International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 06 | June - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

FastAPI vs. The Competition: A Security Feature Showdown with a Proposed Model for Enhanced Protection

Utkarsh Khandelwal[1], Ashwini KB[2]

[1]Department of Information Science Engineering, R.V. College of Engineering, India
[2]Department of Information Science Engineering, R.V. College of Engineering, India

---

Abstract - API security is of paramount importance in modern web applications, as it protects sensitive data and ensures authorized access to resources. FastAPI, a Python-based web framework, offers various security features to developers for building secure APIs. This paper examines the security features provided as well as supported by FastAPI, The provided features consists of OAuth2 authentication, Dependency injection and Security Schemas and Scopes, whereas the supported security features includes Json Web Token (JWT) authentication, Cross Site Request Forgery (CSRF) token support and HTTPS support. The supported features can be implemented by using python libraries and middlewares. The proposed model combines the JWT token authentication, OAuth2 authentication, and Security Scopes and schema security features to enhance the security in the FastAPI application. The proposed model defines scopes for users and uses JWT to generate tokens and using OAuth2 authentication service to only allow a user who has the specific permissions and scopes to access and perform actions in the scope specific API endpoints. The model hence allows secure and safe working of applications by eliminating the threat of unauthorized users to corrupt the application code. The focus on performance and security makes FastAPI an excellent choice for developers seeking to build secure APIs. Overall, this paper highlights the importance of API security and showcases FastAPI's security features, demonstrating how developers can leverage FastAPI to build robust, performant, and secure APIs.

Key Words: FastAPI, Flask, Django, OAuth2, JWT (JSON Web Token), CSRF (Cross Site Request Forgery)

1.INTRODUCTION *(Size 11, Times New roman)*

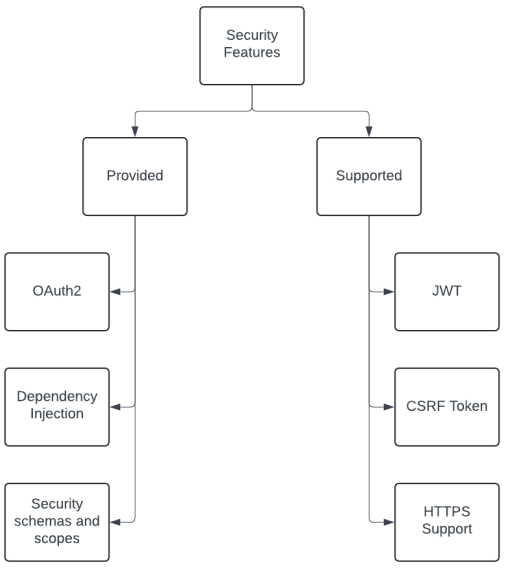
The ever-expanding world of web applications hinges on the seamless exchange of data facilitated by APIs (Application Programming Interfaces). As APIs become the cornerstone of complex systems, ensuring their security becomes paramount. Traditional approaches to API development often require developers to integrate external libraries for crucial security features, introducing complexity and potential vulnerabilities. This paper presents FastAPI, a high-performance web framework that revolutionizes API development by offering a potent combination of developer experience and built-in security features.

FastAPI empowers developers to prioritize security without sacrificing efficiency. Core features like OAuth2 authentication, JSON Web Tokens (JWTs), and dependency injection simplify the process of building secure APIs. Additionally, automatic documentation fosters transparency and facilitates secure API usage. By comparing FastAPI's security strengths to those of other popular frameworks, this paper will demonstrate its advantages in areas like user authentication, data validation, and fostering a secure development environment.

The proposed model demonstrates a FastAPI application implementing OAuth2 password grant flow for user authentication and authorization. It utilizes JWT (JSON Web Tokens) for secure access token generation and leverages dependency injection to manage user access control. The code defines an oauth2_scheme with different scopes ("read", "write", "admin") and an in-memory database to store tokens and user scopes. The verify_token function validates tokens and checks required scopes for access. The get_current_user dependency retrieves user information based on the provided token and ensures the user has the necessary scopes for the requested endpoint. Login functionality is implemented in the /token endpoint, issuing JWTs upon successful authentication. Protected endpoints like /admin/ and /secure-endpoint require specific scopes ("admin" and "read" respectively) for access, demonstrating authorization control through dependency injection. This model serves as a basic example of securing FastAPI applications with OAuth2, JWT, and dependency injection for robust authorization and authentication.

2. Security Features

FastAPI provides various security features to improve security in the API while building them for any application. There are several types of security features provided such as OAuth2 Authentication, JSON Web Tokens (JWT), Dependency Injection, Security Schemes and Security Scopes, etc.



Provided Security Features:

FastAPI prioritizes security by offering a comprehensive set of features to help you build robust and secure APIs. As shown in Figure 1, FastAPI provides OAuth2, Dependency Injection and Security schemas and scopes as inbuilt or provided security features.

