0. Topics

Thursday, May 29, 2025 10:03 PM

1. **Middlewares**
2. **Dependency Injection**
3. **JWT Authentication**
4. **Managing API Keys**
5. **Best Practices**

1. Middleware

Thursday, May 29, 2025 10:08 PM

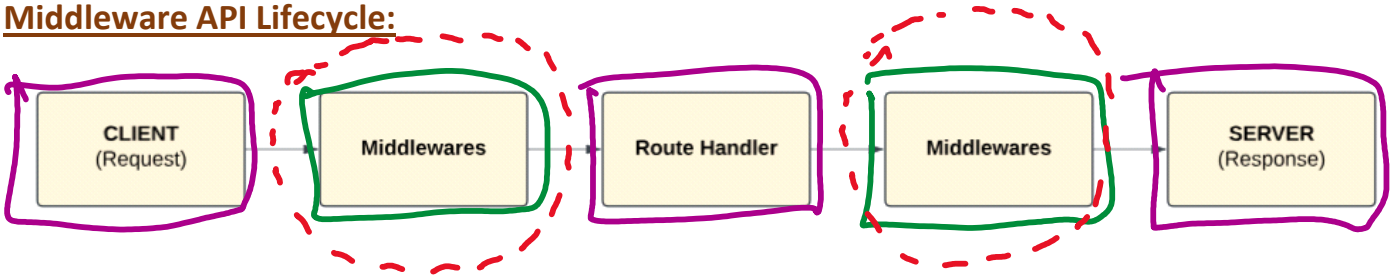
What is a Middleware?

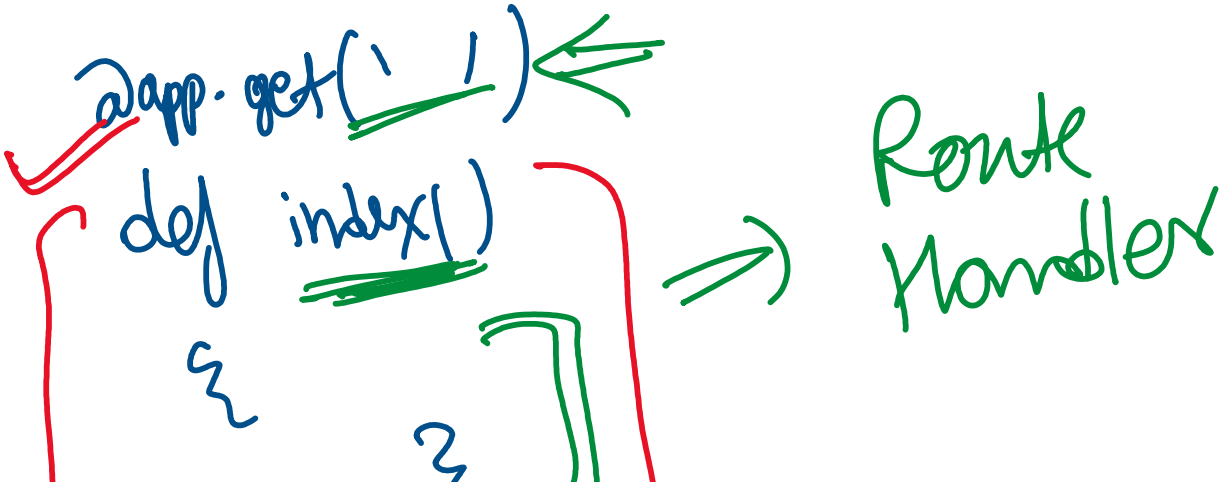
- Middleware is a function or class that intercepts incoming requests before they reach the route handler, or outgoing responses after the route handler has processed the request
- Middleware is a function that runs before or after each request in the application
- Middleware allows us to:
  - **Log requests and responses**
  - **Handle CORS or custom headers**
  - **Measure performance**
  - **Catch and process errors globally**

