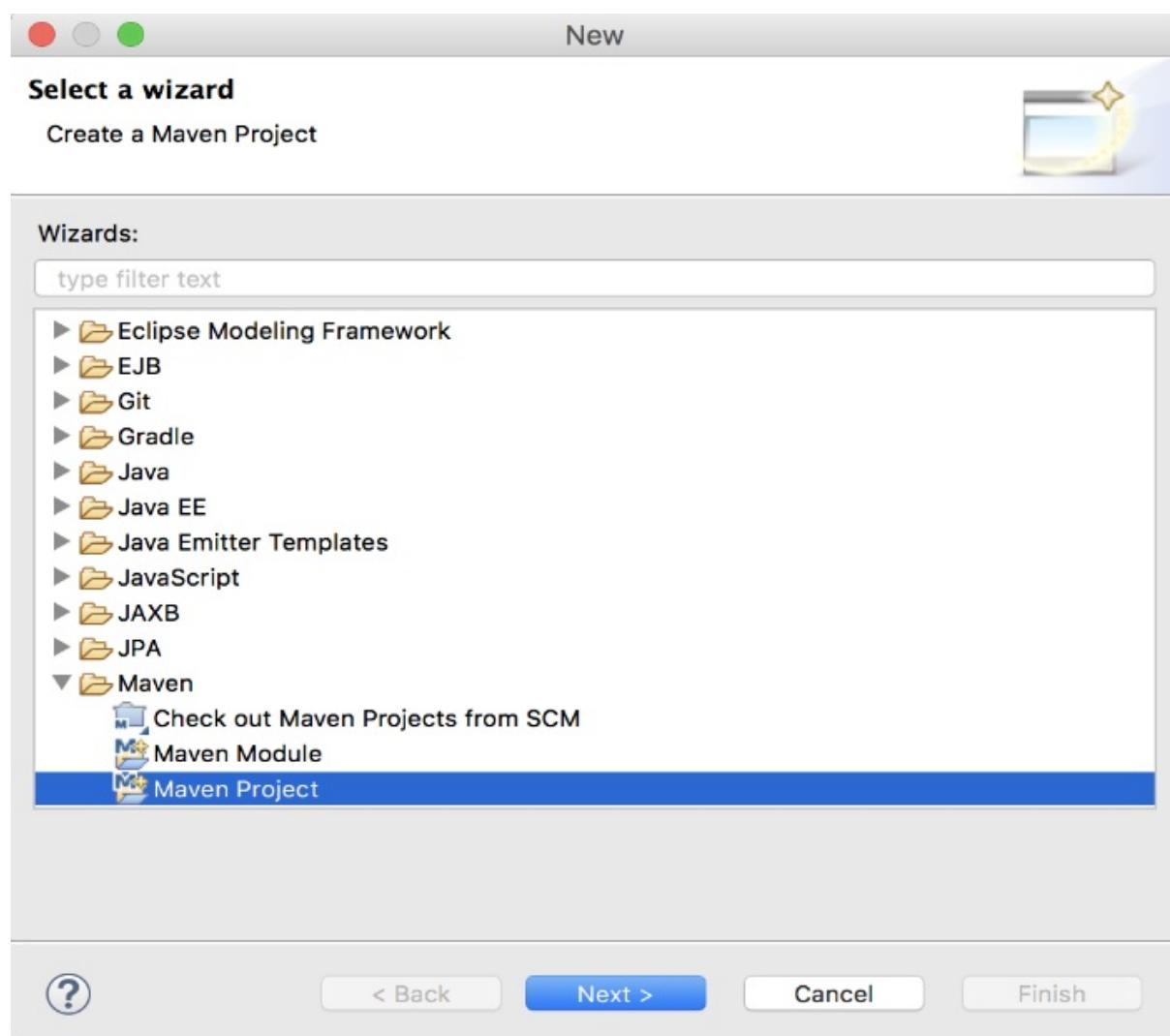
본 장에서는 Eclipse와 Maven을 이용하여 개발 환경을 구성하는 방법에 관해서 설명합니다.

지금부터 생성하는 프로젝트는 서블릿 기반의 리소스 서버를 생성하는 예제입니다.

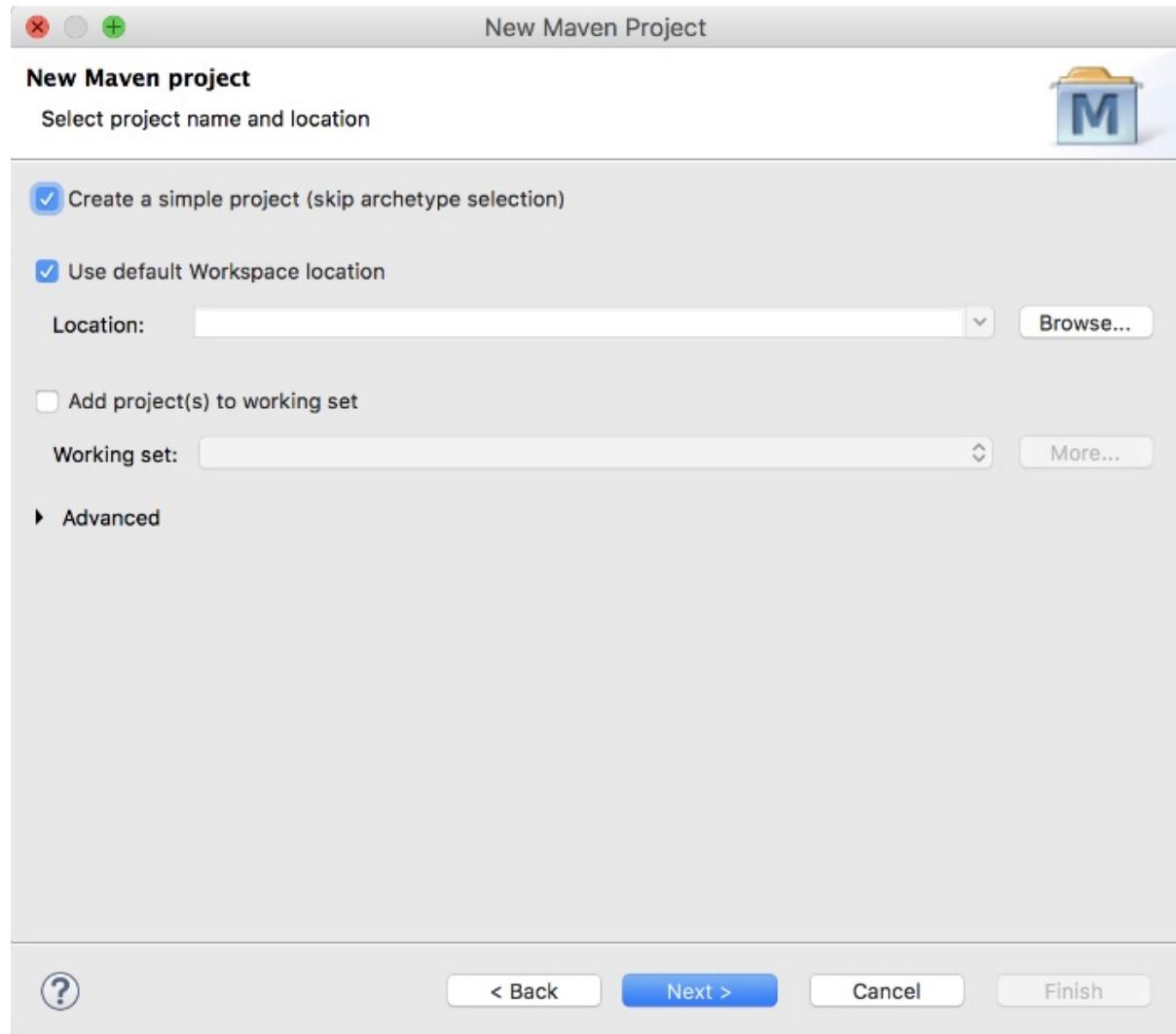
Eclipse에서 Maven 프로젝트 생성하기

개발 환경인 Eclipse에서 인증 모듈 개발을 위한 프로젝트를 생성하겠습니다.

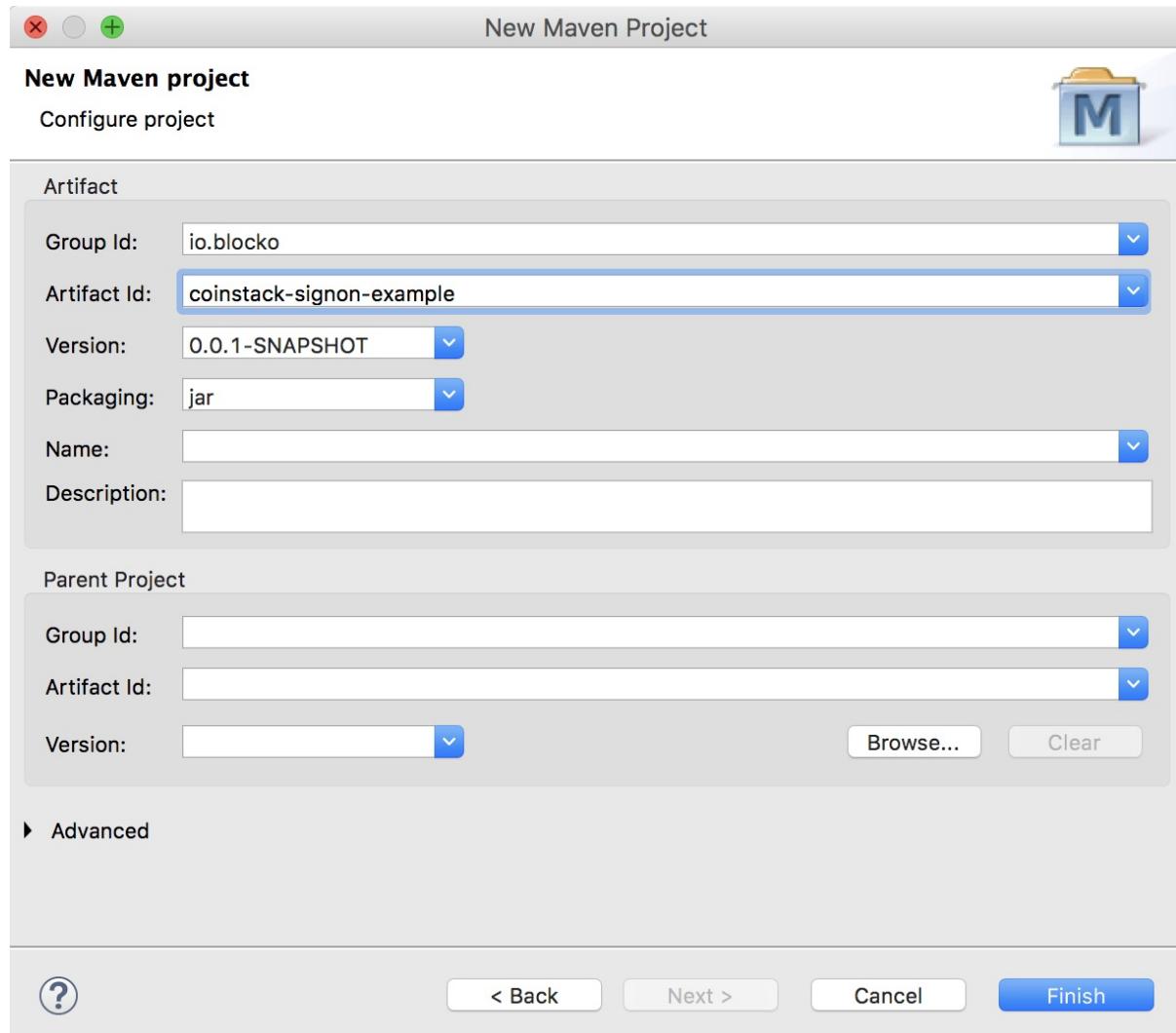
먼저 메뉴에서 File -> New를 선택하고, 리스트에서 **Maven Project**를 선택하고 다음으로 진행합니다.



Create a simple project를 체크하고, 다음 단계로 진행합니다.



<group-id>와 <artifact-id>를 입력하고, 과정을 종료하면 프로젝트가 생성됩니다.



pom.xml 설정하기

개발을 진행하기 위해서는 Maven 저장소에서 라이브러리들을 가져와야 합니다. BLOCKO, Inc는 개발에 필요한 모든 라이브러리를 인터넷을 통해 제공합니다. 인터넷이 단절된 환경에서 개발하기 위해서는 해당 라이브러리를 직접다운로드받아서 프로젝트를 구성할 수 있습니다. 관련 라이브러리를 프로젝트에 포함하기 위해서는 \${PROJECT_HOME}/pom.xml 파일을 수정해야 합니다.

라이브러리 추가

pom.xml에서는 라이브러리를 가져오기 위해서 Maven 저장소 정보를 입력하고, 필요한 라이브러리를 의존성에 정의합니다.

```

<properties>
    <project.build.sourceEncoding>utf-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>utf-8</project.reporting.outputEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
...
<repositories>
    <repository>
        <id>blocko-maven-release-repository</id>
        <url>https://nexus.blocko.io/repository/blocko-maven-release-repository/</url>
    </repository>
    <repository>
        <id>blocko-maven-snapshot-repository</id>
        <url>https://nexus.blocko.io/repository/blocko-maven-snapshot-repository/</url>
    </repository>
</repositories>
...
<dependencies>
```

```

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20080701</version>
</dependency>
</dependencies>
...

```

서블릿 실행 플러그인 추가

서블릿을 실행하기 위한 플러그인을 정의합니다.

개발 편의를 위해 1초마다 페이지 변경을 확인, context의 경로를 /로 지정, 사용자 지정 포트(8888) 등을 설정합니다.

```

...
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-eclipse-plugin</artifactId>
            <configuration>
                <downloadSources>true</downloadSources>
                <downloadJavadocs>true</downloadJavadocs>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-maven-plugin</artifactId>
            <version>9.0.5.v20130815</version>
            <configuration>
                <httpConnector>
                    <port>8888</port>
                </httpConnector>
                <scanIntervalSeconds>1</scanIntervalSeconds>
                <webApp>
                    <contextPath>/</contextPath>
                </webApp>
            </configuration>
        </plugin>
    </plugins>
</build>
...

