zopyx-fastapi-auth

An opinionated security solution for FastAPI

Andreas Jung • info@zopyx.com • www.zopyx.com • www.andreas-jung.com

Agenda

- Introduction to FastAPI
- Quick intro intro authentication and authorization
- Key concepts of fastapi-auth
- Integration and usage in FastAPI
- Pluggable user source architecture
- User management
- Demo time
- Q & A

About me

- Software developer & software architect
- Requirements engineering
- Python developer & Python expert

Publishing developer & consultant
since mid-90s

- CMS Solutions
- Publishing Solutions
- Individual software solutions

Funder of print-css.rocks project
Independent PrintCSS consulting

- NGOs, EDU
- medical & pharmaceutical
- energy
- research & development quantum mechanics

https://www.zopyx.com
https://andreas-jung.com
✉ info@zopyx.com

Introduction to FastAPI

What is FastAPI?

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3 based on standard Python type hints and the data-validation framework Pydantic.

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Reasons for using FastAPI

- **Use of Pydantic**
  FastAPI uses Pydantic for data validation and serialization, ensuring robust input and output handling.
- **Full Async Support**
  FastAPI supports asynchronous programming, enabling high-performance and efficient request handling.
- **Automatic OpenAPI Generation**
  FastAPI automatically generates OpenAPI documentation, simplifying API development and testing.
- **It is Fast**
  FastAPI is optimized for speed, making it one of the fastest web frameworks available.
- **High Adoption Rate**
  FastAPI's growing popularity and community support make it a reliable choice for modern web development.

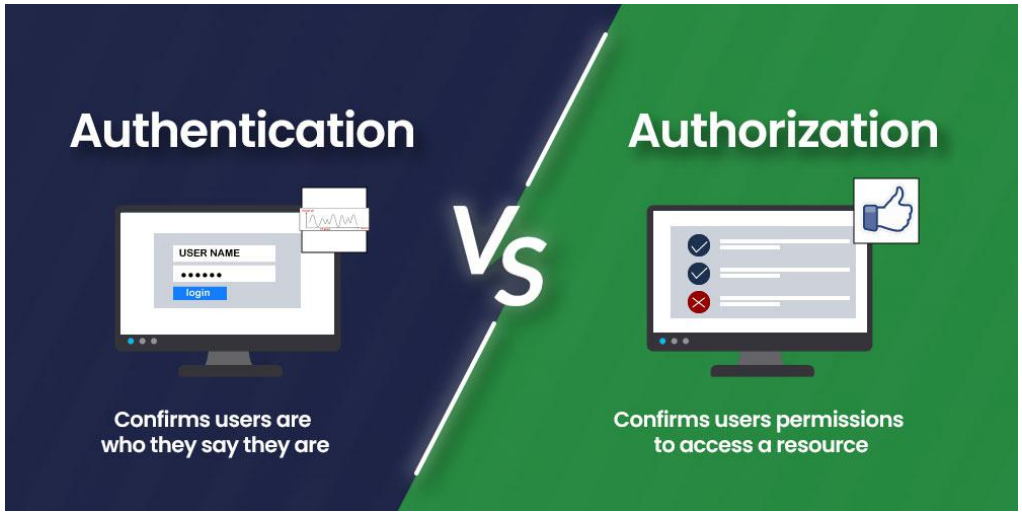
Authentication ↔ Authorization

What is authentication?

- Authentication verifies the identity of a user, device, or entity.
- It uses credentials like **username + passwords**, biometrics, or security tokens.
- This ensures only authorized access to sensitive information or resources.

What is authorization?

- Authorization determines what resources a user or system can access.
- It occurs after authentication and enforces permissions and policies.
- This ensures users can only perform actions they are permitted to.



zopyx-fastapi-auth

An opinionated authentication solution for FastAPI

Why zopyx-fastapi-auth

...when there are already many other solutions!?

- **Usecase**
  Migration of a CMS Plone-based application from Plone
  to FastAPI for the University of Saarbrücken.
- **Problem**
  - None of the existing authentication frameworks for FastAPI has fit our needs
  - No direct translation of security concepts of the legacy system
    available for FastAPI
  - Transparent authentication against three different
    user sources (local, SAP, LDAP)

Concepts: Permissions

An access right that defines what actions a user or role can perform, such as viewing, editing, or deleting resources within an application.

Permissions are defined as code.

```
from fastapi_auth.permissions import Permission

VIEW_PERMISSION = Permission(name="view", description="View permission")
EDIT_PERMISSION = Permission(name="edit", description="Edit permission")
DELETE_PERMISSION = Permission(name="delete", description="Delete permission")
```

Concepts: Roles

A collection of permissions wrapped together as a role.