A cornerstone of this security is its built-in support for OAuth2 authentication. OAuth2 is an industry-standard protocol that allows applications to obtain limited-time access to user accounts on external services like GitHub, Bitbucket, or Jira. By implementing OAuth2 with bearer tokens and passwords using the OAuth2PasswordBearer library, you can ensure that only authorized users can access your API endpoints. This is crucial for protecting sensitive data and functionalities within your application.

FastAPI's dependency injection system further strengthens your API's security posture. Dependency injection is a core concept in FastAPI that promotes code modularity and separation of concerns. This translates to security benefits by allowing you to control how dependencies like database sessions or authentication mechanisms are injected into your code. By managing these dependencies in a controlled and testable manner, you can ensure that sensitive operations are handled securely and can be easily audited if needed.

Beyond authentication and dependency injection, FastAPI empowers you to create fine-grained access control within your API using security schemes and scopes. Security schemes allow you to define different methods for authenticating users, such as API keys placed in headers, query parameters, or cookies, or traditional HTTP Basic authentication. This flexibility caters to various authentication needs. Additionally, security scopes let you specify the level of access granted by a token. You can define specific scopes for different functionalities within your API, ensuring that users only have access to the resources they are authorized for. This adherence to the principle of least privilege minimizes the potential damage caused by unauthorized access.

Supported Security Features:

FastAPI offers a range of features to enhance the security posture of your APIs. As shown in Figure 1, FastAPI provides JSON Web Tokens(JWT), CSRF Token and HTTP Support as supported security features.

While OAuth2 provides a robust foundation for user authentication, FastAPI can be seamlessly integrated with JSON Web Tokens (JWT) for stateless authentication. JWTs are compact tokens used for exchanging claims between parties and are often used in conjunction with OAuth2. You can leverage libraries like fastapi_jwt for verifying and decoding JWTs, ensuring secure access to protected resources within your API.

Beyond authentication, FastAPI allows you to implement Cross-Site Request Forgery (CSRF) protection using middleware or libraries like starlette-wtf. CSRF attacks exploit a user's authenticated session to perform unauthorized actions. By implementing CSRF protection, you can safeguard your API from such attacks.

FastAPI also integrates with HTTPS (HyperText Transfer Protocol Secure). HTTPS encrypts communication between the client and server, protecting data from eavesdropping and man-in-the-middle attacks. You can run your FastAPI application with HTTPS using standard ASGI servers like uvicorn. This ensures that sensitive data transmitted between users and your API remains confidential. Combined, these features empower you to build secure and reliable APIs.

3. COMPARATIVE ANALYSIS WITH OTHER FRAMEWORKS AND API TYPES

There are other frameworks used for developing APIs such as Django, Flask, Express.js, etc. API types such as Representational State Transfer (REST), and QueryLanguage like GraphQL. We have compared the above given other options with FastAPI as shown in Table 1 on the basis of security services to determine how FastAPI is better than them.

Feature	FastAPI	Flask	Django	REST (Architectural style)	Express.js (Node.js)	GraphQL
Built-in Authentication	OAuth2, JWT support (via libraries)	Limited (requires extensions)	Extensive (built-in & community packages)	N/A (architectural style)	Requires middleware/ packages	N/A (data query language)
Type Hinting	Yes	No	Yes (limited)	N/A (architectural style)	No	N/A (data query language)
Automatic Documentation	Yes (OpenAPI)	Limited (requires extensions)	Yes	N/A (architectural style)	Requires middleware/ packages	N/A (data query language)
Focus on Security	Yes	Requires more manual implementation	Yes	Requires secure implementation of underlying protocols	Yes	Focuses on data access control within schema.

Table 1: Comparison between different frameworks on basis of Security Features

4. PROPOSED MODEL

OAuth2 Implementation for Secure API Access with Scopes: In the proposed model, we implemented OAuth2 authentication to secure API endpoints, ensuring that only authorized users can access the specified resource. This implementation leverages FastAPI, a modern and high performing framework for building APIs.

System Architecture

The following figure shows the system architecture of the proposed model.

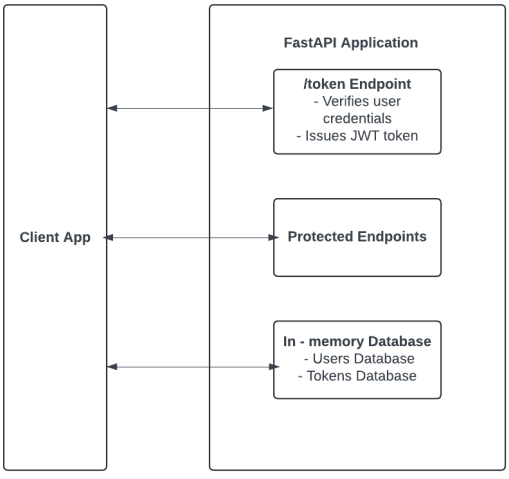


Figure 2: System architecture of the Proposed Model

OAuth2 Scheme and Token Management

We defined an OAuth2 password flow scheme with different scopes (read, write, admin) to control access to various endpoints. This configuration is achieved using FastAPI's OAuth2PasswordBearer, which specifies the token URL and the available scopes.

```
from fastapi.security import OAuth2PasswordBearer

oauth2_scheme = OAuth2PasswordBearer(
    tokenUrl="token",
    scopes={
        "read": "Read access",
        "write": "Write access",
        "admin": "Admin access",
    }
)
```