```

서블릿 web.xml 파일 생성

서블릿 설정을 위해서는 web.xml 파일이 필요합니다. 파일의 경로는 \${PROJECT_HOME}/src/main/webapp/WEB-INF/web.xml입니다. Servlet 3.0부터 어노테이션을 사용한 설정이 가능하기 때문에 파일의 내용은 비워둡니다.

```

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">

    <servlet>
        <servlet-name>main_page</servlet-name>
        <jsp-file>/main_page.jsp</jsp-file>
    </servlet>
    <servlet>
        <servlet-name>user_page</servlet-name>
        <jsp-file>/user_page.jsp</jsp-file>
    </servlet>

```

```
</servlet>
<servlet>
  <servlet-name>admin_page</servlet-name>
  <jsp-file>/admin_page.jsp</jsp-file>
</servlet>

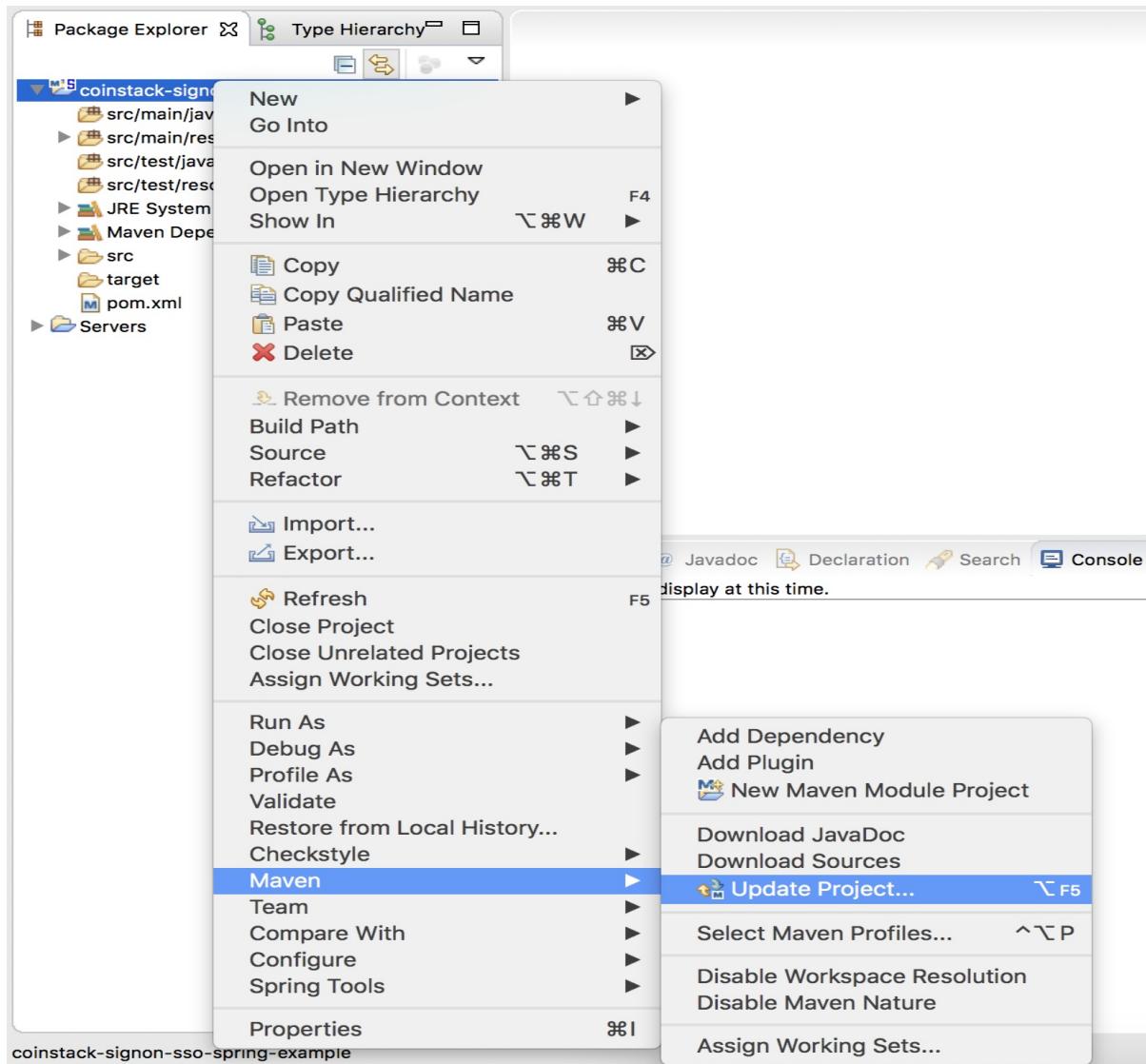
<servlet-mapping>
  <servlet-name>main_page</servlet-name>
  <url-pattern>/main_page</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>user_page</servlet-name>
  <url-pattern>/user_page</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>admin_page</servlet-name>
  <url-pattern>/admin_page</url-pattern>
</servlet-mapping>

<error-page>
  <error-code>403</error-code>
  <location>/403_page.jsp</location>
</error-page>

</web-app>
```

프로젝트 업데이트

pom.xml에 정의한 라이브러리들을 Maven 저장소로부터 가져오려면 해당 프로젝트를 우클릭 -> Maven -> Update Project...를 선택합니다.



선택을 완료하면, Maven 저장소로부터 자동적으로 로컬 저장소로 라이브러리들이 다운로드되어집니다.

Spring 프로젝트 설정

프로젝트 설정

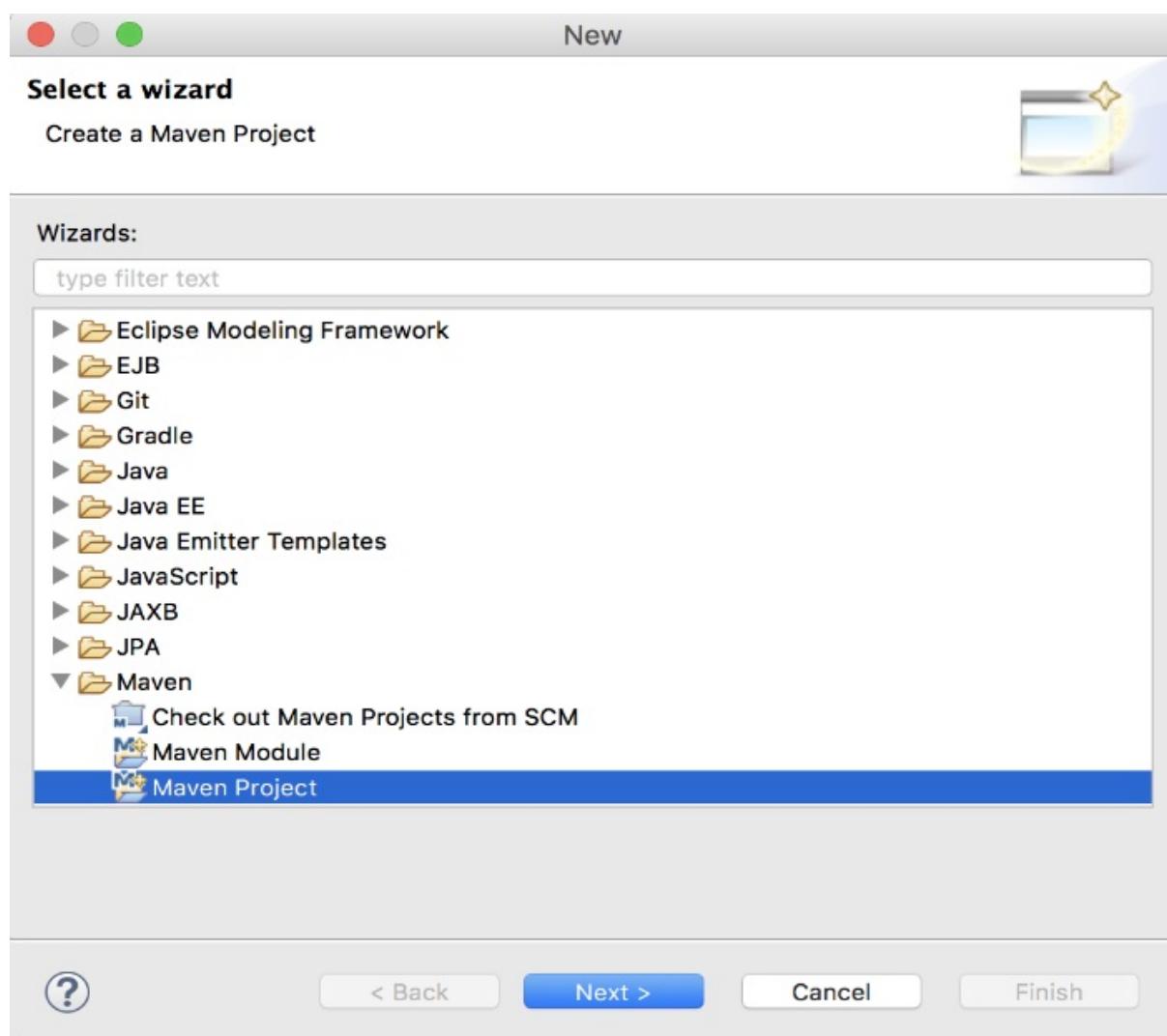
본 장에서는 Eclipse와 Maven을 이용하여 개발 환경을 구성하는 방법에 관해서 설명합니다.

지금부터 생성하는 프로젝트는 스프링 기반의 리소스 서버를 생성하는 예제입니다.

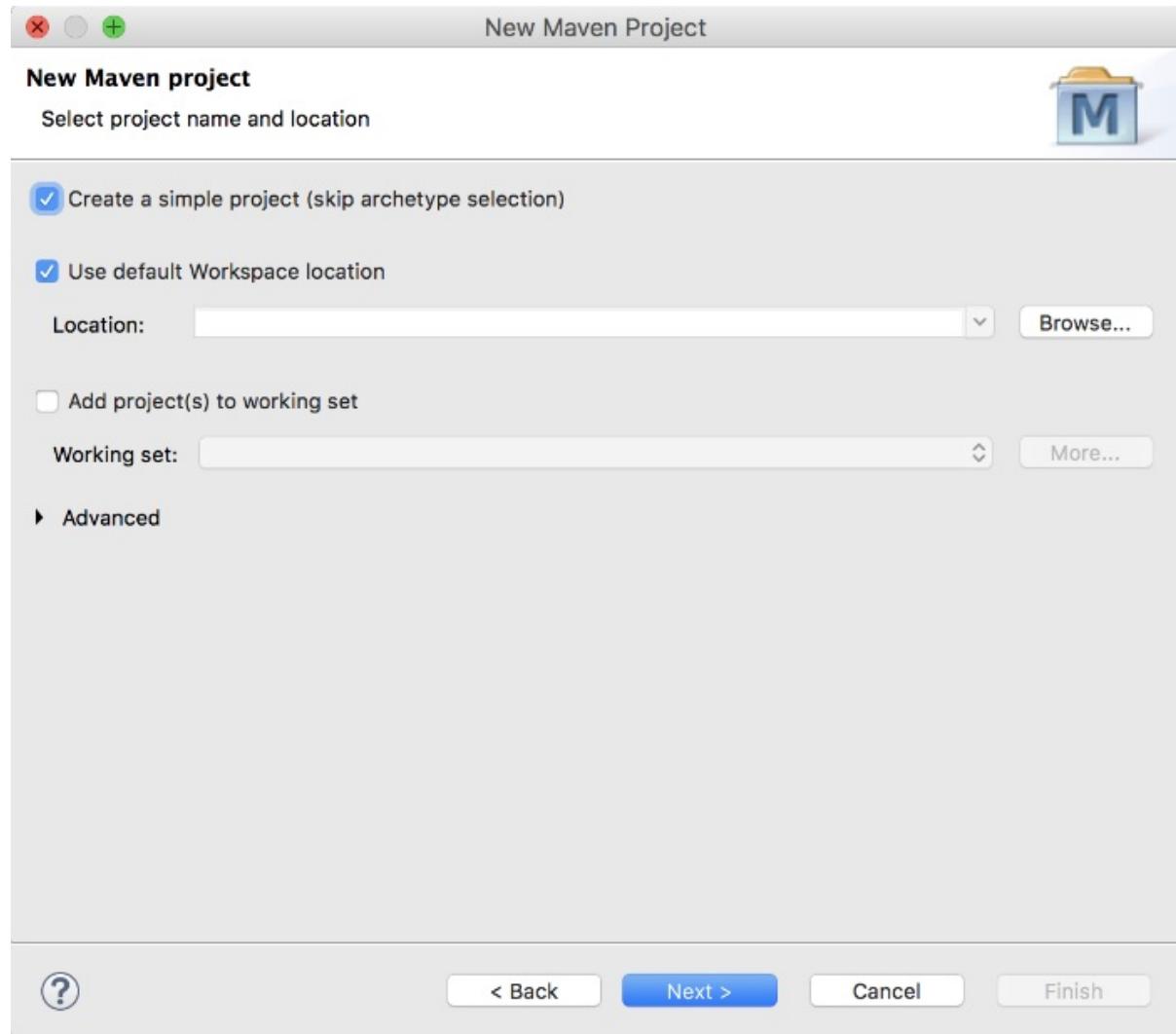
Eclipse에서 Maven 프로젝트 생성하기

개발 환경인 Eclipse에서 인증 모듈 개발을 위한 프로젝트를 생성하겠습니다.

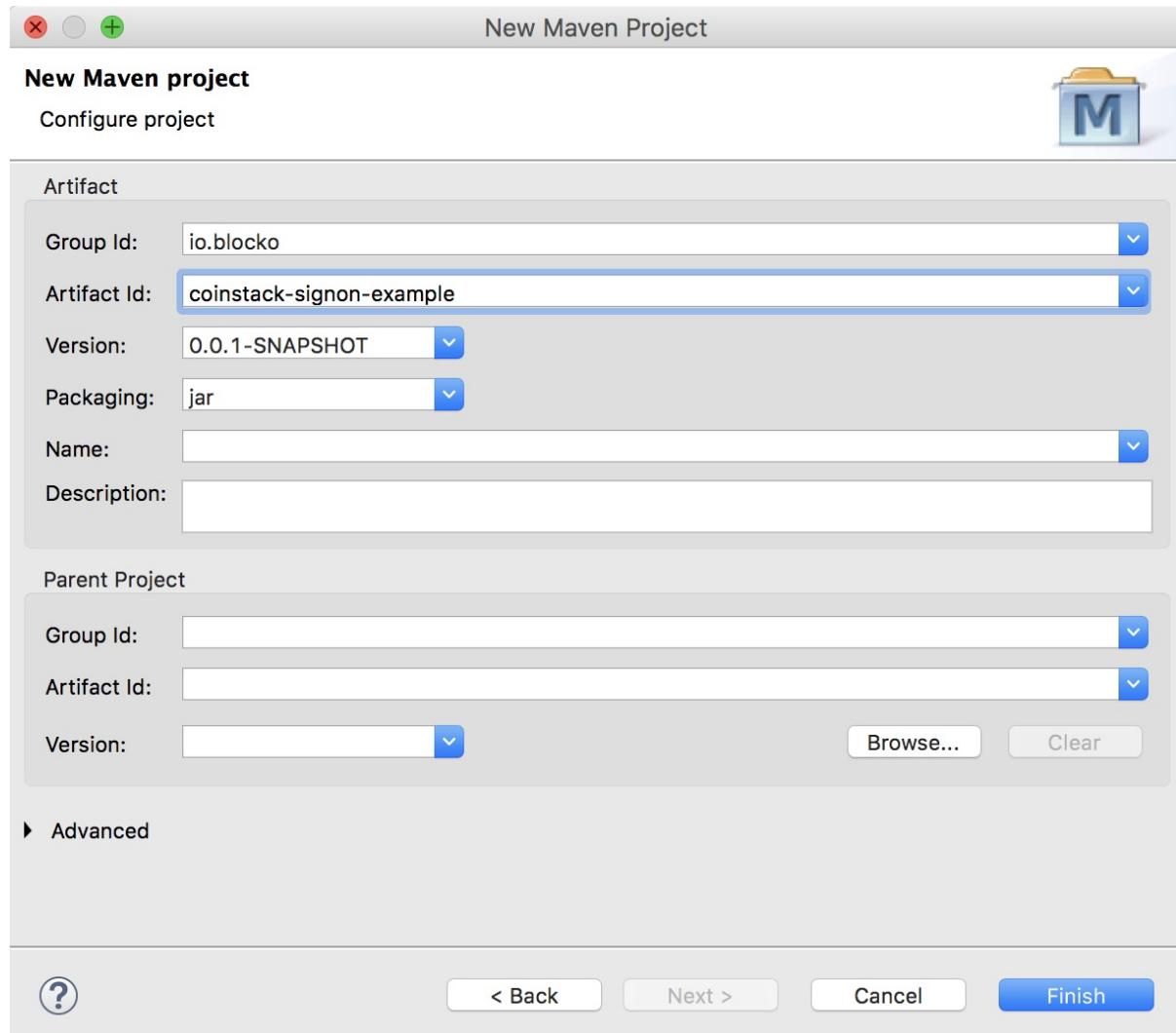
먼저 메뉴에서 File -> New를 선택하고, 리스트에서 **Maven Project**를 선택하고 다음으로 진행합니다.



Create a simple project를 체크하고, 다음 단계로 진행합니다.



<group-id>와 <artifact-id>를 입력하고, 과정을 종료하면 프로젝트가 생성됩니다.



pom.xml 설정하기

개발을 진행하기 위해서는 Maven 저장소에서 라이브러리들을 가져와야 합니다. BLOCKO, Inc는 개발에 필요한 모든 라이브러리를 인터넷을 통해 제공합니다. 인터넷이 단절된 환경에서 개발하기 위해서는 해당 라이브러리를 직접다운로드받아서 프로젝트를 구성할 수 있습니다. 관련 라이브러리를 프로젝트에 포함하기 위해서는 \${PROJECT_HOME}/pom.xml 파일을 수정해야 합니다.

