1. 비밀번호를 해싱하는 것은 반드시 필요하다. 왜? 해커가 DB에 접근했을 때, 가장 최악의 상황은 원본의 데이터들을 가져가는 것이다.

* 해싱, 암호화하는 방법에 관해 공부해보자.

해싱: 단방향. 복호화 하기 힘들다. 보통 DB에 passwd를 저장할 때 해싱한 결과 값을 저장한다고 한다. 원본에 수학적 알고리즘을 적용시켜 재매핑한 결과물을 만들어 내는 것으로, 이렇게 변환된 데이터는 Digest라고 한다.

대표적인 알고리즘으로는 다음과 같다.

SHA, MD, HAS, WHIRLPOOL

하지만 레인보우 테이블로 인해 완벽하게 안전한 방법은 아니다. (동일한 데이터 값은 동일한 Digest 결과물을 반환하기 때문에 쉽게 유추가 가능하다는 것)

이를 해결하기 위해 다음과 같은 방법이 존재

Key-Stretching: 해싱한 결과물을 반복해서 해싱을 시도하는 것.

Salt: 해싱을 돌리기 이전에 임의의 문자열을 추가한 후에 해싱하는 것. 너무 짧으면 의미가 없고 32비트 이상은 되야한다고 한다.

암호화: 양방향. 다시 복호화가 가능하다.

C#에서는 System.Security.Cryptography 에서 MD5 해시 알고리즘을 지원함으로 이를 사용해보기로 했다.

1. 지난 시간 미팅때, Scoped, Trasient, SingleTon 등 여러 가지 방법으로 컨트롤러 의존성을 주입하는 방법에 관해 설명 받았는데, Scoped는 거의 사용하지 않는다고 한다. 그리고 계정 생성에 있어 DB와 직접적인 연결작업을 수행하는 AccountController의 경우 Trasient를 통해 등록하였는데 이에 관련된 이유를 MySQLConnection을 통해 확인해볼 필요가 있다.
2. MsSql MySql 중 왜 하필 MySql을 채택했을까? 에 관한 고민이 필요하다.
3. 내가 처음부터 직접 작업하는 것이 중요. 코드에 관해 모든 것을 내 것으로 변경해서 생각하라. Program.cs에서 httpredirection.. 이런 친구들이 왜 필요한지, 과연 필요한 것인지 잘 생각해서 사용해라.
4. 의존성 주입이라는 빌드 패턴 관련해서 공부할 필요성이 있다. (사전 개념 Interface)

* 통상적인 설명

DI: 의존성 주입 디자인 패턴

의존한다 라는 것의 정의부터 간단하게 살펴보자

B는 A에 의존한다. 라는 관계가 설정되었을 때, 이는 A가 변한다면 B도 변한다. 라는 것을 의미한다. C#에서는 이를 Interface를 활용해서 표현하기도 한다.

이런 내부적인 의존성을 외부에서 런타임 시, 결정하도록 하기 위해 생긴 방법이

DI(의존성 주입)이다.

예를 들어, 물건을 판매하기 위한 Store 클래스가 있다고 가정. 해당 Store에는 펜을 판매한다.

텍스트, 스크린샷, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

그렇다면 이 Store 클래스는 생성자 호출 시, \_pen = new Pen() 형태로 이루어질 것.

즉 Store는 Pen이라는 클래스에 강하게 의존하고 있다. 이런 경우 발생할 수 있는 문제점이 있다.

1. 두 클래스 간의 관계가 강하게 결합되어 있다. 만약 Store에서 다른 물건을 팔려고 시도한다면? 해당 Store 클래스의 전반적인 내용이 변경하거나 불필요한 상속 선언을 수행해야 한다.
2. 객체와 객체 간의 관계가 아닌 클래스와 클래스 간의 관계가 맺어짐으로써 객체 지향적이지 못하다.

해당 문제를 해결하기 위해 Pen, 혹은 다른 상품들을 추상적으로 묶어 처리할 수 있는 인터페이스가 필요하다. Product (상품이라는 인터페이스)를 통해 이를 방지할 수 있다.

의존성 주입에도 여러가지 방법이 존재한다.

1. 생성자 주입
2. 수정자 주입
3. 필드 주입

각 주입법을 통해 다양하게 생성이 가능한데, asp .net core에서는 보통은 생성자 주입법을 보편적으로 사용하는 것 같다.

1. 계정 로그인 후, 임의의 토큰을 부여해 유저를 구분하는 방식으로 구현해보자