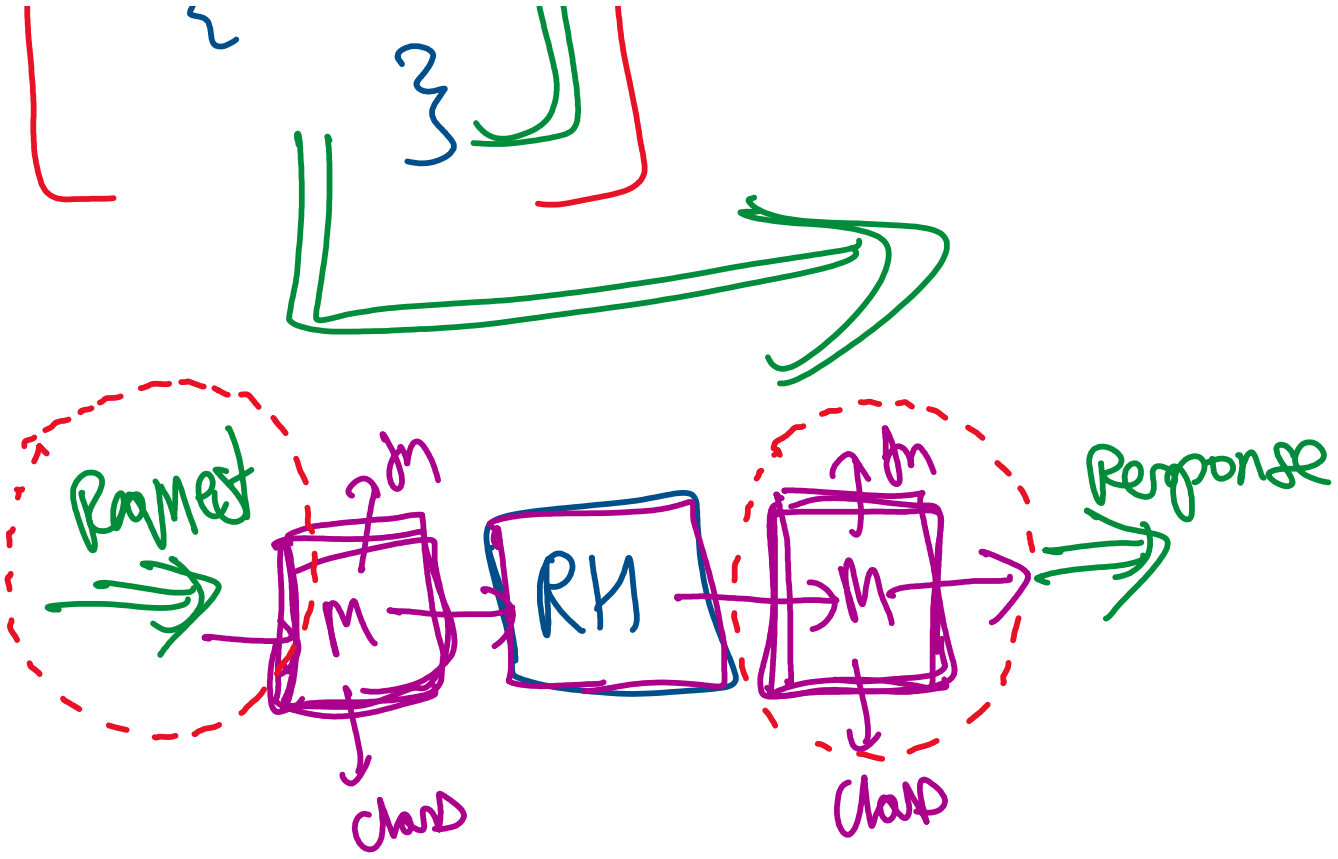
Main Idea Behind Middleware:

- FastAPI uses the concept of middleware similar to other modern frameworks like Node.js or Django
- Middleware runs in a **chain**, one after another, in the order they are registered
- Middleware can modify:
  - **The request before it's passed to the endpoint**
  - **The response returned by the endpoint**

Middleware API Lifecycle:






2
3
Request
M
RH
M
Response
Close
Close

1.1 - Built-in Middlewares

Thursday, May 29, 2025 11:16 PM

1. CORS Middleware:

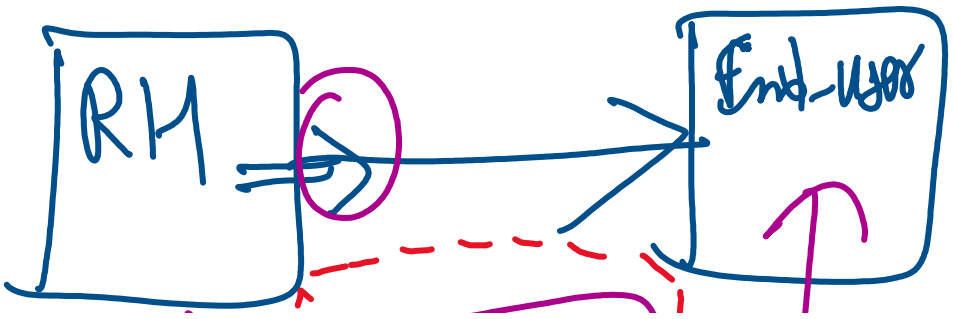
- Cross-Origin Resource Sharing
- Allows cross-origin requests from browsers
- Injects the proper headers in the HTTP responses so that compliant browsers will permit—or reject—those cross-origin calls
- Helps declare exactly which external origins, HTTP methods, headers (and even whether credentials/cookies) are allowed
  - **allow_origins**: which domains are permitted to make browser-based requests
  - **allow_credentials**: whether to let the browser send cookies
  - **allow_methods**: which actions will be accepted
  - **allow_headers**: which headers will be accepted