라이브러리 추가

pom.xml에서는 라이브러리를 가져오기 위해서 Maven 저장소 정보를 입력하고, 필요한 라이브러리를 의존성에 정의합니다.

```

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.RELEASE</version>
    <relativePath /><!-- lookup parent from repository -->
</parent>
...
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
</properties>
...
<dependencies>
    <!-- json -->
    <dependency>
        <groupId>org.json</groupId>
        <artifactId>json</artifactId>
        <version>20080701</version>

```

```

</dependency>

<!-- spring boot -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!-- jsp -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>

<!-- spring security -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
</dependencies>

```

스프링 실행 플러그인 추가

스프링을 실행하기 위한 플러그인을 정의합니다.

```

...
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
...

```

스프링 application-development.yaml 파일 생성

스프링 설정을 위해서는 application-development.yaml 파일이 필요합니다. 파일의 경로는 \${PROJECT_HOME}/src/resource/application-development.yaml 입니다.

application-development.yaml

```

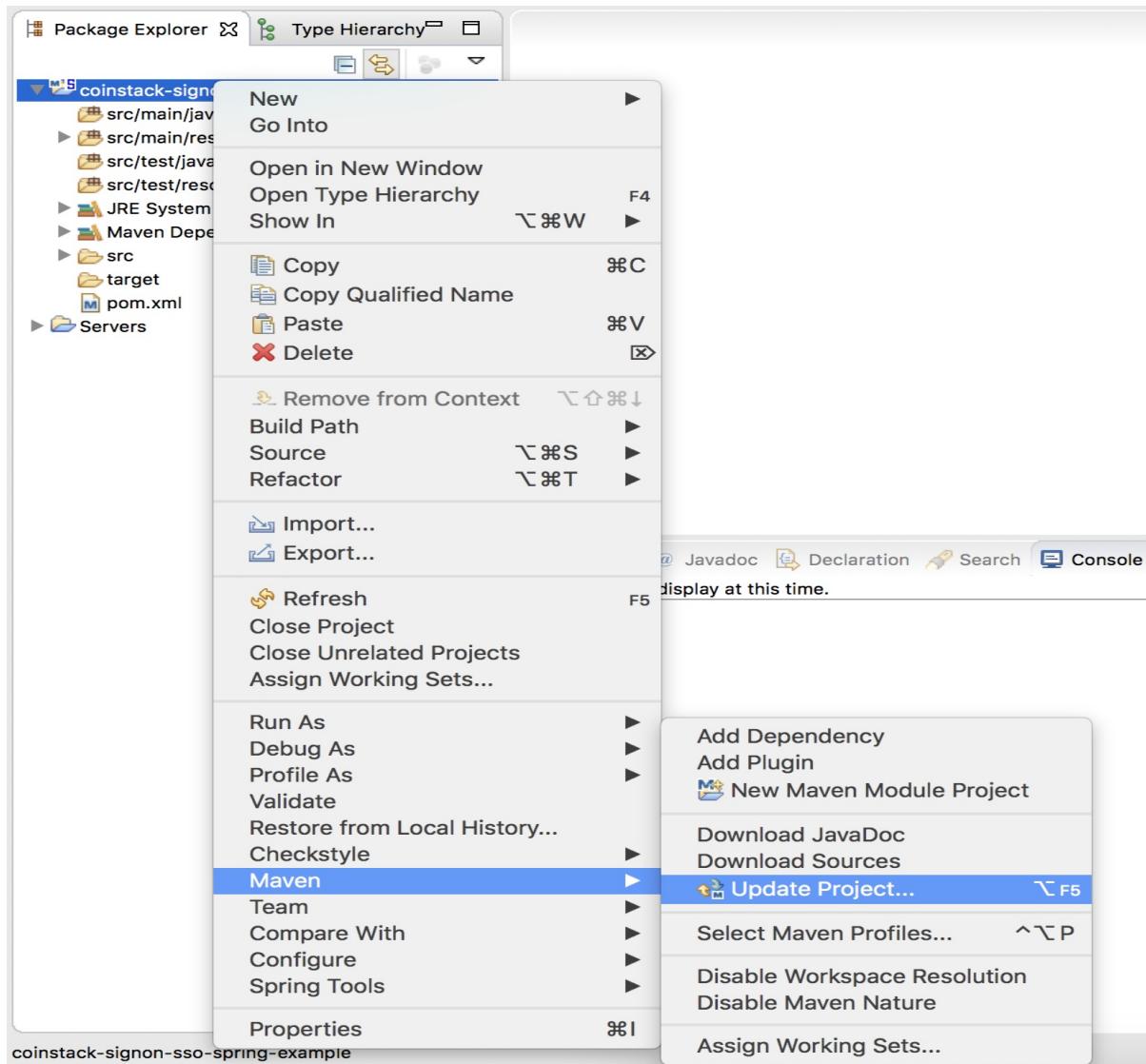
server:
  hostname: {$hostname}
  port : {$port}
spring:
  mvc:
    view:
      prefix: /
      suffix: .jsp

```

예제에서는 server.hostname=localhost:8080 , server.port=8888 을 사용하였습니다.

프로젝트 업데이트

pom.xml에 정의한 라이브러리들을 Maven 저장소로부터 가져오려면 해당 프로젝트를 우클릭 -> Maven -> Update Project...를 선택합니다.



선택을 완료하면, Maven 저장소로부터 자동적으로 로컬 저장소로 라이브러리들이 다운로드되어집니다.

WebApplication.java

Controller에서 각각 URI로 들어온 내용들을 jsp 페이지로 연결해줍니다.

```

@Controller
@SpringBootApplication
public class WebApplication {
    @RequestMapping("/main_page")
    public String mainPage(){
        return "main_page";
    }

    @RequestMapping("/admin_page")
    public String adminPage() {
        return "admin_page";
    }

    @RequestMapping("/user_page")
    public String userPage() {
        return "user_page";
    }

    @Bean
    public ErrorPageRegistrar errorPageRegistrar() {
        return new ErrorPageRegistrar() {
            @Override public void registerErrorPages(ErrorPageRegistry registry) {
                registry.addErrorPages(new ErrorPage(HttpStatus.FORBIDDEN, "/403_page.jsp"));
            }
        };
    }
}

```

```
    };
}

public static void main(String[] args) {
    SpringApplication.run(WebApplication.class, args);
}
}
```

WebSecurityConfig.java

spring security에서 제공하는 필터를 사용하지 않고 직접 만든 SSOFilter를 사용 합니다.

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override protected void configure(HttpSecurity http) throws Exception {
        http.httpBasic().disable() .logout().disable()
            .addFilterAfter(new SSOFilter(), AbstractPreAuthenticatedProcessingFilter.class);
    }
}
```

리소스 서버 만들기

환경 설정

본 예제를 실행하기 위해 환경 설정은 다음과 같은 순서로 진행합니다.

클라이언트 등록

```
$ coinstack-signon client create --privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "clientId": "trusted",
  "clientSecret": "secret",
  "authorizedGrantTypes": ["authorization_code"],
  "scopes": ["read", "write"],
  "registeredRedirectUris": ["http://localhost:8888"]
}
EOF
```

사용자 등록

```
$ coinstack-signon user create --privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "username": "user",
  "password": "password",
  "authorities": ["USER"]
}
EOF
```

SignOn 서버 구동

```
$ coinstack-signon server start
```

이에 대한 자세한 설정은 [서버](#), [클라이언트](#), [사용자](#)를 참조하시기 바랍니다.

리소스 만들기

사용자가 접근하는 리소스(페이지)를 작성합니다.

리소스 파일은 다음과 같으며, \${PROJECT_HOME}/src/main/webapp 디렉터리에 생성합니다.

main_page.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Main page</title>
</head>
<body>
  <b>메인 페이지</b><br>
  Coinstack SSO 예제에 오신걸 환영합니다.
</body>
</html>
```

페이지 확인

페이지 확인을 위해 서버를 구동합니다. \${PROJECT_HOME}에서 다음의 명령어를 실행합니다.

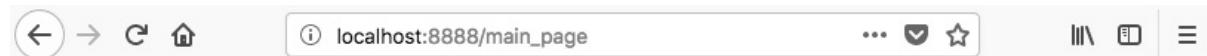
Servlet

```
$ mvn jetty:run
```

Spring

```
$ mvn spring-boot:run -Dspring.profiles.active=development
```

서버가 구동되면 브라우저를 이용해서 http://localhost:8888/main_page.jsp에 접근해봅니다.



Client 정보

다음은 사용자 인증에 필요한 데이터 목록입니다.

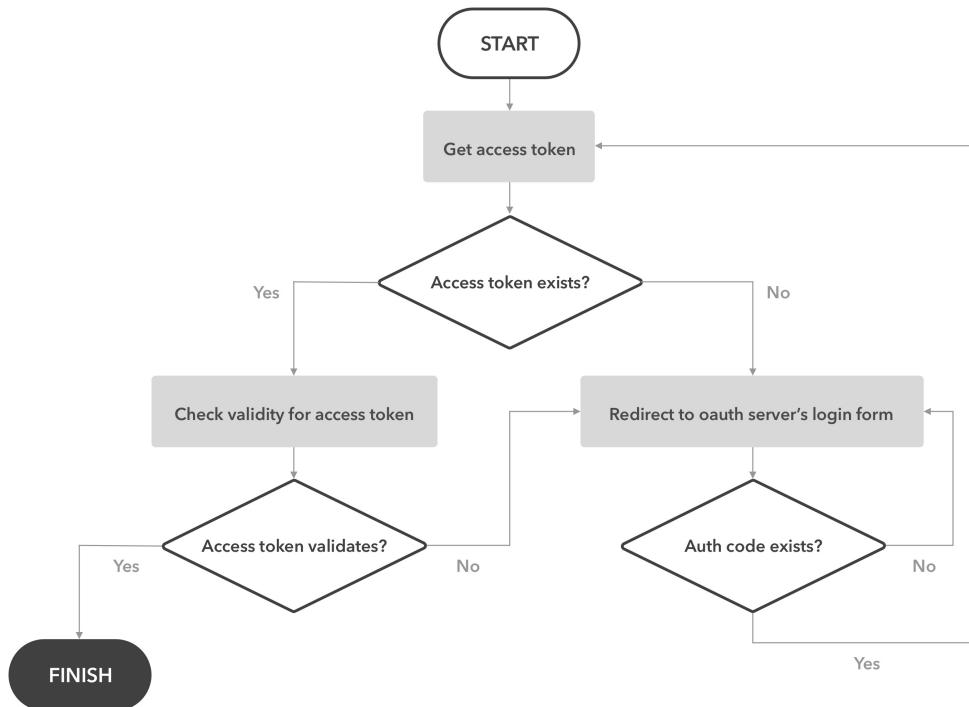
서블릿 필터 만들기

@WebFilter는 필터를 적용할 범위를 지정할 수 있습니다.

예제 코드에서는 "*" 로 설정하며 Request로 들어오는 모든 URI에 대해 필터를 적용하였습니다.

SSO에서 필터의 흐름과 이에 따른 코드는 다음과 같습니다.

Servlet Filter Flow

**SSOFilter.java**

```

@WebFilter("*")
public class SSOFilter implements Filter {
    private AuthService authService = new AuthService();

    @Override
    public void init(final FilterConfig filterConfig) throws ServletException {
        System.out.println("Initializing filter...");
    }

    @Override
    public void doFilter(final ServletRequest request, final ServletResponse response,
                        final FilterChain chain) throws IOException, ServletException {

        // Common Request, Response
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        try {
            // 1. Get access token
            if (authService.hasAccessToken(req)) {
                // 2. Check validity for access token
                if (authService.verifyAccessToken(req)) {
                    chain.doFilter(req, res);
                } else {
                    authService.removeAccessTokenFromCookie(req, res);
                    authService.requestAuthCode(req, res);
                }
            } else {
                // 3. Redirect to oauth server's login form
                if (req.getParameter("code") != null) {
                    authService.addAccessTokenToCookie(req, res);
                    res.sendRedirect(req.getRequestURI());
                } else {
                    authService.requestAuthCode(req, res);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

        }

    }

    @Override
    public void destroy() {
        System.err.println("Destroying filter...");
    }
}

```

AuthService.java

Authorization Service를 지원하는 클래스로 SSO를 제공하기 위해 다음과 같은 메서드들을 사용합니다.

```

public class AuthService {
    private static final String CLIENT_ID = "trusted"; // 클라이언트 ID
    private static final String SECRET = "secret"; // 클라이언트 비밀번호
    private static final String GRANT_TYPE = "authorization_code"; // 권한 취득 방식
    private static final String SCOPE = "read"; // 접근 제어 범위
    private static final String RESPONSE_TYPE = "code"; // 인증서버 응답 방식
    private static final String SERVER_DOMAIN = "http://localhost:8888"; // SSO 서버의 주소
    private static final String ENDPOINT = "http://localhost:8080"; // OAuth 서버의 주소

    // ===== SSO methods ===== //
    // 1. hasAccessToken
    // 2. findAccessTokenFromCookie
    // 3. addAccessTokenToCookie
    // 4. requestAuthCode
    // 5. verifyAccessToken
    // 6. getAccessTokenInfo
    // 7. removeAccessTokenFromCookie
    // ===== //
}

```

hasAccessToken

Access token이 있는지 판단해주는 메서드입니다.

`findAccessTokenFromCookie` 메서드 반환값 통해 Access token 존재 여부를 확인해줍니다.

```

/**
 * AccessToken 유무 확인.
 *
 * @param req HttpServletRequest
 * @return
 */
public boolean hasAccessToken(HttpServletRequest req) {
    boolean result = false;
    if (findAccessTokenFromCookie(req) != null) {
        // accessToken이 있는 경우
        result = true;
    }
    return result;
}

```

findAccessTokenFromCookie

Cookie 값 중 Access token 을 찾아주는 메소드입니다.

값이 없는 경우 `null` 값을 반환합니다.

```

/**
 * Cookie에서 accessToken값을 찾는 함수.
 *
 * @param req HttpServletRequest
 * @return
 */
private String findAccessTokenFromCookie(HttpServletRequest req) {
    String result = null;
    Cookie[] cookies = req.getCookies();
    if (null != cookies) {
        for (int i = 0; i < cookies.length; i++) {
            if ("accessToken".equals(cookies[i].getName())) {
                result = cookies[i].getValue();
            }
        }
    }
}

```

```

        }
    }
    return result;
}

```

addAccessTokenToCookie

Access token을 받아와 Cookie에 추가해주는 메서드입니다.

Login form으로 리다이렉션 후 Resource owner가 로그인할 때 발급되는 Auth code, Grant type, Redirectionuri를 SignOn 서버의 /oauth/token으로 Post 하여 JSONObject 형태의 Access token을 Cookie에 추가합니다.

Auth code가 존재하면 Auth code에 따른 Access token을 Cookie에 추가하는 역할을 합니다.

따라서 Access token을 Cookie에 등록 및 공유하여 SSO 로그인을 가능하게 합니다.

```

/**
 * Access token을 가져와 Cookie에 추가하는 메서드.
 * @param authCode String
 * @param req HttpServletRequest
 * @param res HttpServletResponse
 */
public void addAccessTokenToCookie(HttpServletRequest req, HttpServletResponse res)
    throws Exception {
    String getToken = ENDPOINT + "/oauth/token";
    String data = "grant_type=" + GRANT_TYPE + "&redirect_uri=" + SERVER_DOMAIN
        + req.getRequestURI() + "&code=" + req.getParameter("code");
    String clientInfo = CLIENT_ID + ":" + SECRET;
    String accessInfo = sendPost(getToken, data, clientInfo);
    if (accessInfo != null) {
        JSONObject accessInfoJson = new JSONObject(accessInfo);

        Cookie cookie = new Cookie("accessToken", accessInfoJson.getString("access_token"));
        res.addCookie(cookie);
    }
}

```

requestAuthCode

Auth code를 요청하는 메서드입니다.

접근 하려는 사용자가 Access token 이 없는 경우 SignOn 서버의 oauth/authorize로 리다이렉션 하여 권한 인증을 하게합니다.

