セキュリティ - 最初の一歩

あるドメインに、**バックエンド** APIを持っているとしましょう。

そして、別のドメインか同じドメインの違うパス（またはモバイルアプリケーションの中）に **フロントエンド**を持っています。

さらに、フロントエンドが**ユーザー名**と**パスワード**を使って、バックエンドで認証する方法を用意したいです。

FastAPIでは、これを**OAuth2**を使用して構築できます。

ですが、ちょっとした必要な情報を探すために、長い仕様のすべてを読む必要はありません。

FastAPIが提供するツールを使って、セキュリティを制御してみましょう。

どう見えるか

まずはこのコードを使って、どう動くか観察します。その後で、何が起こっているのか理解しましょう。

`main.py` を作成

`main.py` に、下記の例をコピーします:

```
{!../../../docs_src/security/tutorial001.py!}
```

実行

!!! info "情報" まず `python-multipart` をインストールします。

例えば、`pip install python-multipart`。

これは、**OAuth2**が `ユーザー名` や `パスワード` を送信するために、「フォームデータ」を使うからです。

例を実行します:

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

確認

次のインタラクティブなドキュメントにアクセスしてください: http://127.0.0.1:8000/docs。

下記のように見えるでしょう:

!!! check "Authorizeボタン!" すでにピカピカの新しい「Authorize」ボタンがあります。

そして、あなたの*path operation*には、右上にクリックできる小さな鍵アイコンがあります。

それをクリックすると、 `ユーザー名` と `パスワード` (およびその他のオプションフィールド) を入力する小さな認証フォームが表示されます:

!!! note "備考" フォームに何を入力しても、まだうまくいきません。ですが、これから動くようになります。

もちろんエンドユーザーのためのフロントエンドではありません。しかし、すべてのAPIをインタラクティブにドキュメント化するための素晴らしい自動ツールです。

フロントエンドチームはこれを利用できます (また、あなたも利用できます) 。

サードパーティのアプリケーションやシステムでも使用可能です。

また、同じアプリケーションのデバッグ、チェック、テストのためにも利用できます。

`パスワード` フロー

では、少し話を戻して、どうなっているか理解しましょう。

`パスワード` の「フロー」は、OAuth2で定義されているセキュリティと認証を扱う方法 (「フロー」) の1つです。

OAuth2は、バックエンドやAPIがユーザーを認証するサーバーから独立したものとして設計されていました。

しかし、この場合、同じ**FastAPI**アプリケーションがAPIと認証を処理します。

そこで、簡略化した箇所から見直してみましょう:

- ユーザーはフロントエンドで `ユーザー名` と `パスワード` を入力し、 `Enter` を押します。
- フロントエンド (ユーザーのブラウザで実行中) は、 `ユーザー名` と `パスワード` をAPIの特定のURL (`tokenUrl="token"` で宣言された) に送信します。
- APIは `ユーザー名` と `パスワード` をチェックし、「トークン」を返却します (まだ実装していません)。
    - 「トークン」はただの文字列であり、あとでこのユーザーを検証するために使用します。
    - 通常、トークンは時間が経つと期限切れになるように設定されています。
        - トークンが期限切れの場合は、再度ログインする必要があります。
        - トークンが盗まれたとしても、リスクは低いです。永久キーのように永遠に機能するものではありません（ほとんどの場合）。
- フロントエンドはそのトークンを一時的にどこかに保存します。
- ユーザーがフロントエンドでクリックして、フロントエンドのWebアプリの別のセクションに移動します。
- フロントエンドはAPIからさらにデータを取得する必要があります。
    - しかし、特定のエンドポイントの認証が必要です。
    - したがって、APIで認証するため、HTTPヘッダー `Authorization` に `Bearer` の文字列とトークンを加えた値を送信します。
    - トークンに `foobar` が含まれている場合、 `Authorization` ヘッダーの内容は次のようになります: `Bearer foobar` 。

FastAPIの `OAuth2PasswordBearer`

FastAPIは、これらのセキュリティ機能を実装するために、抽象度の異なる複数のツールを提供しています。

この例では、**Bearer**トークンを使用して**OAuth2**を**パスワード**フローで使用します。これには `OAuth2PasswordBearer` クラスを使用します。