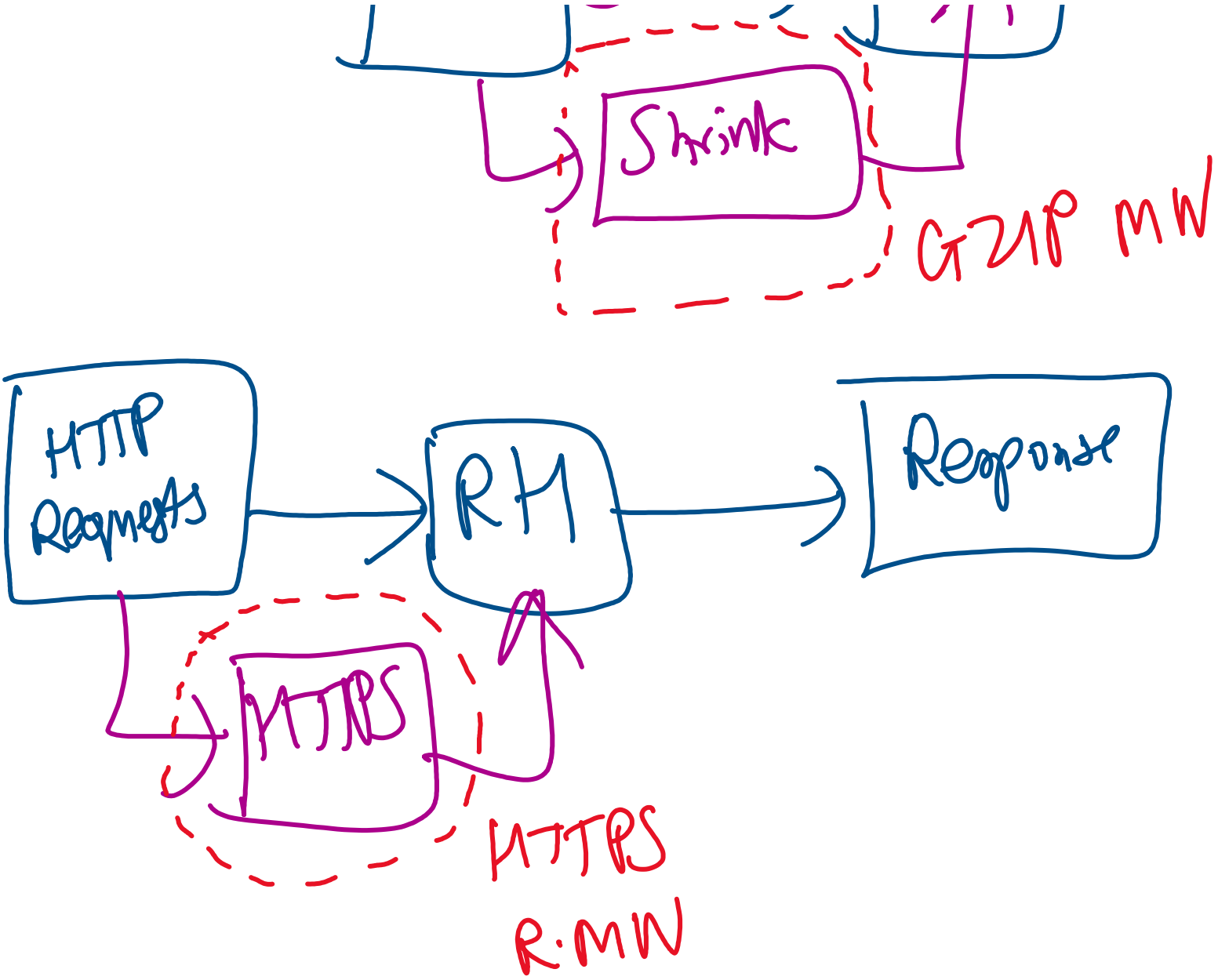
2. GZip Middleware:

- Used to compress HTTP responses before sending them to the client
- Reduces the size of data transferred over the network
- Faster page loads
- Lower bandwidth usage
- Improved performance, especially over slower networks

3. HTTPSRedirectMiddleware:

- Ensures that all incoming HTTP requests are automatically redirected to HTTPS
- Enforcing HTTPS is a modern security standard, which is preferred by many platforms (browsers, Google SEO ranking)