권한 인증의 결과값은 ?code= 형태로 반환됩니다.

```

/**
 * authCode 요청 (loginPage 리다이렉션).
 *
 * @param req HttpServletRequest
 * @param res HttpServletResponse
 * @throws IOException IOException
 */
public void requestAuthCode(HttpServletRequest req, HttpServletResponse res) throws IOException {
    res.sendRedirect(ENDPOINT + "/oauth/authorize?response_type=" + RESPONSE_TYPE + "&grant_type="
        + GRANT_TYPE + "&scope=" + SCOPE + "&client_id=" + CLIENT_ID + "&secret=" + SECRET
        + "&redirect_uri=" + SERVER_DOMAIN + req.getRequestURI().toString());
}

```

verifyAccessToken

Access token이 있는지 유효한지 판단해주는 메서드입니다.

getAccessTokenInfo 메서드 반환값 통해 Access token 유효여부를 확해줍니다.

```

/**
 * accessToken 증명.
 *
 * @param req HttpServletRequest
 * @throws IOException IOException
 */
public boolean verifyAccessToken(HttpServletRequest req) throws IOException {
    boolean result = false;

```

```

if (getAccessTokenInfo(req) != null) {
    // accessToken이 유효한 경우
    result = true;
}
return result;
}

```

getAccessTokenInfo

Access token의 등록정보를 가져오는 메서드입니다.

SignOn 서버의 /oauth/check_token 으로 엑세스 토큰값을 POST하여 JSONObject 형태의 등록정보를 가져옵니다.

엑세스 토큰값이 유효하지 않은경우 null값을 반환합니다.

```

/**
 * accessToken 정보를 읽어오는 메서드.
 *
 * @param req HttpServletRequest
 * @throws Exception Exception
 */
private JSONObject getAccessTokenInfo(HttpServletRequest req) throws Exception {
    String checkAt = "token=" + findAccessTokenFromCookie(req);
    String auth = CLIENT_ID + ":" + SECRET;
    String checkToken = ENDPOINT + "/oauth/check_token";
    JSONObject result = null;

    String response = sendPost(checkToken, checkAt, auth);
    if (response != null) {
        result = new JSONObject(response);
    }
    return result;
}

```

removeAccessTokenFromCookie

Access token이 유효하지 않은 경우 Cookie에서 Access token을 제거하는 메서드입니다.

Cookie의 setMaxAge 메서드를 통해 Cookie의 유효기간을 0으로 설정하여 유효하지 않은 Access token을 제거합니다.

```

/**
 * AccessToken이 유효하지 않은 경우 Cookie값을 삭제하는 메서드.
 * @param req HttpServletRequest
 * @param res HttpServletResponse
 */
public void removeAccessTokenFromCookie(HttpServletRequest req, HttpServletResponse res)
    throws Exception {
    for (Cookie cookie : req.getCookies()) {
        cookie.setMaxAge(0);
        res.addCookie(cookie);
    }
}

```

sendPost

SignOn 서버의 /oauth/token이나 /oauth/check_token으로 HTTP POST 요청 및 응답을 하는 메서드입니다.

/oauth/token으로 POST 요청을 하여 String 형태의 Access token을 반환합니다.

/oauth/check_token으로 POST 요청을 하여 String 형태의 User info를 반환합니다.

```

/**
 * HTTP POST 요청/응답 메서드.
 * @param uri String
 * @param params String
 * @param clientInfo String
 * @return data String
 */
private String sendPost(String uri, String params, String clientInfo) throws Exception {

    URL url = new URL(uri);
    HttpURLConnection con = (HttpURLConnection) url.openConnection();
    if (clientInfo != null) {
        String userpass = clientInfo;
    }
}

```

```

String basicAuth =
    "Basic " + javax.xml.bind.DatatypeConverter.printBase64Binary(userpass.getBytes());
con.setRequestProperty("Authorization", basicAuth);
}
con.setRequestMethod("POST");
con.setDoOutput(true);

PrintWriter pw = new PrintWriter(new OutputStreamWriter(con.getOutputStream(), "UTF-8"));
pw.write(params);
pw.flush();

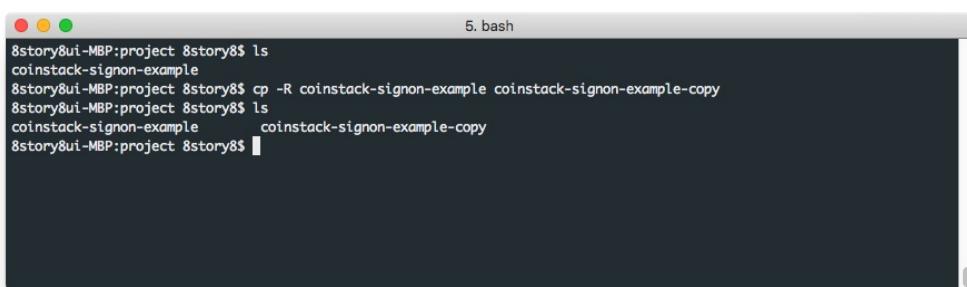
String line;
String data = "";
String result = null;

if (con.getResponseCode() != 400) {
    BufferedReader br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
    while ((line = br.readLine()) != null) {
        data += line + "\n";
    }
    pw.close();
    br.close();
    result = data;
}
return result;
}

```

실행 결과

테스트를 위해 다음과 같이 프로젝트를 복사합니다.



Servlet

복사한 프로젝트의 포트 번호를 8889로 다음과 같이 변경합니다.

"pom.xml"에서 포트를 변경합니다.

```

<httpConnector>
    <port>8889</port>
</httpConnector>

```

이후, 두 서버를 구동하기 위해 각각의 \${PROJECT_HOME}에서 다음의 명령어를 실행합니다.

Servlet

```
$ mvn jetty:run
```

Spring

복사한 프로젝트의 포트 번호를 8889로 다음과 같이 변경합니다.

"application-development.yaml"에서 포트를 변경합니다.

```

server:
  ...
  port : 8889

```

이후, 두 서버를 구동하기 위해 각각의 \${PROJECT_HOME}에서 다음의 명령어를 실행합니다.

Spring

```
$ mvn spring-boot:run -Dspring.profiles.active=development
```

localhost:8888/main_page.jsp로 접속하면 SignOn 서버의 로그인 페이지로 리다이렉션됩니다.

The screenshot shows a web browser window with the URL 'localhost:8080/login'. The page title is 'Login with Username and Password'. It contains two input fields: 'User:' with placeholder text 'user' and 'Password:' with placeholder text '*****'. Below the fields is a 'Login' button.

이후, 등록한 사용자 인증 정보로 다음과 같이 로그인을 합니다.

The screenshot shows the same 'Login with Username and Password' page at 'localhost:8080/login'. The 'User:' field now contains 'user' and the 'Password:' field contains '*****'. The 'Login' button is visible below the fields.

로그인이 성공하면 localhost:8888/main_page.jsp로 리다이렉션되는 것을 확인할 수 있습니다.

The screenshot shows a web browser window with the URL 'localhost:8888/main_page'. The page title is '메인 페이지'. The content area displays the text 'Coinstack SSO 예제에 오신걸 환영합니다.'

이후, localhost:8889/main_page.jsp로 접속하면 별도의 인증 절차 없이 메인 페이지를 확인할 수 있습니다.

The screenshot shows a web browser window with the URL 'localhost:8889/main_page'. The page title is '메인 페이지'. The content area displays the text 'Coinstack SSO 예제에 오신걸 환영합니다.'

권한 제어

환경 설정

본 예제를 실행하기 위해 환경 설정은 다음과 같은 순서로 진행합니다.

클라이언트 등록

```
$ coinstack-signon client create --privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "clientId": "trusted",
  "clientSecret": "secret",
  "authorizedGrantTypes": ["authorization_code"],
  "scopes": ["read", "write"],
  "registeredRedirectUris": ["http://localhost:8888"]
}
EOF
```

사용자 등록

```
$ coinstack-signon user create --privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "username": "admin",
  "password": "admin",
  "authorities": ["ADMIN"]
}
EOF

$ coinstack-signon user create --privatekey ${ADMIN_PRIVATEKEY} <<EOF
{
  "username": "user",
  "password": "password",
  "authorities": ["USER"]
}
EOF
```

SignOn 서버 구동

```
$ coinstack-signon server start
```

이에 대한 자세한 사항은 [서버](#), [클라이언트](#), [사용자](#)를 참조하시기 바랍니다.

리소스 만들기

사용자가 접근하는 리소스(페이지)를 작성합니다.

리소스 파일은 다음과 같으며, \${PROJECT_HOME}/src/main/webapp 디렉터리에 생성합니다.

user_page.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/htm14/loose.dtd">
<html>
<head>
  <title>User page</title>
</head>
<body>
  <b>사용자 페이지</b><br>
  USER, ADMIN 권한을 가진 사람이 접근할 수 있습니다.
</body>
</html>
```

admin_page.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>Admin page</title>
</head>
<body>
    <b>관리자 페이지</b><br>
    ADMIN 권한을 가진 사람만 접근할 수 있습니다.
</body>
</html>
```

403_page.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <title>403 page</title>
</head>
<body>
    <b>403 페이지</b><br>
    접근할 수 없는 페이지입니다.
</body>
</html>
```

리소스별 접근 권한

생성한 페이지에 대한 접근 권한은 다음과 같습니다.

사용자	user_page.jsp	admin_page.jsp
admin	O	O
user	O	접근 권한이 없으므로 403_page.jsp로 이동

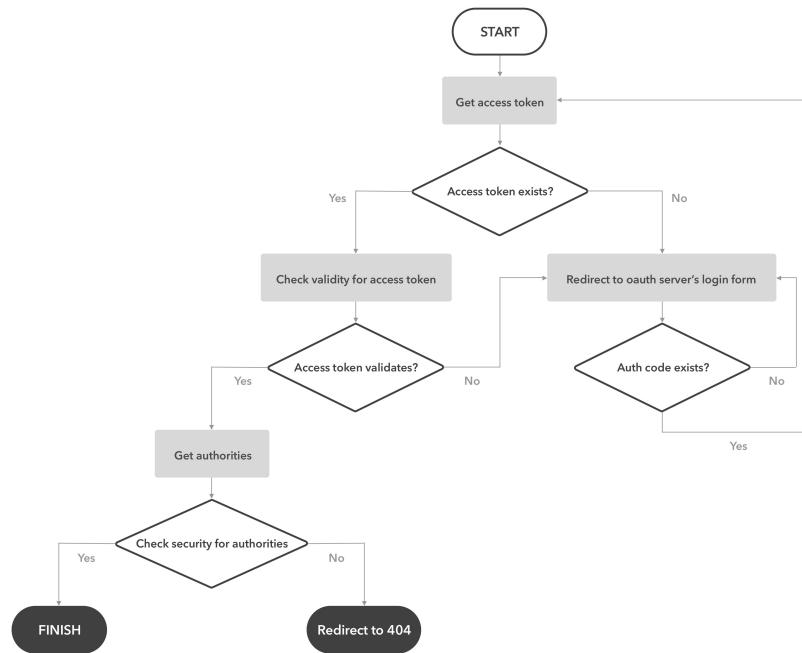
권한 제어 서블릿 필터 만들기

다음의 코드는 SSO 서블릿 필터 코드에 권한 제어 코드를 추가한 것입니다.

SSO 서블릿 필터 코드와 관련된 설명, 설정은 [Servlet 기반의 리소스 서버 만들기](#)를 참조하시기 바랍니다.

권한 제어에서 필터의 흐름과 이에 따른 코드는 다음과 같습니다.

Authorization Control Servlet Filter Flow



SSOFilter.java

```

@WebFilter("*")
public class SSOFilter implements Filter {
    private AuthService authService = new AuthService();

    @Override
    public void init(final FilterConfig filterConfig) throws ServletException {
        System.out.println("Initializing filter...");
    }

    @Override
    public void doFilter(final ServletRequest request, final ServletResponse response,
                        final FilterChain chain) throws IOException, ServletException {

        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        try {
            if (authService.hasAccessToken(req)) {
                if (authService.verifyAccessToken(req)) {
                    // 1. Check authorities
                    if (authService.checkAuthorities(req)) {
                        chain.doFilter(req, res);
                    } else {
                        res.sendRedirect("/403_page.jsp");
                    }
                } else {
                    authService.removeAccessTokenFromCookie(req, res);
                    authService.requestAuthCode(req, res);
                }
            } else {
                if (req.getParameter("code") != null) {
                    authService.addAccessTokenToCookie(req, res);
                    res.sendRedirect(req.getRequestURI());
                } else {
                    authService.requestAuthCode(req, res);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
  
```

```

@Override
public void destroy() {
    System.err.println("Destroying filter...");
}
}

```

AuthService.java

Authorization service를 지원해주는 클래스로 권한 제어를 제공하기 위해 다음과 같은 메서드를 사용합니다.

```

public class AuthService {
    private static final String CLIENT_ID = "trusted"; // 클라이언트 ID
    private static final String SECRET = "secret"; // 클라이언트 비밀번호
    private static final String GRANT_TYPE = "authorization_code"; // 권한 취득 방식
    private static final String SCOPE = "read"; // 접근 제어 범위
    private static final String RESPONSE_TYPE = "code"; // 인증서버 응답 방식
    private static final String SERVER_DOMAIN = "http://localhost:8888"; // SSO 서버의 주소
    private static final String ENDPOINT = "http://localhost:8080"; // OAuth 서버의 주소

    // ===== SSO methods ===== //
    // 1. hasAccessToken
    // 2. findAccessTokenFromCookie
    // 3. addAccessTokenToCookie
    // 4. requestAuthCode
    // 5. verifyAccessToken
    // 6. getAccessTokenInfo
    // 7. removeAccessTokenFromCookie
    // ===== //

    // ===== Access Control methods ===== //
    // 1. checkAuthorities
    // 2. getAuthorities
    // 3. arrangeAuthorities
    // ===== //
}

```

checkAuthorities

사용자의 권한을 체크해주는 메서드입니다.

getAuthorities 메서드 반환값을 통해 사용자가 접근하려는 페이지에 접근 권한이 있는지 판단합니다.

접근 권한이 없는 페이지에 접근하는 경우 false 값을 반환합니다.

```

/**
 * 권한체크 메서드.
 *
 * @param req HttpServletRequest
 * @throws Exception Exception
 */
public boolean checkAuthorities(HttpServletRequest req) throws Exception {
    Boolean result = false;
    String[] authorities = getAuthorities(req);
    if ("/admin_page.jsp".equals(req.getRequestURI())) {
        for (String authority : authorities) {
            if (authority.equals("ADMIN")) {
                result = true;
                break;
            }
        }
    } else {
        result = true;
    }

    return result;
}

```

getAuthorities

Access token의 권한정보를 가져오는 메서드입니다.

getAccessTokenInfo 메서드의 반환값 중 authorities를 추출하고 arrangeAuthorities 메서드를 이용해 정리하여 반환합니다.

```

    /**
     * accessToken 권한정보를 가져오는 메서드.
     *
     * @param req HttpServletRequest
     * @throws Exception Exception
     */
    private String[] getAuthorities(HttpServletRequest req) throws Exception {
        JSONObject accessTokenInfo = getAccessTokenInfo(req);
        return arrangeAuthorities(accessTokenInfo.getString("authorities"));
    }
}

```

arrangeAuthorities

String 형태의 Authorities를 String 배열로 정돈시키는 메서드입니다.

String 형태의 Authorities를 정규 표현식으로 정돈하여 String 배열 형태로 Authorities를 반환합니다.

