JWT

JSON Web Token (JWT) is an internet standard for creating access tokens based on JSON. They don't need to be stored in a database: the data is self-contained inside and cryptographically signed.

Configuration

```python
from fastapi_users.authentication import JWTStrategy

SECRET = "SECRET"

def get_jwt_strategy() -> JWTStrategy:
    return JWTStrategy(secret=SECRET, lifetime_seconds=3600)
```

As you can see, instantiation is quite simple. It accepts the following arguments:

- `secret` (`Union[str, pydantic.SecretStr]`): A constant secret which is used to encode the token. **Use a strong passphrase and keep it secure.**
- `lifetime_seconds` (`Optional[int]`): The lifetime of the token in seconds. Can be set to `None` but in this case the token will be valid **forever**; which may raise serious security concerns.
- `token_audience` (`Optional[List[str]]`): A list of valid audiences for the JWT token. Defaults to `["fastapi-users:auth"]`.
- `algorithm` (`Optional[str]`): The JWT encryption algorithm. See RFC 7519, section 8. Defaults to `"HS256"`.
- `public_key` (`Optional[Union[str, pydantic.SecretStr]]`): If the JWT encryption algorithm requires a key pair instead of a simple secret, the key to **decrypt** the JWT may be provided here. The `secret` parameter will always be used to **encrypt** the JWT.

Why it's inside a function?

To allow strategies to be instantiated dynamically with other dependencies, they have to be provided as a callable to the authentication backend.

For `JWTStrategy`, since it doesn't require dependencies, it can be as simple as the function above.

RS256 example

```
from fastapi_users.authentication import JWTStrategy

PUBLIC_KEY = """-----BEGIN PUBLIC KEY-----
# Your RSA public key in PEM format goes here
-----END PUBLIC KEY-----"""

PRIVATE_KEY = """-----BEGIN RSA PRIVATE KEY-----
# Your RSA private key in PEM format goes here
-----END RSA PRIVATE KEY-----"""

def get_jwt_strategy() -> JWTStrategy:
    return JWTStrategy(
        secret=PRIVATE_KEY,
        lifetime_seconds=3600,
        algorithm="RS256",
        public_key=PUBLIC_KEY,
    )
```

Logout

On logout, this strategy **won't do anything**. Indeed, a JWT can't be invalidated on the server-side: it's valid until it expires.