!!! info "情報" 「bearer」トークンが、唯一の選択肢ではありません。

しかし、私たちのユースケースには最適です。

あなたがOAuth2の専門家で、あなたのニーズに適した別のオプションがある理由を正確に知っている場合を除き、ほとんどのユースケースに最適かもしれません。

その場合、**FastAPI**はそれを構築するためのツールも提供します。

`OAuth2PasswordBearer` クラスのインスタンスを作成する時に、パラメーター `tokenUrl` を渡します。このパラメーターには、クライアント (ユーザーのブラウザで動作するフロントエンド) がトークンを取得するために `ユーザー名` と `パスワード` を送信するURLを指定します。

```
{!../../../docs_src/security/tutorial001.py!}
```

!!! tip "豆知識" ここで、 `tokenUrl="token"` は、まだ作成していない相対URL `token` を指します。相対URLなので、 `./token` と同じです。

相対URLを使っているので、APIが`https://example.com/`にある場合、`https://example.com/token`を参照します。しかし、APIが`https://example.com/api/v1/`にある場合は`https://example.com/api/v1/token`を参照することになります。

相対 URL を使うことは、[プロキシと接続](./.../advanced/behind-a-proxy.md){.internal-link target=_blank}のような高度なユースケースでもアプリケーションを動作させ続けるために重要です。

このパラメーターはエンドポイント/ *path operation*を作成しません。しかし、URL `/token` はクライアントがトークンを取得するために使用するものであると宣言します。この情報は OpenAPI やインタラクティブな API ドキュメントシステムで使われます。

実際のpath operationもすぐに作ります。

!!! info "情報" 非常に厳格な「Pythonista」であれば、パラメーター名のスタイルが `token_url` ではなく `tokenUrl` であることを気に入らないかもしれません。

それはOpenAPI仕様と同じ名前を使用しているからです。そのため、これらのセキュリティスキームについてもっと調べる必要がある場合は、それをコピーして貼り付ければ、それについての詳細な情報を見つけることができます。

変数 `oauth2_scheme` は `OAuth2PasswordBearer` のインスタンスですが、「呼び出し可能」です。

次のように、呼ぶことができます:

```
oauth2_scheme(some, parameters)
```

そのため、 `Depends` と一緒に使うことができます。

使い方

これで `oauth2_scheme` を `Depends` で依存関係に渡すことができます。

```
{!../../../docs_src/security/tutorial001.py!}
```

この依存関係は、*path operation function*のパラメーター `token` に代入される `str` を提供します。

FastAPIは、この依存関係を使用してOpenAPIスキーマ (および自動APIドキュメント) で「セキュリティスキーム」を定義できることを知っています。

!!! info "技術詳細" **FastAPI**は、 `OAuth2PasswordBearer` クラス (依存関係で宣言されている) を使用してOpenAPIのセキュリティスキームを定義できることを知っています。これは `fastapi.security.oauth2.OAuth2` 、 `fastapi.security.base.SecurityBase` を継承しているからです。

OpenAPIと統合するセキュリティユーティリティ （および自動APIドキュメント） はすべて`SecurityBase`を継承しています。それにより、**FastAPI**はそれらをOpenAPIに統合する方法を知ることができます。

どのように動作するか

リクエストの中に `Authorization` ヘッダーを探しに行き、その値が `Bearer` と何らかのトークンを含んでいるかどうかをチェックし、そのトークンを `str` として返します。

もし `Authorization` ヘッダーが見つからなかったり、値が `Bearer` トークンを持っていなかったりすると、401ステータスコードエラー (`UNAUTHORIZED`) で直接応答します。

トークンが存在するかどうかをチェックしてエラーを返す必要はありません。関数が実行された場合、そのトークンに `str` が含まれているか確認できます。

インタラクティブなドキュメントですでに試すことができます:

まだトークンの有効性を検証しているわけではありませんが、これはもう始まっています。

まとめ

つまり、たった3~4行の追加で、すでに何らかの基礎的なセキュリティの形になっています。