```

    /**
     * String으로 넘어오는 authorities를 array로 깔끔하게 만들어 주는 메서드.
     *
     * @param authorities authorities
     * @return
     */
    private String[] arrangeAuthorities(String authorities) {
        // beautify userAuth
        String[] authList;
        authList = authorities.split(",");
        for (int i = 0; i < authList.length; i++) {
            authList[i] = authList[i].replaceAll("[\\\\\\\\\\\\\\\\\\\\\\\\]", "");
        }
        return authList;
    }
}

```

실행 결과

사용자 페이지

localhost:8888/user_page.jsp로 접속하면 SignOn 서버에서 권한을 체크하고 로그인 페이지로 리다이렉션합니다.

이후 관리자 계정인 admin과 일반 사용자 계정인 user로 각각 로그인합니다.

User: user
Password: *****
Login

이처럼 user_page.jsp는 모든 사용자가 접근할 수 있습니다.

사용자 페이지
USER, ADMIN 권한을 가진 사람이 접근할 수 있습니다.

관리자 페이지

localhost:8888/admin_page.jsp로 접속하면 SignOn 서버에서 권한을 체크하고 로그인 페이지로 리다이렉션합니다.

이후 admin_page.jsp의 접근 권한이 없는 user로 로그인합니다.

User: user
Password: *****
Login

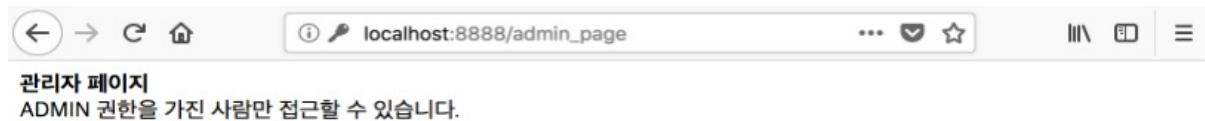
user는 admin_page.jsp의 접근 권한을 갖고 있지 않기 때문에 403_page.jsp로 리다이렉션합니다.

403 페이지
접근할 수 없는 페이지입니다.

admin_page.jsp의 접근 권한을 가진 admin으로 로그인합니다.

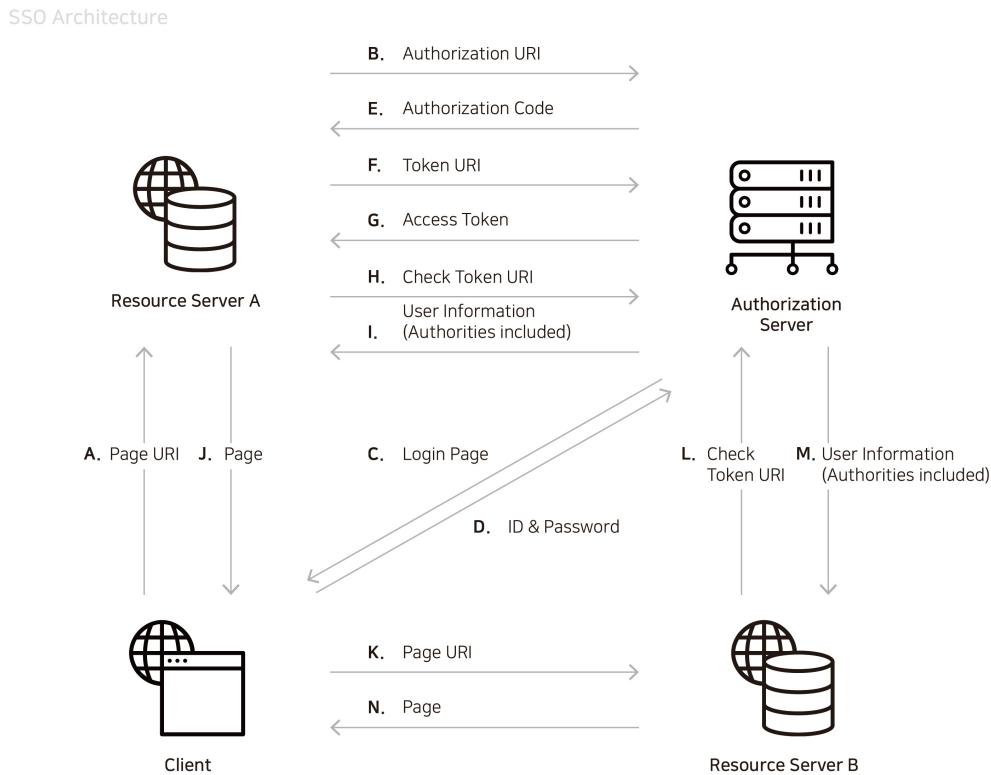
User: admin
Password: *****
Login

admin은 admin_page.jsp의 접근 권한을 갖고 있기 때문에 해당 페이지로 리다이렉션합니다.



시스템 구성

SSO 시스템 구성도는 다음과 같습니다.



- A. Client는 Resource server A에 접속하기 위해 Page URI를 전송합니다.
- B. 접속 요청을 받은 Resource server A는 Authorization server에 인증을 받기 위해 Authorization URI를 전송합니다.
- C. Authorization server는 인증을 위해 Client에게 Login page를 제공합니다.
- D. Client는 ID & Password를 Authorization server에 전송하여 로그인합니다.
- E. Authorization server는 Resource server A에게 Authorization code를 발급합니다.
- F. Resource server A는 Authorization code를 Token URI에 포함해 Authorization server에 전송합니다.
- G. Authorization server는 Token URI를 활용하여 Access token을 Resource server A에 발급합니다.
- H. Resource server A는 Access token을 Check token URI에 포함해 Authorization server에 전송하여 유효성을 검증합니다.
- I. Authorization server는 Authorities를 포함한 User information을 Resource server A에 전송합니다.
- J. Resource server A는 사용자의 접근 권한에 따른 페이지(user_page.jsp, admin_page.jsp, 404_page.jsp)를 Client에게 제공합니다.
- K. Client는 Resource server B에 접속하기 위해 Page URI를 전송합니다.
- L. Client는 Access token을 보유한 상태이기 때문에 B~G 과정을 거치지 않습니다.
- 보유한 Access token의 유효성을 검증하기 위해 Access token을 Check token URI에 포함해 Authorizationserver에 전송합니다.
- M. Authorization server는 Authorities를 포함한 User information을 Resource server B에 전송합니다.
- N. Resource server B는 사용자의 접근 권한에 따른 페이지(user_page.jsp, admin_page.jsp, 404_page.jsp)를 Client에게 제공합니다.

사용자 정의 로그인 페이지

사용자 정의 로그인 페이지 관련된 내용을 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 환경 구성
- Nginx 설정

환경 구성

Nginx를 이용해서 사용자 정의 로그인 페이지를 화면에 표기하고 Cointack SignOn 구동을 원활하게 하기 위해서는 환경 구성은 완료해야 합니다. 본 장에서는 Nginx 설정 파일 구조와 Cointack SignOn 구동을 위한 환경을 구성하는 방법에 대해서 설명합니다.

Nginx

HTTP, Reverse proxy 등의 서버를 구동할 수 있는 오픈 소스 웹서버 프로그램으로, Cointack SignOn에서 리소스 서버와 SignOn 서버의 Reverse proxy 서버로 활용될 수 있습니다. 자세한 사항은 <https://nginx.org/en/>을 참조하시기 바랍니다.

Nginx 설치

설치는 Cointack SignOn이 설치된 OS에서 해야 하며, 이에 대한 명령어는 다음과 같습니다.

redhat 계열

redhat 계열은 Nginx에 대한 Repository를 제공하지 않기 때문에, 다음과 같이 저장소를 생성해야 합니다.

```
# vi /etc/yum.repos.d/nginx.repo
```

/etc/yum.repos.d/nginx.repo의 내용을 다음과 같이 작성합니다.

`${OS_VERSION}`에는 OS의 버전을 작성하면 됩니다. (CentOS 7일 경우, `${OS_VERSION}`은 7)

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/${OS_VERSION}/$basearch/
gpgcheck=0
enabled=1
```

다음의 명령어를 통해 Nginx를 설치합니다.

```
# yum update -y
# yum install -y nginx
```

debian 계열

debian 계열은 Nginx에 대한 Repository를 제공합니다.

따라서 별도로 저장소를 생성할 필요 없이, 다음의 명령어를 통해 Nginx를 설치합니다.

```
# apt-get update
# apt-get install -y nginx
```

Nginx 구동

Nginx 구동을 제어하는 명령어는 다음과 같이 2가지로 구분됩니다.

```
# systemctl start nginx
# systemctl stop nginx
# systemctl restart nginx
```

```
# service nginx start
# service nginx stop
# service nginx restart
```


Nginx 설정

Nginx 디렉터리 구조

redhat 계열

```
HTML 파일  
/usr/share/nginx/html

설정 파일  
/etc/nginx/conf.d
```

debian 계열

```
HTML 파일  
/var/www/html

설정 파일  
/etc/nginx/sites-enabled
```

로그인 페이지 작성

login.html

login.html 파일을 다음과 같이 작성합니다.

이후, redhat 계열일 경우, **/usr/share/nginx/html**,

debian 계열일 경우, **/var/www/html** 디렉터리 아래에 위치시킵니다.

```
<html>
  <head>
    <title>Custom Login Page</title>
  </head>
  <body>
    <h3>Welcome to custom login example</h3>
    <div>
      <form action="/login" method="post">
        <table>
          <tr><td>User:</td><td><input type="text" id="username" name="username"/></td></tr>
          <tr><td>Password:</td><td><input type="password" id="password" name="password"/></td></tr>
          <tr><td colspan='2'> <button type="submit" class="btn">Login</button></td></tr>
        </table>
      </form>
    </div>
  </body>
</html>
```

Nginx 설정

Nginx 기본 설명

사용자 정의 로그인 페이지 설정에 필요한 부분만 설명하였습니다.

```
root : html 파일을 참조할 경로 지정
  ex) root /usr/share/nginx/html; -> /usr/share/nginx/html 아래 있는 html 파일을 참조하게 해줍니다.

location : 사용자가 접근하려는 경로에 대한 제어를 가능하게 해줍니다.
  ex) location / {} -> root 경로로 접근할 경우 제어

proxy_pass : 내부적으로 처리할 URI로 연결해줍니다.
  ex) proxy_pass http://localhost:8080; -> 어떤 URI로 접근 하더라도 localhost:8080에서 처리가능하게 해줍니다.
```

```

root : html 파일을 참조할 경로 지정
    ex) root /usr/share/nginx/html; -> /usr/share/nginx/html 아래 있는 html 파일을 참조하게 해줍니다.

try_files : root 경로 아래서 참조할 파일을 찾아줍니다.
    ex) location /test {
        try_files $uri /login.html; -> /test로 접근할 경우 login.html 페이지를 보여줍니다.
    }
}

```

default customization

Nginx 설정은 다음과 같습니다.

redhat 계열은 `/etc/nginx/conf.d` 아래의 **default.conf** 파일을 수정해야 하며,

debian 계열은 `/etc/nginx/sites-enabled` 아래의 **default** 파일을 수정하면 됩니다.

#은 주석을 의미하므로 OS에 맞게 설정하면 됩니다.

```

server {
    listen 80 default_server;
    listen [::]:80 default_server;

    # redhat 계열
    # root /usr/share/nginx/html;

    # debian 계열
    # root /var/www/html;

    server_name _;

    location / {
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header Host $http_host;
        proxy_pass "http://127.0.0.1:8080";
    }

    location /login {
        if ( $request_method = POST ) {
            proxy_pass http://localhost:8080;
        }

        try_files $uri /login.html;
    }

    location ~* \.(css|jpg|jpeg|png|js)$ {
        # redhat 계열
        # root /usr/share/nginx/html/public;

        # debian 계열
        # root /var/www/html/public;
        break;
    }
}

```

Coinstack SignOn과 같은 서버에 구동하였고 **default 포트가 8080이기 때문에 proxy_pass**를 `http://localhost:8080**`으로 설정해 주었습니다.

Login 경로는 **2가지**의 처리를 담당합니다.

- Login 페이지를 표현 (Request method **GET** 방식)
- Login 페이지로부터 전달받는 데이터를 처리(Request method **POST** 방식)

두가지 처리를 구분하기 위해 if 문으로 request_method 값을 체크하였습니다.

※ Nginx 설정 변경 후 반드시 Nginx를 **재시작**해야 변경 결과가 적용되며, 관련 명령어는 [환경 구성](#)을 참조하시기 바랍니다.

Welcome to custom login example

User:

Password:

Appendix

Coinstack SignOn을 사용하는데 필요한 부가적인 정보를 설명합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- 오픈 소스 라이센스
- 설정
- 명령행 자동 완성
- 테스트
- 성능
- 알려진 이슈

오픈 소스 라이센스

Coinstack SignOn은 일부 기능에 있어서 오픈 소스 라이브러리를 사용하고 있습니다. 라이센스 파일은 \${INSTALL_PATH}/doc/license 폴더에 존재합니다.

라이브러리별 라이센스

오픈 소스 라이브러리의 목록과 그에 따른 라이센스는 다음과 같습니다.

라이브러리	라이센스
ch.qos.logback:logback-classic:1.2.3	Eclipse Publicv1, GNULesserGeneralPublic 2.1
ch.qos.logback:logback-core:1.2.3	Eclipse Publicv1, GNULesserGeneralPublic 2.1
com.beust:jcommander:1.72	Apache 2.0
com.fasterxml.jackson.core:jackson-annotations:2.9.0	Apache 2.0
com.fasterxml.jackson.core:jackson-core:2.9.4	Apache 2.0
com.fasterxml.jackson.core:jackson-databind:2.9.4	Apache 2.0
com.fasterxml.jackson.databind:jackson-datatype-jdk8:2.9.4	Apaache 2.0
com.fasterxml.jackson.databind:jackson-datatype-jsr310:2.9.4	Apache 2.0
com.fasterxml.jackson.module:jackson-module-parameter-names:2.9.4	Apache 2.0
com.fasterxml:classmate:1.3.1	Apache 2.0
com.github.rholder:guava-retrying:2.0.0	Apache 2.0
com.google.code.findbugs:jsr305:2.0.2	Apache 2.0
com.google.errorprone:error_prone_annotations:2.0.18	Apache 2.0
com.google.guava:guava:23.0	Apache 2.0
com.google.j2objc:j2objc-annotations:1.1	Apache 2.0
commons-codec:commons-codec:1.6	Apache 2.0
io.blocko:coinstack:3.5.1	MIT
io.micrometer:micrometer-core:1.0.1	Apache 2.0
javax.annotation:javax.annotation-api:1.3.2	CDDL
javax.validation:validation-api:2.0.1.Final	Apache 2.0
joda-time:joda-time:2.9.9	Apache 2.0
org.apache.commons:commons-pool2:2.5.0	Apache 2.0
org.apache.logging.log4j:log4j-api:2.10.0	Apache 2.0
org.apache.logging.log4j:log4j-to-slf4j:2.10.0	Apache 2.0
org.apache.tomcat.embed:tomcat-embed-core:8.5.28	Apache 2.0
org.apache.tomcat.embed:tomcat-embed-el:8.5.28	Apache 2.0
org.apache.tomcat.embed:tomcat-embed-websocket:8.5.28	Apache 2.0

org.apache.tomcat:tomcat-annotations-api:8.5.28	Apache 2.0
org.bouncycastle:bcpkix-jdk15on:1.56	BouncyCastle
org.bouncycastle:bcprov-jdk15on:1.58	BouncyCastle
org.codehaus.jackson:jackson-core-asl:1.9.13	Apache 2.0
org.codehaus.jackson:jackson-mapper-asl:1.9.13	Apache 2.0
org.codehaus.mojo:animal-sniffer-annotations:1.14	MIT
org.hdrhistogram:HdrHistogram:2.1.10	PublicDomain, perCreativeCommonsCC0, BSD-2-Clause
org.hibernate.validator:hibernate-validator:6.0.7.Final	Apache 2.0
org.jboss.logging:jboss-logging:3.3.0.Final	Apache 2.0
org.latencyutils:LatencyUtils:2.0.3	PublicDomain, perCreativeCommonsCC0
org.slf4j:jul-to-slf4j:1.7.25	MIT
org.slf4j:slf4j-api:1.7.25	MIT
org.springframework.boot:spring-boot-actuator-autoconfigure:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-actuator:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-autoconfigure:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-starter-actuator:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-starter-json:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-starter-logging:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-starter-security:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-starter-tomcat:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-starter-web:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot-starter:2.0.0.RELEASE	Apache 2.0
org.springframework.boot:spring-boot:2.0.0.RELEASE	Apache 2.0
org.springframework.security.oauth:spring-security-oauth2:2.0.8.RELEASE	Apache 2.0
org.springframework.security:spring-security-config:5.0.3.RELEASE	Apache 2.0
org.springframework.security:spring-security-core:5.0.3.RELEASE	Apache 2.0
org.springframework.security:spring-security-jwt:1.0.9.RELEASE	Apache 2.0
org.springframework.security:spring-security-web:5.0.3.RELEASE	Apache 2.0
org.springframework:spring-aop:5.0.4.RELEASE	Apache 2.0
org.springframework:spring-beans:5.0.4.RELEASE	Apache 2.0

org.springframework:spring-context:5.0.4.RELEASE	Apache 2.0
org.springframework:spring-core:5.0.4.RELEASE	Apache 2.0
org.springframework:spring-expression:5.0.4.RELEASE	Apache 2.0
org.springframework:spring-jcl:5.0.4.RELEASE	Apache 2.0
org.springframework:spring-web:5.0.4.RELEASE	Apache 2.0
org.springframework:spring-webmvc:5.0.4.RELEASE	Apache 2.0
org.yaml:snakeyaml:1.19	Apache 2.0

설정

다음은 Cointstack SignOn을 구동하기 위해 설정한 목록입니다.