Shrink
GZIP MW
HTTP Requests
RH
Response
HTTPS
HTTPS
R·MW

1.2 - Custom Middlewares

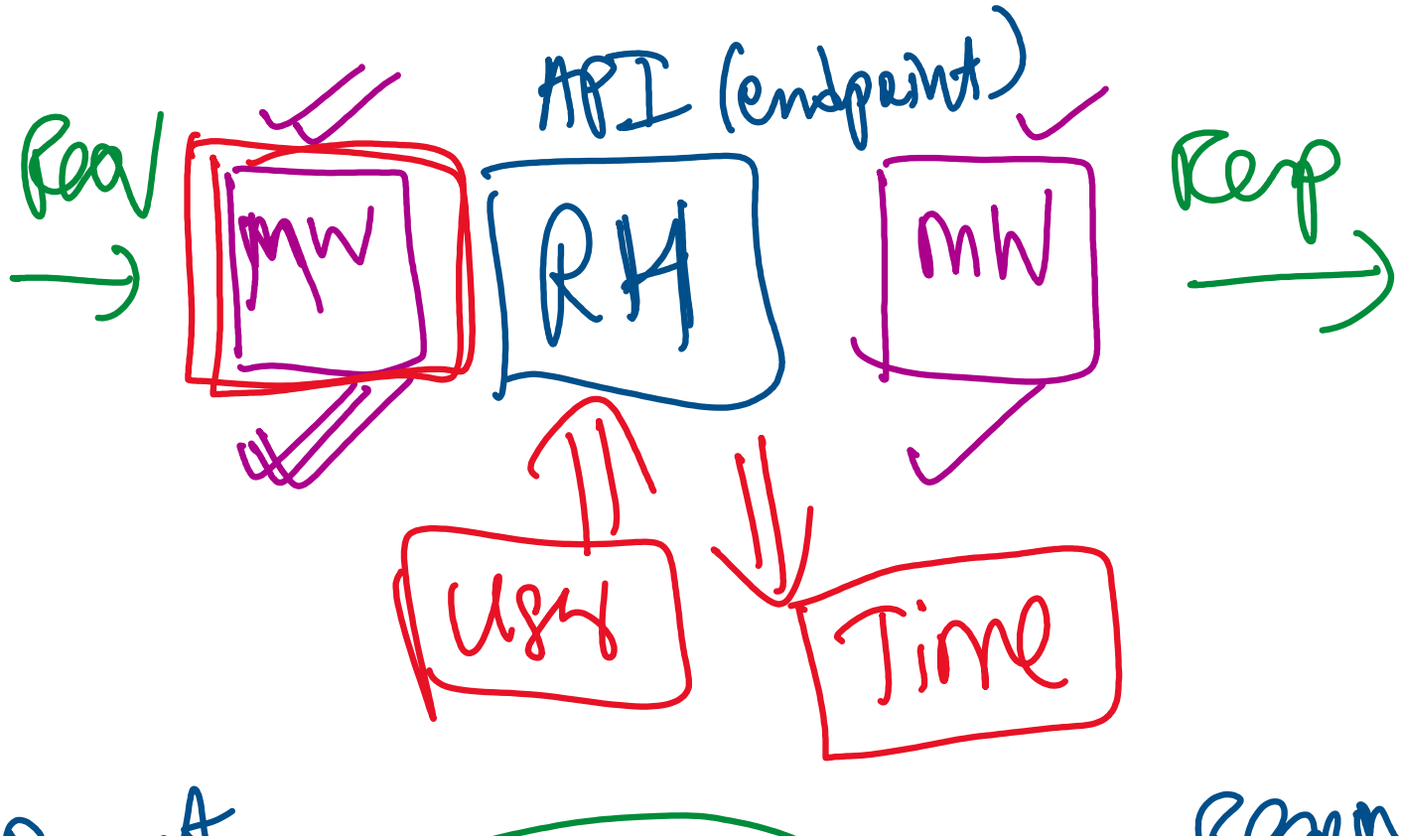
Thursday, May 29, 2025 11:17 PM

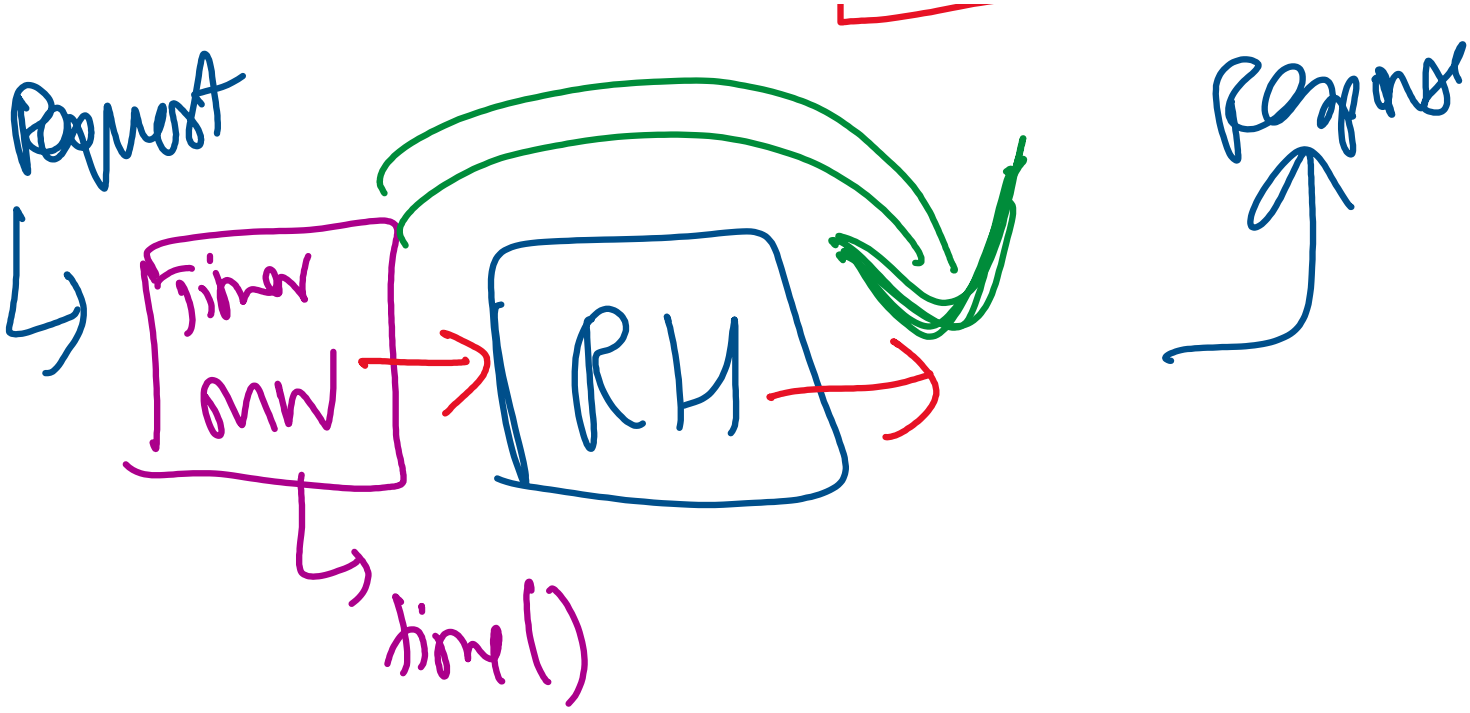
Implement a Custom Middleware:

- Define a class for the middleware
- Implement the dispatch method which contains the logic
- Register the middleware
- Add a route to handle requests
- Run the application
- Observe the terminal logs

Summary:

COMPONENT	PURPOSE
BaseHTTPMiddleware	Base class to create custom middleware
dispatch()	Handles all requests and lets you add custom logic