Roles define a user's access level and responsibilities within an application.

A user can have multiple roles, each granting a different set of permissions.

Roles are defined in code.

```
from fastapi_auth.permissions import  Role

ADMIN_ROLE = Role(
    name="Administrator",
    description="Admin role",
    permissions=[VIEW_PERMISSION, EDIT_PERMISSION, DELETE_PERMISSION],
)

USER_ROLE = Role(
    name="User",
    description="User role",
    permissions=[VIEW_PERMISSION, EDIT_PERMISSION],
)

VIEWER_ROLE = Role(
    name="Viewer",
    description="Viewer role",
    permissions=[VIEW_PERMISSION],
)
```

Concepts: Roles II

Roles must be registered with the (gobal) roles registry.
Interim solution...likely to go away soon.

```
from fastapi_auth.roles import ROLES_REGISTRY

ROLES_REGISTRY.register(ADMIN_ROLE)
ROLES_REGISTRY.register(USER_ROLE)
ROLES_REGISTRY.register(VIEWER_ROLE)
```

Concepts: Users

A user represents the currently user context accessing the system.

User types:

- Anonymous User
- Authenticated User
- (Superuser)

Users are stored in (external) user sources like a database, LDAP etc.

The User object in fastapi-auth

- The `Protected()` method is used to protect an endpoint by permission, role or a custom checker

```
@app.get(„/admin")
def admin(user: User = Depends(Protected(<protection_conditions)):
    return {"user": user}
```

- Integration with FastAPI through **dependency injection**
- Successful authorization will return a `User` object
- Unsuccessful authorization ➡️ HTTP 403/Forbidden
- `User` properties: `name`, `description`, `roles`, `is_anonymous`

Concepts II

- A user can have multiple roles
- A role can multiple permissions

How to use fastapi-auth in your FastAPI app?

Endpoint protection by role(s)

```
# This is an endpoint that requires the user to be authenticated.  In this case,
# the user must have the ADMIN_ROLE role.  It is also possible to require a
# permission instead.  Use the Protected dependency to require authentication.
# An unauthenticated request as ANONYMOUS_USER will be rejected.

@app.get(„/admin")
def admin(user: User = Depends(Protected(required_roles=[ADMIN_ROLE]))):
    return {"user": user}
```

How to use fastapi-auth in your FastAPI app?

Endpoint protection by permission

```
from fastapi_auth.dependencies import Protected

@app.get(„/admin2")
def admin2(user: User = Depends(Protected(required_permission=VIEW_PERMISSION))):
    return {"user": user}
```

How to use fastapi-auth in your FastAPI app?

Endpoint protection by custom checker

```
from fastapi_auth.dependencies import Protected

def my_check(request: Request, user: User) -> bool:
    # perform some checks based on request and/or user....
    return True # or False

@app.get(„/admin3")
def admin3(user: User = Depends(Protected(required_checker=my_check))):
    return {"user": user}
```

Concept of user sources

- A user source manages user accounts
- A user source can authenticate (or not) a user based on the parameters of a login form
- …usually determined by validating the username and the password against the data stored in a database
- Typical: RDBMS, LDAP etc.

Pluggable authenticators

```
from fastapi import Request
from fastapi.authenticator_registry import Authenticator, AUTHENTICATOR_REGISTRY
from fastapi.users import User

class MyAuthenticator(Authenticator):

    async def authenticate(request: Request) -> User:

        # extract credentials from request
        username = request.form....
        password = request.form....

        # perform authentication against your own authentication system
        user_data = my_backend.authenticate_user(username, password)

        return User(name=user_data["name"], roles=[...])

AUTHENTICATOR_REGISTRY.add_authenticator(MyAuthenticator(), 0)
```

Build-in user management

- zopyx-fastapi-auth comes with a default user management
- with sqlite as database backend
- commandline utility `fastapi-auth-user-admin`

```
> fastapi-auth-user-admin add <username> <password> „Role1,Role2…“
> fastapi-auth-user-admin delete <username>
> fastapi-auth-user-admin lists-users
> fastapi-auth-user-admin set-password <username> <new-password>
```

Internals

- based on *starlette.middleware.sessions*
- authentication information stored and exchanged through an encrypted cookie (symmetric encryption)
- client/browser can not decrypt the cookie
- lifetime of cookie: session or persistent
- **ToDo:**
  - **switching to secure cookies (starlette-securecookies)**
  - **better control over lifetime of cookies, expiration, revocation!?**

Demo time 🥳

Resources

- https://github.com/zopyx/fastapi-auth
- https://pypi.org/project/zopyx-fastapi-auth/

Questions & Answers

Thank you for listening

Andreas Jung • info@zopyx.com • www.zopyx.com • www.andreas-jung.com