설정 이름	의미	필수 유무	
coinstack.endpoint	CoinstackNode 의 Endpoint	필수	http://localhost:3000
jwt.privatekey	토큰 생성을 위한 개인키		
signon.admin.address	관리자의 어드레스	필수	1KkM2NLZEK9SL4sChJVgg45k3godj7Tnix
signon.server.port	서버의 포트		8080
signon.server.privatekeys	서버가 사용하는 개인키들	필수	L2CMbG8JqaAQpJDgrmoeVq7fcpKnycCd9rQg5vCNBTxZFX
signon.server.token.validity	토큰의 유효시간. 단위는 초		600
signon.server.user-repository	사용자 정보에 접근하는 구현체의 클래스명		coinstack.signon.repository.SimpleUserRepository
signon.peer.addresses	다른 서버의 어드레스		

명령행 자동 완성

명령행을 사용해서 쉽게 조작하기 위해서 자동 완성 스크립트를 제공합니다. 설치 시 `~/.bash_completion.d` 안에 자동완성 스크립트가 복사가 됩니다. 본 문서에서는 이를 삭제하였을시 다시 설정하는 방법을 다룹니다. 자동 완성 기능은 Bash 기반으로 작성되어 있습니다.

설치

자동 완성 스크립트를 설치할 디렉터리를 생성합니다.

```
$ mkdir -p ~/.bash_completion.d
```

자동 완성 스크립트를 복사합니다.

```
$ cp ${INSTALL_PATH}/doc/coinstack-signon.completion ~/.bash_completion.d
```

해당 파일을 읽어 들이는 스크립트를 `~/.bash_completion` 파일에 작성합니다.

```
$ cat <<"EOF" >> ~/.bash_completion
for bcfie in ~/.bash_completion.d/* ; do
    . $bcfile
done
EOF
```

재로그인을 하거나 다음 명령으로 적용할 수 있습니다.

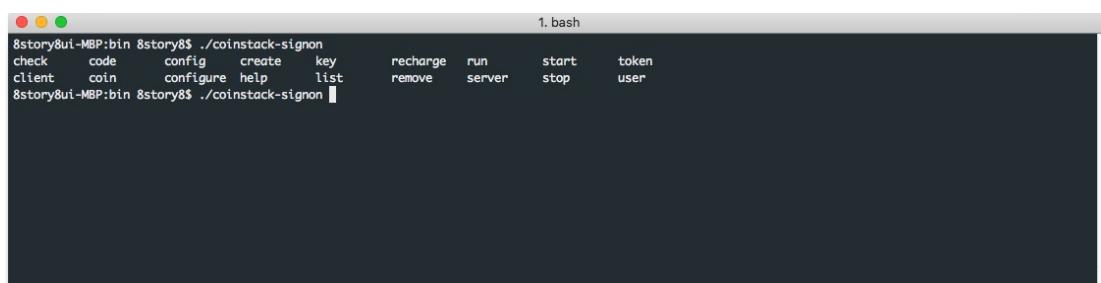
```
$ . ~/.bash_completion
```

자동 완성 기능 확인

자동 완성을 확인하기 위해 다음 명령을 실행합니다.

```
$ coinstack-signon [tab]
```

아래와 같이 수행할 수 있는 명령들을 보여줍니다.



테스트

사용자의 편의를 위해 간단한 테스트 스크립트를 제공합니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

- Postman 테스트
- 기능 테스트
- 부하 테스트

Postman 테스트

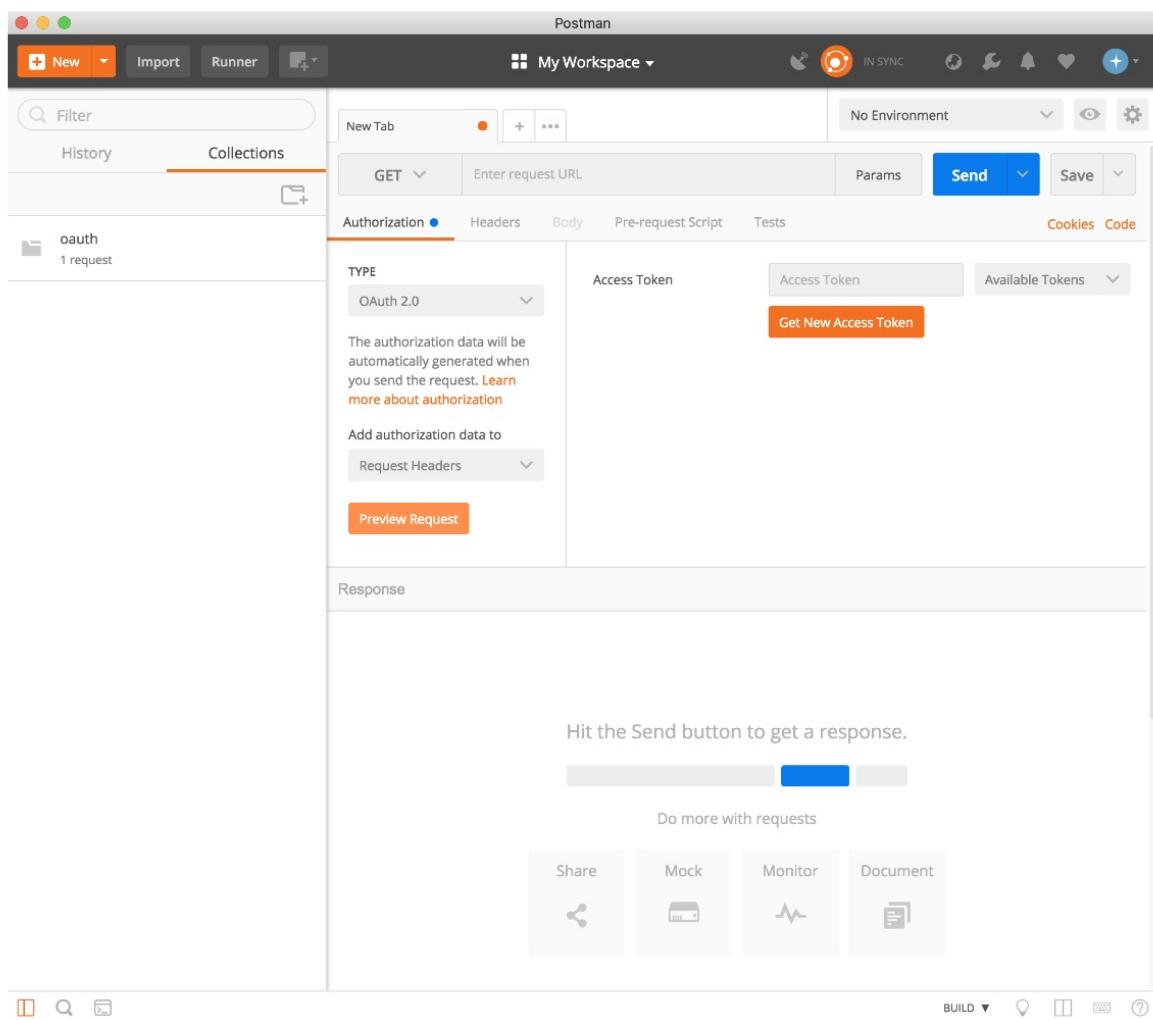
Coinstack SignOn의 OAuth 2.0 인증 서비스 흐름을 확인하는 Postman 테스트를 제공합니다.

테스트 예제의 Postman 버전은 6.0.10입니다.

OAuth 2.0 설정

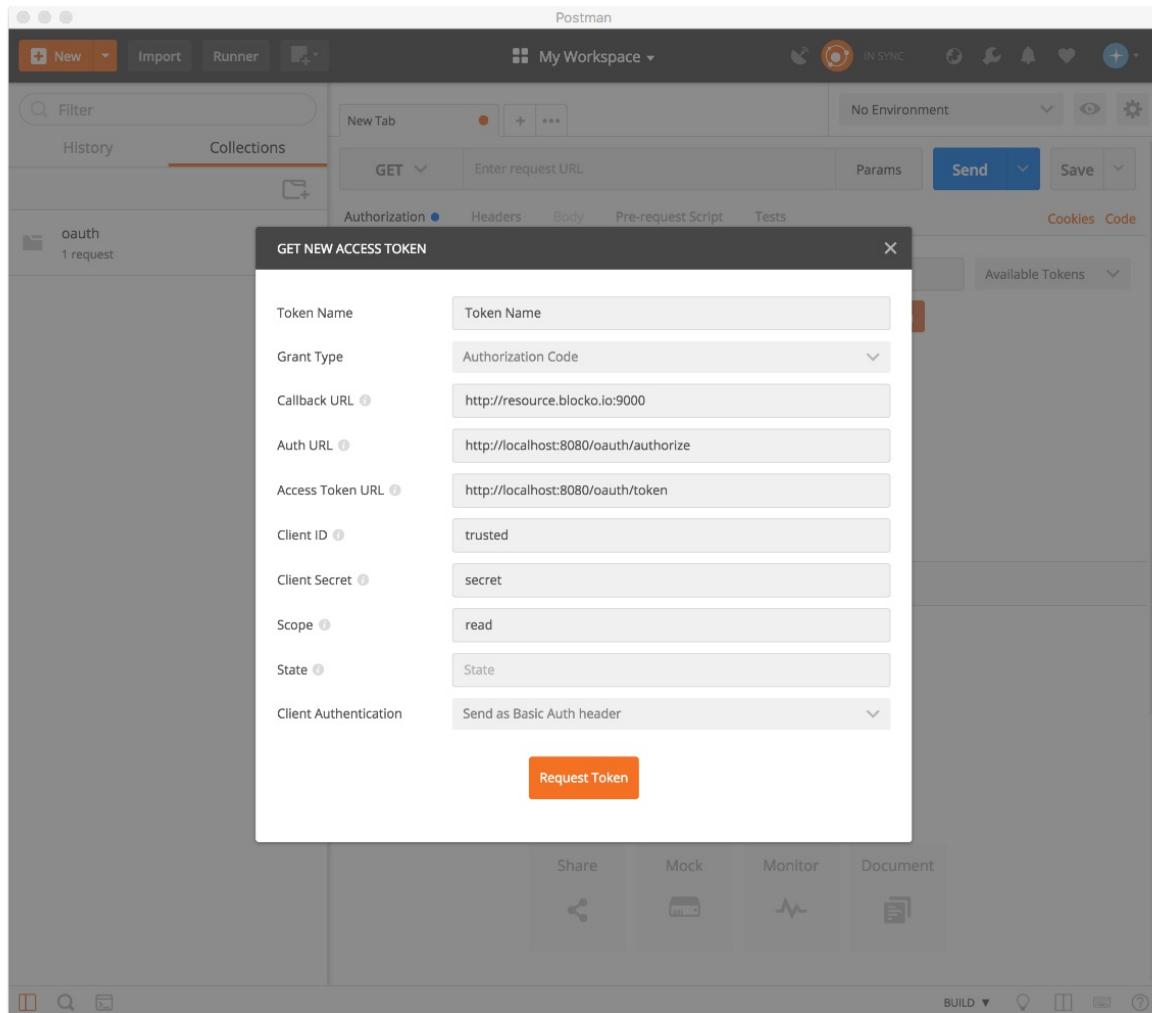
Postman의 Authorization TYPE을 OAuth 2.0으로 다음 화면처럼 설정한 뒤,

액세스 토큰을 발급받기 위해 Get New Access Token 버튼을 클릭합니다.

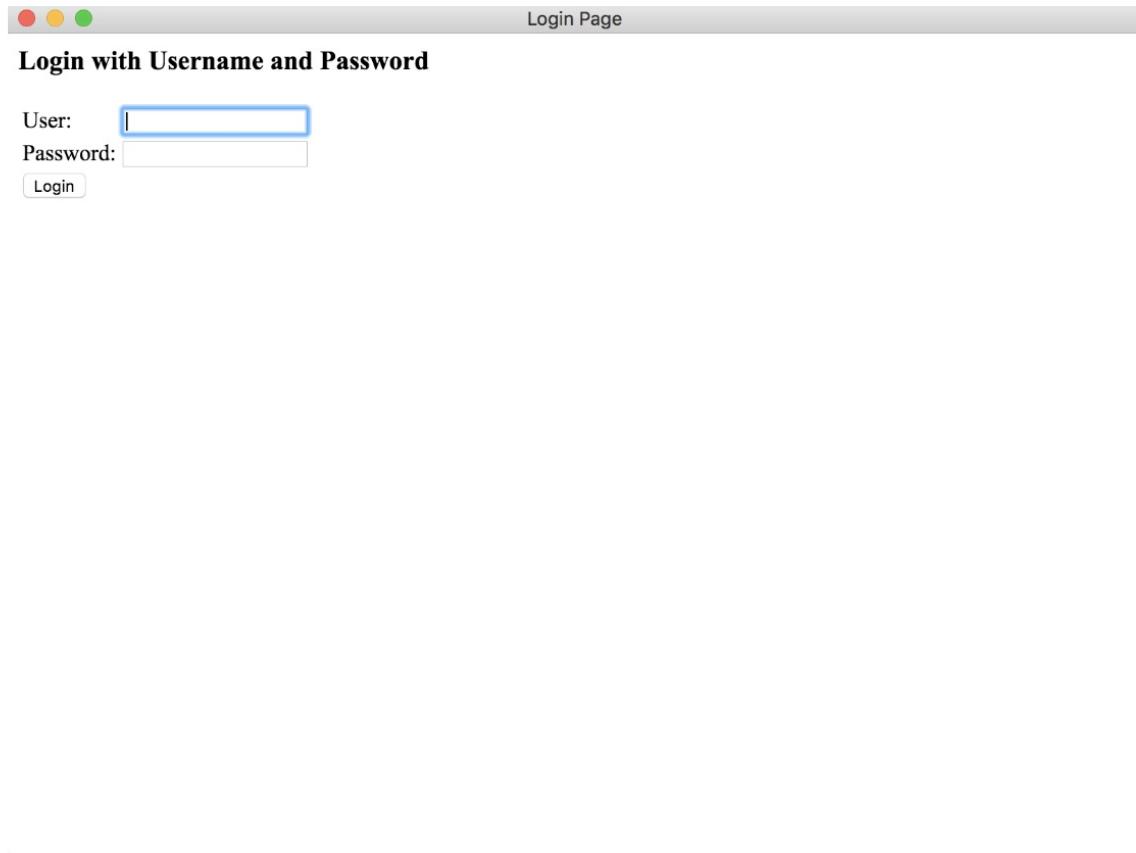


액세스 토큰 발급

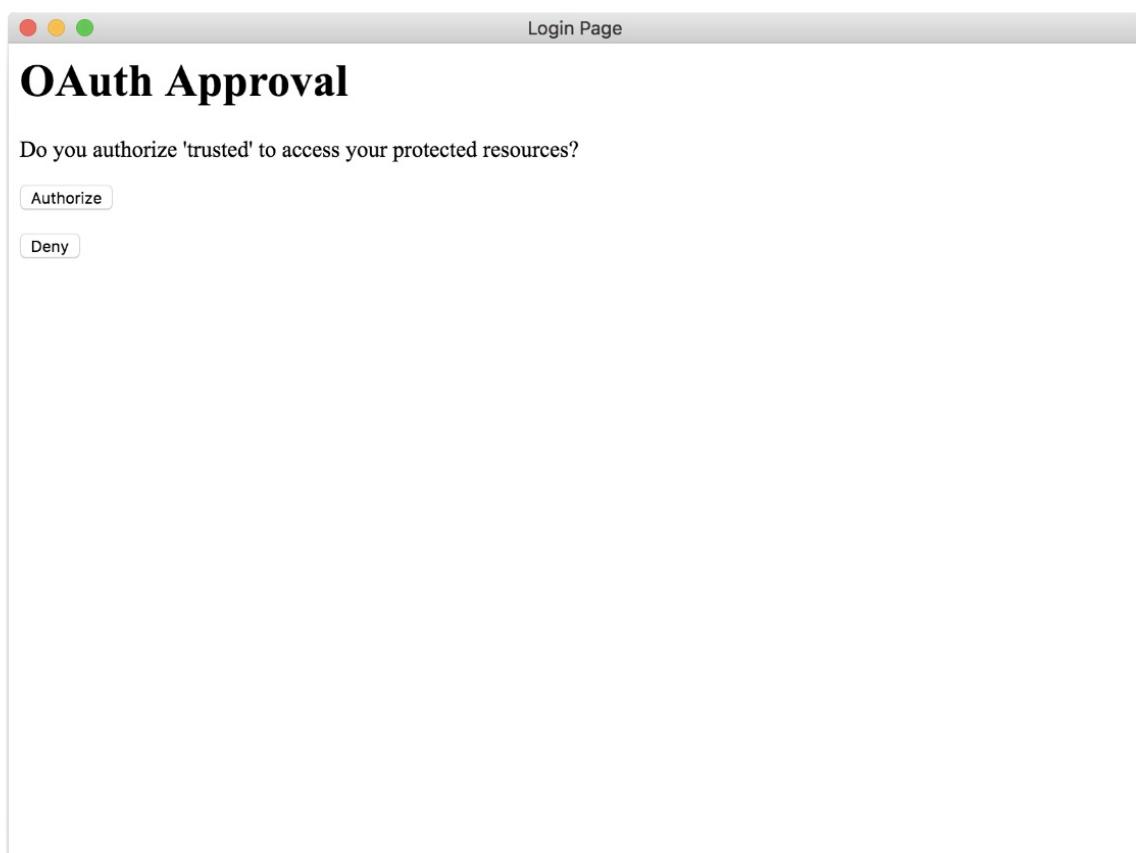
액세스 토큰을 발급받기 위해 등록된 클라이언트 정보에 따라 양식을 다음 화면처럼 설정한 뒤, Request Token 버튼을 클릭합니다



Request Token 버튼을 통해 Token을 요청하면 로그인 페이지로 리다이렉트됩니다.