call_next(request)	Passes request to the next layer (middleware or route handler)
print()	Logs the time taken for each request
app.add_middleware()	Tells FastAPI to run the custom middleware for every request




Request
Timer
MW
RH
time()
Response

2. Dependency Injection

Friday, May 30, 2025 12:06 AM

What is Dependency Injection?

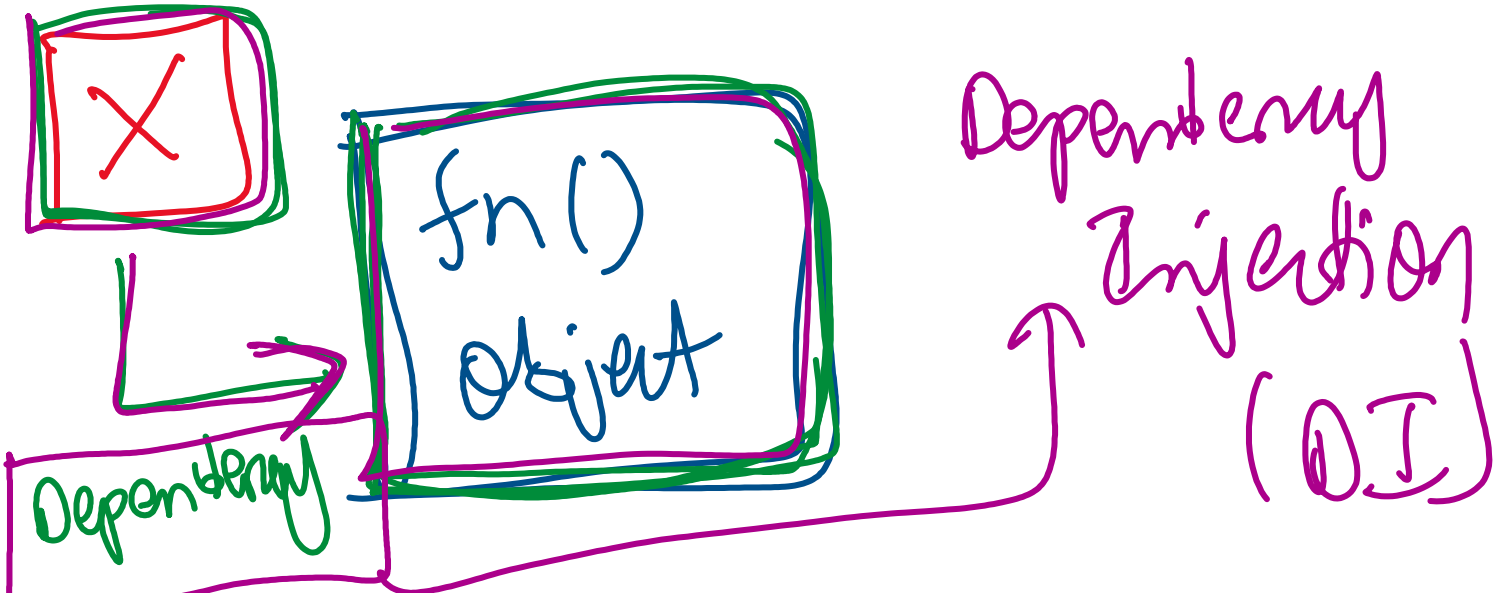
- Dependency Injection (DI) is a software design pattern that allows objects or functions to receive their dependencies from external sources rather than creating them internally
- FastAPI uses the **Depends** class to resolve and inject dependencies automatically

