パスワード（およびハッシュ化）によるOAuth2、JWTトークンによるBearer

これでセキュリティの流れが全てわかったので、JWTトークンと安全なパスワードのハッシュ化を使用して、実際にアプリケーションを安全にしてみましょう。

このコードは、アプリケーションで実際に使用したり、パスワードハッシュをデータベースに保存するといった用途に利用できます。

本章では、前章の続きから始めて、コードをアップデートしていきます。

JWT について

JWTとは「JSON Web Tokens」の略称です。

JSONオブジェクトをスペースのない長く密集した文字列で表現したトークンの仕様です。例えば次のようになります：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fw
```

これらは暗号化されていないので、誰でもコンテンツから情報を復元できてしまいます。

しかし、トークンは署名されているため、あなたが発行したトークンを受け取った人は、あなたが実際に発行したということを検証できます。

例えば、1週間の有効期限を持つトークンを作成したとします。ユーザーが翌日そのトークンを持って戻ってきたとき、そのユーザーはまだシステムにログインしていることがわかります。

1週間後、トークンが期限切れとなるとどうなるでしょうか？ユーザーは認可されず、新しいトークンを得るために再びサインインしなければなりません。また、ユーザー（または第三者）がトークンを修正して有効期限を変更しようとした場合、署名が一致しないため、トークンの修正を検知できます。

JWT トークンを使って遊んでみたいという方は、https://jwt.io をチェックしてください。

`python-jose` のインストール

PythonでJWTトークンの生成と検証を行うために、`python-jose` をインストールする必要があります：

```
$ pip install python-jose[cryptography]

---> 100%
```

また、Python-joseだけではなく、暗号を扱うためのパッケージを追加で必要とします。

ここでは、推奨されているものを使用します：pyca/cryptography。

!!! tip "豆知識" このチュートリアルでは以前、PyJWTを使用していました。

しかし、`Python-jose`は、`PyJWT`のすべての機能に加えて、後に他のツールと統合して構築する際におそらく必要となる可能性のあるいくつかの追加機能を提供しています。そのため、代わりに`Python-jose`を使用するように更新されました。

パスワードのハッシュ化

「ハッシュ化」とは、あるコンテンツ（ここではパスワード）を、規則性のないバイト列（単なる文字列）に変換することです。

特徴として、全く同じ内容（全く同じパスワード）を渡すと、全く同じ規則性のないバイト列に変換されます。

しかし、規則性のないバイト列から元のパスワードに戻すことはできません。

パスワードのハッシュ化を使う理由

データベースが盗まれても、ユーザーの平文のパスワードは盗まれず、ハッシュ値だけが盗まれます。

したがって、泥棒はそのパスワードを別のシステムで使えません（多くのユーザーはどこでも同じパスワードを使用しているため、危険性があります）。

`passlib` のインストール

PassLib は、パスワードのハッシュを処理するための優れたPythonパッケージです。

このパッケージは、多くの安全なハッシュアルゴリズムとユーティリティをサポートします。

推奨されるアルゴリズムは「Bcrypt」です。

そのため、Bcryptを指定してPassLibをインストールします：

```
$ pip install passlib[bcrypt]

---> 100%
```

!!! tip "豆知識" `passlib` を使用すると、**Django**や**Flask**のセキュリティプラグインなどで作成されたパスワードを読み取れるように設定できます。

例えば、`Django`アプリケーションからデータベース内の同じデータを`FastAPI`アプリケーションと共有できるだけではなく、同じデータベースを使用して`Django`アプリケーションを徐々に移行することもできます。

また、ユーザーは`Django`アプリまたは`**FastAPI**`アプリからも、同時にログインできるようになります。

パスワードのハッシュ化と検証

必要なツールを `passlib` からインポートします。

PassLib の「context」を作成します。これは、パスワードのハッシュ化と検証に使用されるものです。

!!! tip "豆知識" PassLibのcontextには、検証だけが許された非推奨の古いハッシュアルゴリズムを含む、様々なハッシュアルゴリズムを使用した検証機能もあります。

例えば、この機能を使用して、別のシステム（Djangoなど）によって生成されたパスワードを読み取って検証し、Bcryptなどの別のアルゴリズムを使用して新しいパスワードをハッシュするといったことができます。

そして、同時にそれらはすべてに互換性があります。

ユーザーから送られてきたパスワードをハッシュ化するユーティリティー関数を作成します。

また、受け取ったパスワードが保存されているハッシュと一致するかどうかを検証するユーティリティも作成します。

さらに、ユーザーを認証して返す関数も作成します。

```
{!../../../docs_src/security/tutorial004.py!}
```

!!! note "備考" 新しい（偽の）データベース `fake_users_db` を確認すると、ハッシュ化されたパスワードが次のようになっていることがわかります： `"$2b$12$EixZaYVK1fsbw1ZfbX3OXePaWxn96p36WQoeG6Lruj3vjPGga31lW"`

JWTトークンの取り扱い

インストールした複数のモジュールをインポートします。

JWTトークンの署名に使用されるランダムな秘密鍵を生成します。

安全なランダム秘密鍵を生成するには、次のコマンドを使用します：