이후 등록된 사용자의 ID/Password로 로그인, 사용자 허가의 과정을 수행합니다.

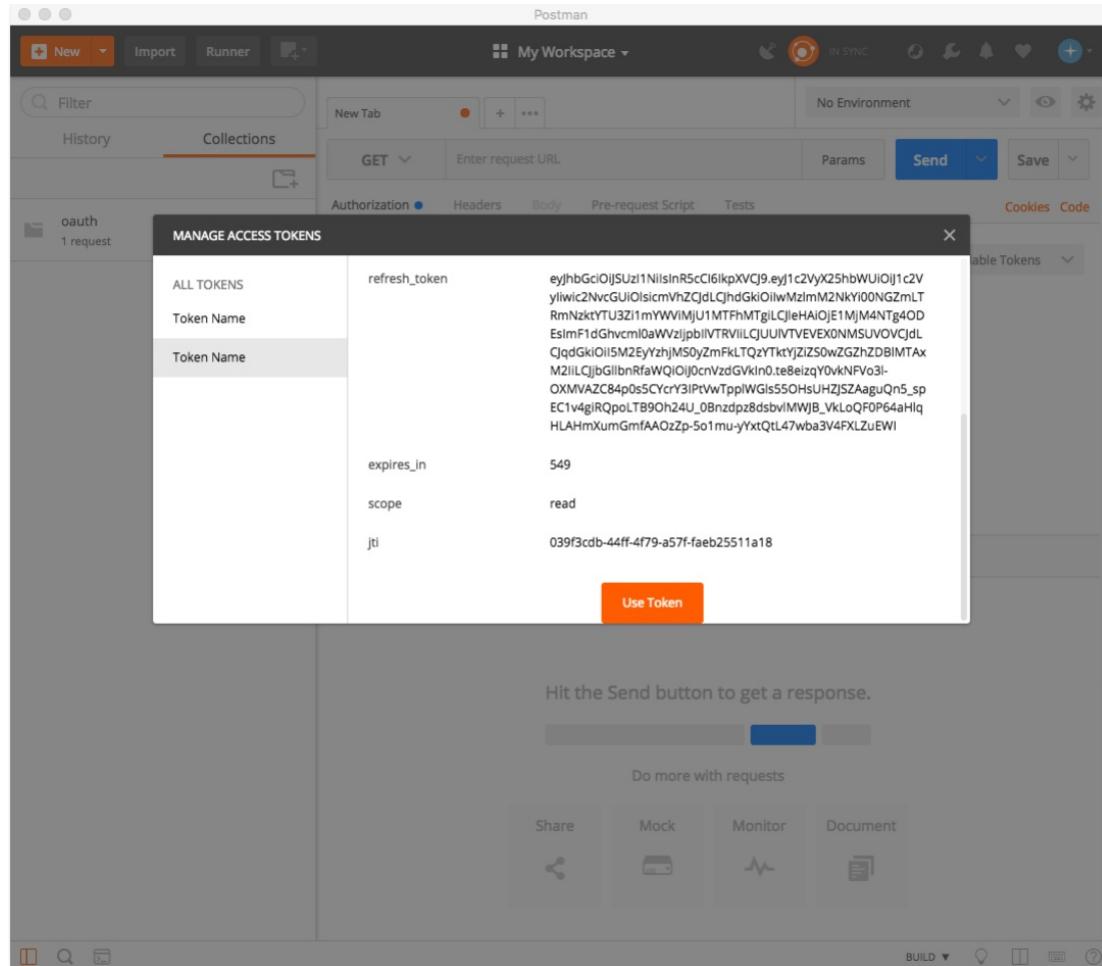


이후 다음 화면처럼 액세스 토큰에 대한 정보를 확인할 수 있습니다.

The screenshot shows the Postman application window. At the top, there are tabs for 'New', 'Import', 'Runner', and 'My Workspace'. Below the tabs, there's a search bar labeled 'Filter' and a 'Collections' section. A modal window titled 'MANAGE ACCESS TOKENS' is open in the center. The modal has two columns: 'Token Name' and 'Token Value'. It lists two tokens: 'Access Token' and 'refresh_token'. The 'Access Token' value is a long, complex string of characters, and its 'Token Type' is listed as 'bearer'. The 'refresh_token' value is also a long string, and its 'Token Type' is listed as 'refresh_token'. At the bottom of the modal, there's a message 'Hit the Send button to get a response.' and a large blue 'Send' button. Below the modal, there are buttons for 'Share', 'Mock', 'Monitor', and 'Document'. The bottom right corner of the screen shows the Postman interface's status bar.

Token Name	Token Value
Access Token	eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOiE1MjM4NTg4ODEsInVzZXJfbmFtZSIsInVzZXIiLCJhdXRob3p0dGlcyI6WyVU0ViSlwlfFVU1FRFR9DTEIFTIQiXSwianRpIjoiMDM5ZjNjZGltNDRmZl00ZjcsLWE1N2Y1ZmFyJi1NTExYTE4liwiY2xpZW50X2lkIjoidH1c3RIZCIsInNjb3BlIjpbInIjWQoXX0.ebyVNHYEjWYYTjuL36DP3_TS5cdEtHhf0PRmNDYfDXsd0IKR77_sX33f53BTk9LSiE34P-SpLkWZ2U6j_IN_FM1_cm91-zAWBDAhmb6AcM4-1Kj3cnNj9GL0DCht-GuAJKUgj24Ub7tp5sz2pjX_fpPuYly1y2TkNOB
refresh_token	eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9eyJ1c2VyX25hbWUiOiJ1c2Vylivic2NvcGUoIscmVhZCjdLCJhdGkiOiIwMzlmM2Nkyj00NGZmLTNmNzktTU3Zi1mYWViMjU1MTFhMTgjLCJleHAiOiE1MjM4NTg4ODEsImF1dGhvcml0aWZjpbIlVTRVilLQUlV7VEVEX0NMSUVOCjdlCiogdGkiOISM2EYzgbIMSOiZm5klTOxTkxViZlZSDwZGzbZDBIMTAy

액세스 토큰의 유효성 검사를 위해 Use Token 버튼을 클릭합니다.



액세스 토큰 유효성 검사

check_token.json

액세스 토큰 유효성 검사를 손쉽게 하기 위해 다음 파일을 Import하시기 바랍니다.

```
{
  "id": "b4b1a30f-e756-949c-b665-13df6af2276c",
  "name": "oauth",
  "description": "",
  "order": [
    "ff1fda8d-7c1d-e954-bb35-309b87795198"
  ],
  "folders_order": [],
  "folders": [],
  "owner": "3456127",
  "hasRequests": true,
  "requests": [
    {
      "id": "ff1fda8d-7c1d-e954-bb35-309b87795198",
      "headers": "Content-Type: application/x-www-form-urlencoded\nAuthorization: Basic dHJ1c3RlZDpzZWNyZXQ=\n",
      "headerData": [
        {
          "key": "Content-Type",
          "value": "application/x-www-form-urlencoded",
          "description": "",
          "enabled": true
        },
        {
          "key": "Authorization",
          "value": "Basic dHJ1c3RlZDpzZWNyZXQ=",
          "description": "",
          "enabled": true
        }
      ]
    }
  ]
}
```

```

],
"url": "localhost:8080/oauth/check_token",
"queryParams": [],
"preRequestScript": null,
"pathVariables": {},
"pathVariableData": [],
"method": "POST",
"data": [
{
  "key": "token",
  "value": "",
  "description": "",
  "type": "text",
  "enabled": true
}
],
"dataMode": "urlencoded",
"tests": null,
"currentHelper": "normal",
"helperAttributes": {},
"time": 1524212770984,
"name": "check_token",
"description": "",
"collectionId": "b4b1a30f-e756-949c-b665-13df6af2276c",
"responses": []
}
]
}

```

발급받은 액세스 토큰의 유효성을 검사하기 위해서 다음 화면처럼 check_token URL을 입력합니다.

그리고 Headers 탭에서 Content-Type에 application/x-www-form-urlencoded로 데이터 포맷을 정합니다.

이후, Authorization에 \${CLIENT_ID}:\${CLIENT_SECRET}를 **Base64로 Encoding**한 뒤,

Authorization의 Value에 **Basic \${ENCODED_TEXT}**를 다음 화면처럼 입력합니다.

Base64 Encoding/Decoding 방법

Encoding

```
echo -n '${CLIENT_ID}:${CLIENT_SECRET}' | base64
```

Decoding

```
echo -n '${ENCODED_TEXT}' | base64 -d | xargs
```

The screenshot shows the Postman application interface. In the top navigation bar, there are buttons for 'New', 'Import', 'Runner', and a workspace dropdown set to 'My Workspace'. The status bar indicates 'IN SYNC' with a sync icon. Below the navigation, there's a search bar labeled 'Filter' and tabs for 'History' and 'Collections'. A collection named 'oauth' is selected, showing one request. The main workspace displays a POST request to 'localhost:8080/oauth/check_token'. The 'Headers' tab is active, showing two entries: 'Content-Type' with value 'application/x-www-form-urlencoded' and 'Authorization' with value 'Basic dHJ1c3RlDpZWNyZXQ='. Below the headers, there's a 'Body' tab, a 'Pre-request Script' section, and a 'Tests' section. On the right side, there are buttons for 'Send', 'Save', 'Cookies', and 'Code'. A large central area says 'Hit the Send button to get a response.' Below it is a blue 'Send' button. At the bottom of the interface, there are buttons for 'Share', 'Mock', 'Monitor', and 'Document', along with other standard window controls.

Body 탭에서 x-www-form-urlencoded를 데이터 타입으로 지정한 후, 다음과 같이 입력합니다.

token의 value에 입력한 값은 발급받은 액세스 토큰 값입니다. Send 버튼을 클릭합니다.

The screenshot shows the Postman application interface. At the top, there are tabs for 'New', 'Import', 'Runner', and a workspace dropdown set to 'My Workspace'. Below the header, there's a search bar labeled 'Filter' and a 'Collections' section with a 'History' tab and a 'check_token' collection highlighted. The 'check_token' collection has one request under the 'oauth' folder.

The main area displays a POST request to 'localhost:8080/oauth/check_token'. The 'Body' tab is selected, showing the 'x-www-form-urlencoded' type with a single key-value pair: 'token' with the value 'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyleHAIQJE1MjM4NTg4ODEsInVzZXJfbmFtZSI6InVzZXIiLCJhdRob3pjdGlycy6WjyVU0VSliwiVFJVU1RFRF9DTEFTIQjXSwianRpjoIMDM52jNjZGtNDRmZl00Zjc5LWE1N2YzNmFjYj1NTEExTE4liwYxpZW50X2lkjoidHj1c3RIZCisInNjb3BlIjpbinJlYWQiXX0.e8yVNHYEjWYYTjuqL36DP3.T5cddEtUhhf0PRmnNDYfdXSxdOIKR77_-sX3f53BTk9LBIE34P-SpLkWZ2U6j_IN_FMI_cm91-zAWBDAhmb6AcM4-1Kj3cnN9LGL0DCtht-GuAkUGl24Ub7tp5s2pjX_fpPuYly1y2TkNO8'.

Below the request details, there are buttons for 'Share', 'Mock', 'Monitor', and 'Document'. At the bottom of the window, there are standard OS-level controls (minimize, maximize, close) and a 'BUILD' dropdown.

이후 다음과 같이 액세스 토큰의 유효성을 확인할 수 있습니다.

The screenshot shows the Postman application interface. In the top navigation bar, there are buttons for 'New', 'Import', 'Runner', and a workspace dropdown set to 'My Workspace'. The status bar indicates 'IN SYNC' with a progress bar.

The main workspace displays a collection named 'oauth' containing one request. A specific request titled 'check_token' is selected, showing a POST method to 'localhost:8080/oauth/check_token'. The 'Body' tab is active, showing a form-data key 'token' with the value 'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...'. Below the body, the response status is '200 OK', time is '42 ms', and size is '548 B'. The response body is displayed in a JSONpretty-printed format:

```
1 [ {  
2   "exp": 1523859956,  
3   "user_name": "user",  
4   "authorities": [  
5     "USER",  
6     "TRUSTED_CLIENT"  
7   ],  
8   "jti": "bbf8b685-3c36-4819-9cbc-9e216dfda90f",  
9   "client_id": "trusted",  
10  "scope": [  
11    "read"  
12  ]  
13 } ]
```

기능 테스트

OAuth 2.0 RFC 스펙을 만족하는지 확인하는 테스트 스크립트를 제공합니다. 테스트 스크립트는 Bash 기반으로 작성되어 있습니다. OAuth 2.0 RFC 스펙은 다음의 URL에서 확인하실 수 있습니다.