Common Use Cases of Dependency Injection:

1. **Database Connections**
2. **Configuration Management**
3. **User Authentication**
4. **Background Task Setup**

Best Practices:

PRACTICE	DESCRIPTION
Keep dependencies pure	No side effects; easier to test and reuse
Use classes for related parameters	Better structure and organization
Avoid heavy computation inside dependencies	Keep them fast and efficient
Use yield for resource management	Useful for DB sessions, file access, etc.
Group reusable logic	Like authentication, configuration, logging
Apply dependencies at router/middleware level	For authentication, rate limiting, etc.
Override dependencies in tests	Isolate logic and improve test reliability



2.1 - Database Connections

Friday, May 30, 2025 12:37 AM

- Database connection is required and must be established before any database operation is performed
- **get_db()** is a dependency function
- **Depends(get_db)** tells FastAPI to call **get_db()** before the request and inject the result into the **db** parameter

2.2 - Configuration Management

Friday, May 30, 2025 12:37 AM

- Provides a centralized way to store and access configuration values (ex. **api_key**, **debug**)
- Injects these values into FastAPI route handlers using **Depends()**
- All config values are encapsulated in one place (**Settings**), improving maintainability
- API keys, database URLs, etc., can be loaded here securely

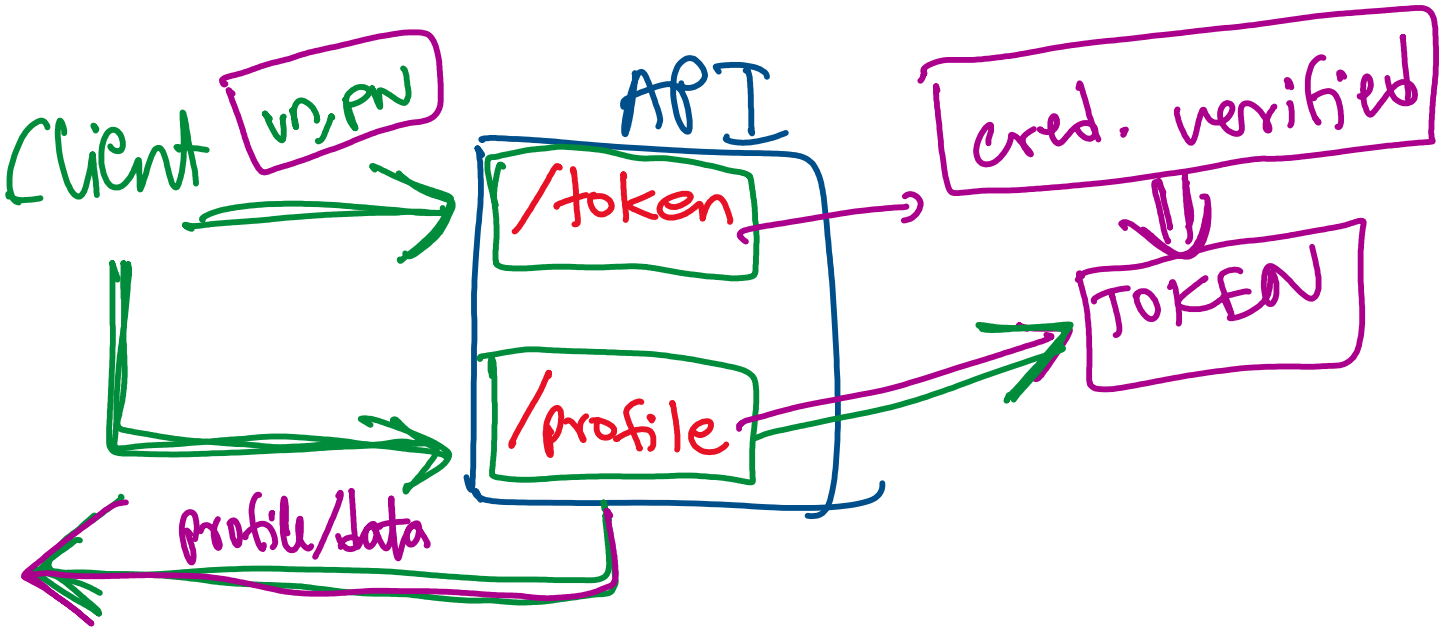
When the /config/ endpoint is called:

- FastAPI calls **get_settings()**
- The result is passed into the **get_config** function as the **settings** argument
- The function returns a dictionary containing the **api_key**

2.3 - User Authentication

Friday, May 30, 2025 12:37 AM

- The main idea is to protect an API route so that only authenticated users with a valid token can access it
- **oauth2_scheme** is a dependency provided by FastAPI:
  - Extracts the token from **Authorization** header
  - Automatically makes the token string available for further use
- **decode_token(token)** is a placeholder to implement logic:
  - Validate the token
  - Decode the token
  - Return the user data embedded in the token
- **get_current_user()**: Makes user info available to routes
- **Depends(get_current_user)**: Protects the route—only accessible if authenticated
- If no token is provided or it's invalid, FastAPI automatically raises a 401 error
- If the token is valid, the endpoint returns the username of the logged-in user



3. JWT Authentication

Friday, May 30, 2025 10:04 PM

What is JWT?

- JWT (**JSON Web Token**) is a compact, URL-safe means of representing claims between two parties
- JWTs allow to **validate user identity** and **protect secure routes** without needing to store session state on the server
- Commonly used for authentication and information exchange

JWT Structure:

- **Header** – Metadata (e.g., algorithm used)
- **Payload** – Claims like user ID, expiration time, etc.
- **Signature** – Ensures token integrity using secret key

Installation: Install dependencies required for secure token handling and password hashing

```
pip install fastapi uvicorn authlib passlib[bcrypt]
```

- **authlib:**
  - Used for building **authentication and authorization systems**
  - Implements modern security protocols like **OAuth2** and **JWT** properly
  - Eliminates the need to combine multiple packages
  - Smooth integration with FastAPI, Flask, etc.
- **passlib[bcrypt]**
  - Is a password hashing library
  - Mainly used for securely hashing and verifying passwords
  - **[bcrypt]** installs **bcrypt**, a secure hashing algorithm, used for storing passwords in a safe way

Implementation of JWT Authentication for Login in FastAPI:

- **auth.py**
- **models.py**
- **utils.py**
- **main.py**

3.1 - auth.py

Friday, May 30, 2025 10:21 PM

Purpose:

- This module handles **JWT** creation and verification
- This is the security engine of the application
- This is essential for:
  - Creating **access tokens** after successful user authentication
  - Verifying and decoding **access tokens** to authorize users accessing **protected routes**

Constants:

- **SECRET_KEY:**
  - The application's secret key
  - Must be kept confidential (e.g., in a .env file)
  - Used to sign tokens
- **ALGORITHM:**
  - Specifies the signing algorithm
  - **HS256** is widely used (HMAC with SHA-256)
- **ACCESS_TOKEN_EXPIRE_MINUTES:**
  - Token expiry time (in minutes)
  - Controls how long a token remains valid

Functions:

- **create_access_token:**
  - Generates a JWT access token using user data (typically the username or user ID)

- Makes a copy to avoid mutating the original
- Adds an expiry claim to the token payload; JWTs must contain expiration for security
- Encodes and signs the token using the secret and algorithm

- **verify_token:**
  - Verifies and decodes a token to extract user identity and check validity
  - Decodes the token using the secret key and expected algorithm
  - Raises **JoseError** if signature or expiry is invalid
  - Returns the user identifier extracted from token

3.2 - models.py

Friday, May 30, 2025 10:21 PM

Why use UserInDB(User)?