```
$ openssl rand -hex 32

09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7
```

そして、出力された文字列を変数 `SECRET_KEY` にコピーします。（例に記載している秘密鍵は実際に使用しないでください）

JWTトークンの署名に使用するアルゴリズム `"HS256"` を指定した変数 `ALGORITHM` を作成します。

トークンの有効期限を指定した変数 `ACCESS_TOKEN_EXPIRE_MINUTES` を作成します。

レスポンスのトークンエンドポイントで使用するPydanticモデルを定義します。

新しいアクセストークンを生成するユーティリティ関数を作成します。

```
{!../../../docs_src/security/tutorial004.py!}
```

依存関係の更新

`get_current_user` を更新して、先ほどと同じトークンを受け取るようにしますが、今回はJWTトークンを使用します。

受け取ったトークンを復号して検証し、現在のユーザーを返します。

トークンが無効な場合は、すぐにHTTPエラーを返します。

```
{!../../../docs_src/security/tutorial004.py!}
```

`/token` パスオペレーションの更新

トークンの有効期限を表す `timedelta` を作成します。

JWTアクセストークンを作成し、それを返します。

```
{!../../../docs_src/security/tutorial004.py!}
```

JWTの"subject" `sub` についての技術的な詳細

JWTの仕様では、トークンのsubjectを表すキー `sub` があるとされています。

使用するかどうかは任意ですが、 `sub` はユーザーの識別情報を入れるように規定されているので、ここで使用します。

JWTは、ユーザーを識別して、そのユーザーがAPI上で直接操作を実行できるようにする以外にも、他の用途で使用されることがあります。

例えば、「車」や「ブログ記事」を識別することができます。

そして、「ドライブ」（車の場合）や「編集」（ブログの場合）など、そのエンティティに関する権限も追加できます。

また、JWTトークンをユーザー（またはボット）に渡すことができます。ユーザーは、JWTトークンを使用するだけで、アカウントを持っていなくても、APIが生成したJWTトークンを使ってそれらの行動（車の運転、ブログ投稿の編集）を実行できるのです。

これらのアイデアを使用すると、JWTをより高度なシナリオに使用できます。

しかしながら、それらのエンティティのいくつかが同じIDを持つ可能性があります。例えば、`foo`（ユーザー`foo`、車`foo`、ブログ投稿`foo`）などです。

IDの衝突を回避するために、ユーザーのJWTトークンを作成するとき、subキーの値にプレフィックスを付けることができます（例えば、`username:`）。したがって、この例では、`sub`の値は次のようになっている可能性があります：`username:johndoe`

覚えておくべき重要なことは、`sub`キーはアプリケーション全体で一意の識別子を持ち、文字列である必要があるということです。

確認

サーバーを実行し、ドキュメントに移動します：http://127.0.0.1:8000/docs

次のようなユーザーインターフェイスが表示されます：

前回と同じ方法でアプリケーションの認可を行います。

次の認証情報を使用します：

Username: `johndoe` Password: `secret`

!!! check "確認" コードのどこにも平文のパスワード"`secret`"はなく、ハッシュ化されたものしかないことを確認してください。

エンドポイント`/users/me/`を呼び出すと、次のようなレスポンスが得られます：

```
{
  "username": "johndoe",
  "email": "johndoe@example.com",
  "full_name": "John Doe",
  "disabled": false
}
```

開発者ツールを開くと、送信されるデータにはトークンだけが含まれており、パスワードはユーザーを認証してアクセストークンを取得する最初のリクエストでのみ送信され、その後は送信されないことがわかります。

!!! note "備考" ヘッダーの`Authorization`には、`Bearer`で始まる値があります。

`scopes`を使った高度なユースケース

OAuth2には、「スコープ」という概念があります。

これらを利用して、JWTトークンに特定の権限セットを追加することができます。

そして、このトークンをユーザーに直接、または第三者に与えて、制限付きでAPIを操作できます。

これらの使用方法や**FastAPI**への統合方法については、**高度なユーザーガイド**で後ほど説明します。

まとめ

ここまでの説明で、OAuth2やJWTなどの規格を使った安全な**FastAPI**アプリケーションを設定することができます。

ほとんどのフレームワークにおいて、セキュリティを扱うことは非常に複雑な課題となります。

簡略化しすぎたパッケージの多くは、データモデルやデータベース、利用可能な機能について多くの妥協をしなければなりません。そして、あまりにも単純化されたパッケージの中には、実はセキュリティ上の欠陥があるものもあります。

FastAPIは、どのようなデータベース、データモデル、ツールに対しても妥協することはありません。

そのため、プロジェクトに合わせて自由に選択することができます。

また、**FastAPI**は外部パッケージを統合するために複雑な仕組みを必要としないため、`passlib`や`python-jose`のようなよく整備され広く使われている多くのパッケージを直接使用することができます。

しかし、柔軟性、堅牢性、セキュリティを損なうことなく、可能な限りプロセスを簡素化するためのツールを提供します。

また、OAuth2のような安全で標準的なプロトコルを比較的簡単な方法で使用できるだけではなく、実装することもできます。

OAuth2の「スコープ」を使って、同じ基準でより細かい権限システムを実現する方法については、**高度なユーザーガイド**で詳しく説明しています。スコープ付きのOAuth2は、Facebook、Google、GitHub、Microsoft、Twitterなど、多くの大手認証プロバイダが、サードパーティのアプリケーションと自社のAPIとのやり取りをユーザーに代わっ

て認可するために使用している仕組みです。