<https://tools.ietf.org/html/rfc6749>

실행

다음과 같이 실행할 수 있습니다.

```
$ coinstack-function-test
```

실행 시 다음과 같은 화면을 확인하실 수 있습니다. 블록체인에 클라이언트, 사용자가 등록되어 있지 않다면 **최초 한번**은 y를 눌러서 사용자, 클라이언트를 등록해야 합니다.

```
Do you wanna setup the client, user for function test [y|n]?
>
```

y를 누르면 관리자의 개인키를 입력하는 창이 출력됩니다.

```
Input the administrator's private key [default : KyQxDqRvUYAKnMHN3PvHwQUwpQvtu5Bz2MDARaNBZiTsdTRAaj5e]
>
```

관리자의 개인키를 입력하고 엔터를 치면 기능 테스트를 위한 클라이언트, 사용자를 생성 후 기능 테스트를 실행합니다. 그냥 엔터를 칠 경우 디폴트 값으로 설정 후 진행합니다.

```
Do you wanna setup the client, user for function test [y|n]?
> y
Input the administrator's private key [default : KyQxDqRvUYAKnMHN3PvHwQUwpQvtu5Bz2MDARaNBZiTsdTRAaj5e]
>
Setup processing..

scripts/2/CH2_1TC1.sh.....SUCCESS
scripts/2/CH2_3TC1.sh.....SUCCESS
scripts/2/CH2_3_1TC1.sh.....SUCCESS

...
scripts/6/CH6_3TC2.sh.....SUCCESS
scripts/6/CH6_3TC3.sh.....SUCCESS
scripts/6/CH6_4TC1.sh.....SUCCESS
success/all : 58 / 58
```

n을 누르면 기능 테스트를 바로 수행합니다.

```
Do you wanna setup the client, user for function test [y|n]?
> n
scripts/2/CH2_1TC1.sh.....SUCCESS
scripts/2/CH2_3TC1.sh.....SUCCESS
scripts/2/CH2_3_1TC1.sh.....SUCCESS

...
scripts/6/CH6_3TC2.sh.....SUCCESS
scripts/6/CH6_3TC3.sh.....SUCCESS
scripts/6/CH6_4TC1.sh.....SUCCESS
success/all : 58 / 58
```

만약 클라이언트, 사용자를 등록하지 않고 진행 시에 다음과 같이 29개만 성공합니다.

```
Do you wanna setup the client, user for function test [y|n]?
> n
```

```
scripts/2/CH2_1TC1.sh.....FAILURE
scripts/2/CH2_3TC1.sh.....FAILURE
scripts/2/CH2_3_1TC1.sh.....FAILURE

...
scripts/6/CH6_3TC2.sh.....FAILURE
scripts/6/CH6_3TC3.sh.....FAILURE
scripts/6/CH6_4TC1.sh.....FAILURE
success/all : 29 / 58
```

부하 테스트

서버가 감당할 수 있는 부하를 확인하는 부하 테스트 스크립트가 제공됩니다. 테스트 스크립트는 Bash 기반으로 JMeter를 사용하여 작성되어 있습니다.

JMeter 설치

본 스크립트는 JMeter 4.0 기반으로 작성되어 있습니다. 다음의 명령을 통해 JMeter 4.0 바이너리를 받아올 수 있습니다.

```
$ curl -o apache-jmeter-4.0.zip http://mirror.apache-kr.org//jmeter/binaries/apache-jmeter-4.0.zip
```

혹은 http://jmeter.apache.org/download_jmeter.cgi에서 받으실 수 있습니다.

정상적으로 다운로드가 완료되면 다음의 명령어를 통해 압축을 풀 수 있습니다.

```
$ unzip apache-jmeter-4.0.zip
```

정상적으로 압축을 해제하면 다음의 경로에 실행 파일이 있습니다. 다음의 명령어를 통해 환경변수에 JMeter를 추가합니다. \${INSTALL DIRECTORY}는 JMeter를 압축 해제한 디렉터리의 절대 경로입니다.

```
$ cat << EOF > ~/.bashrc
export PATH=\$PATH:${INSTALL DIRECTORY}/apache-jmeter-4.0/bin
EOF
```

재로그인을 하거나 다음 명령으로 적용할 수 있습니다.

```
$ . ~/.bashrc
```

실행

다음의 명령어를 통해 실행할 수 있습니다. \${N_THREAD}는 스레드의 개수입니다.

```
$ coinstack-load-test -n ${N_THREAD}
```

구체적인 인자는 -h 명령어를 통해 확인하실 수 있습니다. 괄호 안의 값은 기본값입니다.

```
$ coinstack-load-test -h
-i : server ip address          (SERVER_IP : localhost)
-p : server port                 (SERVER_PORT : 8080)
-n : number of threads to use   (N_THREAD : 1)
-r : rampup time in seconds     (RAMPUP_TIME : 30)
-d : test duration in seconds    (DURATION : 300)
-t : timeout per each request   (TIMEOUT : 0 (0 means no timeout))
-h : show options
```

인자	설명
-i	서버 아이피 (기본값 : localhost)
-p	서버 포트 (기본값 : 8080)
-n	스레드 개수 (기본값 : 1)
-r	램프업 시간 (기본값 : 30초)
-d	테스트 시간 (기본값 : 5분)
-t	각 요청 별 타임아웃 (기본값 : 없음)

-h

도움말 출력

실행 시 다음과 같은 화면을 보실 수 있습니다. 블록체인에 클라이언트, 사용자가 등록되어 있지 않다면 **최초 한번**은 y를 눌러서 클라이언트, 사용자를 등록해야 합니다.

```
Do you wanna setup the client, user for load test [y|n]?
>
```

y를 누르면 관리자의 개인키를 입력하는 창이 출력됩니다.

```
Input the administrator's private key [default : KyQxDqRvUYAKnMHN3PvHwQUwpQvtu5Bz2MDARaNBZiTsdTRAaj5e]
>
```

관리자의 개인키를 입력하고 엔터를 치면 부하 테스트를 위해 필요한 사용자, 클라이언트를 생성 후 부하 테스트를 실행합니다. 그냥 엔터를 칠 경우 디폴트 값으로 설정 후 진행합니다. **사용자의 개수는 테스트 수행 시 인자로 들어온 \${N_THREAD} 만큼 생성되므로 이후 \${N_THREAD}보다 큰 스레드 개수로 테스트를 수행하려고 할 시 다시 실행하여야 합니다.** 그렇지 않으면 최초 한번 실행 후 실행하지 않아도 됩니다.

```
Do you wanna setup the client, user for function test [y|n]?
> y
Input the administrator's private key [default : KyQxDqRvUYAKnMHN3PvHwQUwpQvtu5Bz2MDARaNBZiTsdTRAaj5e]
>
Processing clients..
<===== 100 %
...
...
```

n을 누르면 부하 테스트를 바로 수행합니다.

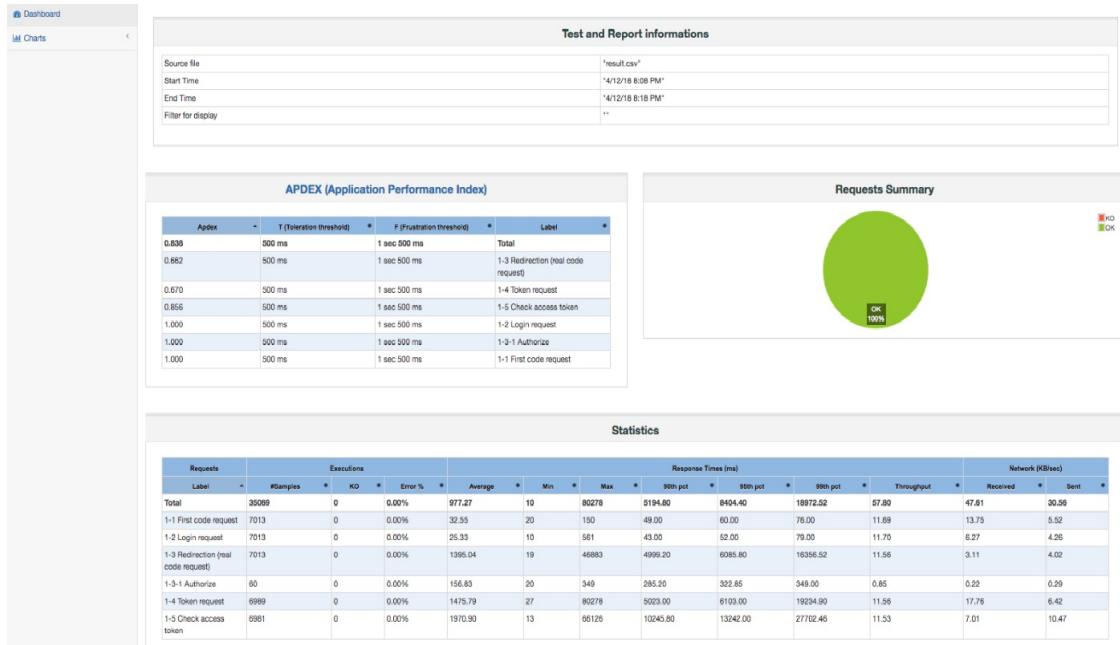
```
Do you wanna setup the client, user for function test [y|n]?
> n
Settings:
  host                  http://localhost:8080
  number of thread      1
  rampup time           60 seconds
  test duration          600 seconds
  timeout per each request  0 milliseconds
  result directory name postfix  dashboard
...
...
```

테스트 결과는 load-test-n\${N_THREAD}-dashboard 디렉터리 안의 index.html을 브라우저로 열면 확인하실 수 있습니다.

```
$ ls load-test-n10-dashboard/
README.TXT    content      index.html      jmeter.log      result.csv      sbadmin2-1.0.7
```

같은 스레드 개수로 부하 테스트를 재실행시 중복제거를 위해 기존 디렉터리는 사라지니 다른 곳으로 옮기고 실행해야 합니다.

index.html을 브라우저로 열면 다음과 같은 화면을 확인할 수 있습니다.



성능 측정의 대표적인 지표인 TPS는 Response Times의 Throughput으로 확인하실 수 있습니다.

자세한것은 다음의 URL에서 확인할 수 있습니다.

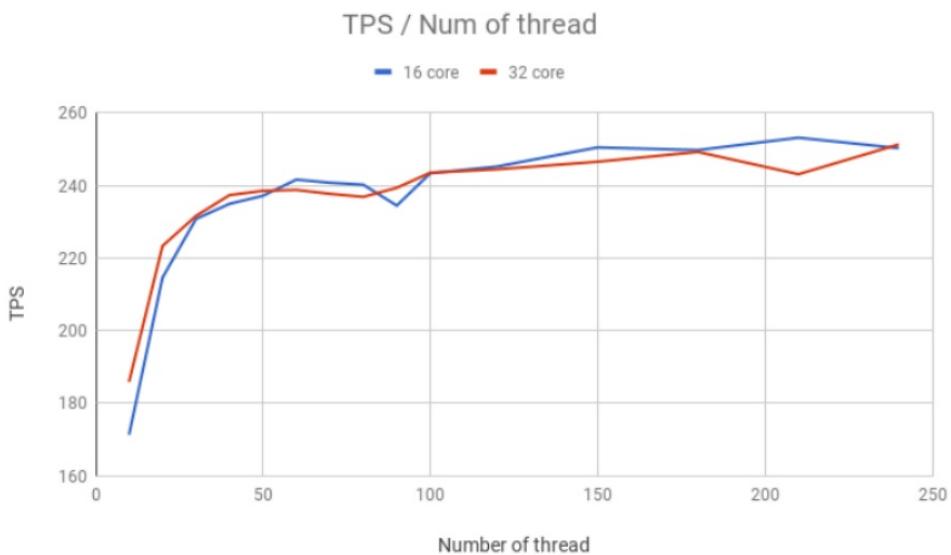
<https://jmeter.apache.org/usermanual/generating-dashboard.html>

성능

Azure 기반 가상머신에서 진행된 부하 테스트의 결과를 보여줍니다. 테스트는 서버와 노드를 같은 PC에 설치한 후 사용자의 수를 바꿔가며 5분간 진행한 후 5분간의 평균 TPS(Transfer Per Second)를 측정하였습니다. 편차를 줄이기 위해 같은 시나리오로 5번 측정한 후 결과를 평균 내었습니다. 테스트는 2.4GHz Intel Xeon® E5-2673v3(Haswell) 프로세서 또는 최신 2.3GHz Intel XEON® E5-2673 v4(Broadwell) 프로세서를 기반으로 하는 Dsv3 시리즈에서 Standard_D16s_v3(16코어/64기가 메모리), Standard_D32s_v3(32코어/128기가) 시리즈의 가상머신을 구축한 후 수행하였습니다. 가상머신에 대한 자세한 정보는 다음의 URL에서 확인할 수 있습니다.

<https://docs.microsoft.com/ko-kr/azure/virtual-machines/windows/sizes-general>

결과



결과 그래프를 보면 스레드의 개수가 증가함에 따라 30개 정도까지는 TPS(Transfer Per Second)가 선형적으로 증가하다가 그 이후로는 완만하게 증가하는 것을 볼 수 있습니다. 이후 스레드의 개수가 증가함에 따라 TPS가 250정도에 수렴하는 것을 확인할 수 있습니다. 자세한 측정 결과는 다음의 도표에서 확인할 수 있습니다.

n_thread	16 core						32 core					
	1	2	3	4	5	average	1	2	3	4	5	average
10	160.73	172.44	171.95	181.03	170.17	171.264	182.05	185.62	188.77	186.32	186.5	185.852
20	202.63	218.95	216.46	220.25	214.63	214.584	209.84	228.32	231.43	222.62	224.42	223.326
30	216.69	232.43	241	233.55	230.2	230.774	220.3	235.52	237.41	233.99	230.8	231.604
40	218.75	240.22	242.82	238.43	234.45	234.934	224.12	245.32	243.17	237.44	236.64	237.338
50	221.01	243.55	240.8	242.31	238.07	237.148	215.65	247	243.12	245.16	241.69	238.524
60	227.26	248.79	245.64	244.75	241.66	241.62	219.61	246.31	244.92	248.77	234.21	238.764
70	230.46	245.24	247.89	241.27	238.88	240.748	220.69	246.47	247.54	244.34	229.41	237.69
80	229.31	246.64	242.3	240.14	242.66	240.21	220.2	246.93	239.5	246.36	231.29	236.856
90	229.28	235.63	233.58	223.06	250.69	234.448	232.41	246.74	240.11	241.36	236.15	239.354
100	253.73	241.91	248.54	228.28	244.32	243.356	251.79	242.33	243.16	247.37	232.95	243.52
120	243.06	247.04	242.58	248.84	244.62	245.228	249.19	245.26	240.02	254.09	233.75	244.462
150	260.5	247.37	247.6	254.74	242.24	250.49	255.8	240.78	247.09	256	233.05	246.544
180	256.89	248	247.9	257.04	238.96	249.758	258.8	236.56	254.14	254.97	241.76	249.246
210	262.36	250.34	256.86	258.7	237.6	253.172	255.52	229.21	242.16	251.67	236.96	243.104
240	256.93	258.35	259.92	248.72	227.51	250.286	263.31	234.32	262.34	261.39	235.26	251.324

알려진 이슈

Coinstack SignOn에서 발생할 수 있는 이슈들에 관한 내용을 담고 있습니다.

본 장에서는 다음과 같은 내용을 기술하고 있습니다.

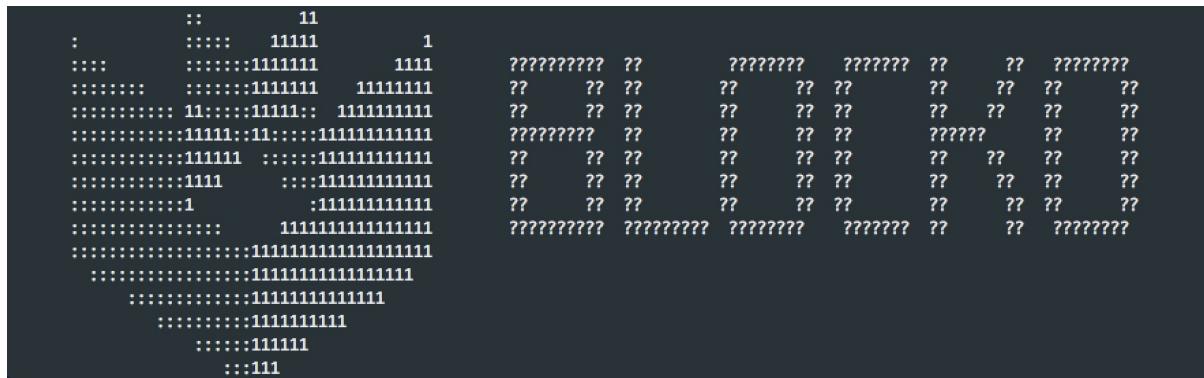
- 문자셋 미지정

문자셋 미지정

OS는 각각 사용자가 어떤 언어를 사용하는지에 대한 문자 집합인 Locale을 가지고 있습니다.

만일 Locale이 지정되지 않으면, 한글이나 특수 문자를 읽고 쓰는데 문제가 있을 수 있습니다.

만일 Locale에 문제가 있다면, SignOn 서버 시작 시 출력되는 배너가 깨져서 보이는 현상이 발생합니다.



다음 명령어를 통해 Locale을 추가하여 문제를 해결합니다. (CentOS의 경우)

```
# localedef -f UTF-8 -i ko_KR ko_KR.UTF-8
```

문자셋이 올바르다면 다음과 같은 배너를 볼 수 있습니다.