- Reuse **username** and **password** fields
- Add **hashed_password** for DB logic only
- Clearly separate:
  - What the client sends (**User**)
  - What the server works with internally (**UserInDB**)
- This structure improves code reusability, security, and clarity

3.3 - utils.py

Friday, May 30, 2025 10:21 PM

Purpose:

- The purpose of this module is to encapsulate utility functions related to user data and password hashing/verification
- Acts as a helper module that makes the codebase more modular and reusable

3.4 - main.py

Friday, May 30, 2025 10:21 PM

Purpose:

- Connects all the pieces (auth.py, utils.py, models.py) and defines the API endpoints
- The login endpoint issues JWT tokens
- A protected route is created, accessible only with a valid token

Endpoints:

- **/token:**
  - This endpoint authenticates the user
  - Try to fetch the user using **get_user()**
  - If not found, raise an error with status code 400
  - Verify the password using **verify_password()**
  - If incorrect, raise an error with status code 400
- **/users:**
  - Extracts the token from the **Authorization** header and injects it into the **token** parameter
  - Decodes and validates the JWT
  - Returns the username encoded in the token

Execution:

- Open the Swagger UI
- Execute the endpoint **/token** and give the below details:
  - **Username**: johndoe
  - **Password**: secret123
- Save the returned **access_token**
- Click the **Authorize** button (top-right of Swagger UI) and paste the token

- Execute the endpoint **/users** to receive the username

3.5 - Workflow

Friday, May 30, 2025 11:40 PM

1. **User Login and Token Issuing (/token):**
   - The client sends a **POST** request to with **username** and **password**
   - The app checks if the user exists (**get_user** from a fake DB)
   - It verifies the plain password against the stored hashed password (**verify_password**)
   - If valid, the app creates a JWT access token (using a secret key and algorithm)
   - It sends back the token in the response, which the client will use for subsequent requests
   - This step shows **credential verification and token issuance** — the core of logging in

2. **Token Usage for Protected Routes (/users):**
   - This endpoint requires a token — it uses **Depends(oauth2_scheme)** and extracts the token from the **Authorization** header
   - The token is verified (**verify_token**) and decodes the JWT to confirm it's valid and not expired
   - The username is extracted from the token payload
   - The endpoint returns info about the current user (in this case, the **username**)
   - This step shows **token validation and access control** — only users with a valid token can access this protected endpoint

3. **Password Hashing & Verification:**
   - User passwords are never stored or transmitted in plain text
   - Passwords are hashed (using **bcrypt** via **passlib**) when stored
   - On login, the plain password entered is verified against the stored hash using **verify_password**
   - This ensures **passwords are kept secure**

4. Managing API Keys

Friday, May 30, 2025 11:53 PM

What is an API Key?

- An API key is like a password for accessing a web service
- It's a long string of letters and numbers that:
  - **Identifies who is making the request**
  - **Verifies whether they have permission to access the data/service**
  - **Helps the API provider track usage**

Why Do We Need API Keys?

1. **Authentication & Authorization:**
   - To verify that a request is coming from a trusted source
   - Some APIs have public access, but many restrict data to **authenticated users**

2. **Rate Limiting:**
   - Prevents abuse by limiting how many requests a single user or app can make in a given time
   - Without API keys, it's hard to control **spam or overload**

3. **Usage Tracking:**
   - Helps the API provider analyze how their service is being used
   - Enables billing or quota enforcement for paid APIs

4. **Security:**
   - Prevents unauthorized access to sensitive data or actions
   - Can be revoked or rotated if compromised

Implementing API Keys with FastAPI via:

- **Headers**
- **Environment Variables (.env)**

4.1 - Headers

Saturday, May 31, 2025 6:29 AM

- **Implement the application**

- **Run the server**

- **Execute the curl command:**
  - **curl -H "api-key: my-secret-key" http://localhost:8000/get-data**

4.2 - Environment Variables

Saturday, May 31, 2025 6:29 AM

- **Create a .env file and include the value:**
  - **api_key=my-secret-key**

- **Implement the application**
  - Install pydantic-settings (if not already)
  - **pip install pydantic-settings**

  - **BaseSettings** is a great utility that reads environment variables and casts them to the correct type
  - The **Config** class tells Pydantic to load from a **.env** file
  - **settings.api_key** can now be accessed securely in the app

- **Run the server**

- **Execute the curl command:**
  - **curl -H "api-key: my-secret-key" http://localhost:8000/get-data**

5. Best Practices

Friday, May 30, 2025 11:54 PM

- Use **HTTPS** in production
- Store **JWT** secrets and **API keys** in **.env** file or **secure vaults**
- Set proper **CORS** and **CSRF** protections:
    - **CORS (Cross-Origin Resource Sharing):**
        - Security feature implemented by web browsers to control how web pages from one origin (domain) can request resources from another origin
    - **CSRF (Cross-Site Request Forgery):**
        - Web security vulnerability where a malicious website tricks a user's browser into performing unwanted actions on a different website where the user is authenticated
- Use hashing (e.g., **bcrypt**) for passwords
- Set token expiration time
- Implement **Role-Based Access Control** (**RBAC**) where needed