Token Validation and Scope Validation

A key aspect of the implementation is verifying tokens and validating the required scopes for each API request. Here, we use an in-memory database to store tokens and their associated user scopes for demonstration purposes.

```
def verify_token(token: str, required_scopes: List[str]):
    try:
        payload = jwt.decode(token, "secret_key", algorithms=["HS256"])
    except jwt.exceptions.DecodeError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid token",
            headers={"WWW-Authenticate": "Bearer"},
        )

    user_scopes = tokens_db[token]["scopes"]
    for scope in required_scopes:
        if scope not in user_scopes:
            raise HTTPException(
                status_code=status.HTTP_403_FORBIDDEN,
                detail="Not enough permissions",
                headers={"WWW-Authenticate": f'Bearer scope="{required_scopes}"'},
            )

    return tokens_db[token]
```

This function was implemented to ensure that only valid tokens with the necessary scopes can access protected resources.

Authentication and Authorization Flow

Authentication and Authorization process involves following steps:

1. **Requesting a Token:** Users are authenticated by sending their credentials to the '/token' endpoint. If the credentials are valid then they will receive a SHA256 encoded token.

```
curl -X POST "http://127.0.0.1:8000/token" -H "Content-Type: application/x-www-form-urlencoded" -d "username=user1&password=secret"
```

2. Accessing Protected Endpoints: To access the protected endpoints, users need to use their obtained token by including it in the Authorization header.

```
curl -X GET "http://127.0.0.1:8000/secure-endpoint" -H "Authorization: Bearer {token}"
```

Login Endpoint and Dependency Injection

Login endpoint was implemented so that it can simulate the user authentication and issue a token with appropriate scopes on basis of the user signing in and its allowed scopes.

```
@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    user = fake_users_db.get(form_data.username)
    if not user or user["password"] != form_data.password:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Incorrect username or password"
        )

    payload = {
        "username": form_data.username,
        "scopes": user["scopes"]
    }
    access_token = jwt.encode(payload, "secret_key", algorithm="HS256")
    tokens_db[access_token] = {"username": form_data.username, "scopes": user["scopes"]}

    return {"access_token": access_token, "token_type": "bearer"}
```

For dependency injection we used FastAPI's dependency injection to ensure that the users with valid tokens and required scopes can access certain endpoints.

```
async def get_current_user(security_scopes: SecurityScopes, token: str = Depends(oauth2_scheme)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": f'Bearer scope="{security_scopes.scope_str}"'},
    )

    user = verify_token(token, security_scopes.scopes)

    if user is None:
        raise credentials_exception

    return user
```

5. CONCLUSION

In conclusion, FastAPI provides several security features that enhance the security of APIs and protect against common vulnerabilities. Some of the key security features offered by FastAPI include OAuth2 authentication, JSON Web Tokens (JWT), dependency injection, security schemes and scopes, CSRF protection, and HTTPS support. By incorporating OAuth2 authentication, FastAPI ensures secure access to API endpoints through the use of bearer tokens and password verification. JWT integration allows for stateless authentication, enabling secure transfer of claims between parties. FastAPI's dependency injection system enables modularization and separation of concerns, enhancing security by managing sensitive operations and dependencies in a controlled and testable manner. FastAPI's support for security schemes and scopes allows for granular access control, enforcing authentication and authorization based on user roles and required scopes. While FastAPI does not provide built-in CSRF protection, it can be implemented using middleware or third-party libraries. When comparing FastAPI to other frameworks, such as Flask or Express.js, FastAPI stands out for its built-in support for OAuth2 and JWT authentication, type hinting, automatic documentation generation, and validation. FastAPI's focus on high performance and security make it a strong choice for developers building secure APIs.

The proposed model uses OAuth2 authentication along with JWT token and Security scope and schemas to improve the security provided by a FastAPI application. The proposed model combines all the security features and works so that the scopes of the users are decided and while login the model will generate the JWT token which consists all the information about access about the user and then using OAuth2 authentication scheme we will verify the token and after decoding the token user will only be able to access the endpoints which have allowed scopes i.e if the user does not have admin access then he won't be able to access endpoints requiring admin access. This model will increase security in the system and does not allow anonymous users to perform attacks that can compromise the system.

In summary, FastAPI provides a comprehensive set of security features that help developers build secure APIs. Its support for OAuth2, JWT, dependency injection, security schemes, and scopes, along with its emphasis on performance and type hinting, make it a powerful framework for developing secure and efficient applications.

REFERENCES

1. A. Chatterjee, M. W. Gerdes, P. Khatiwada and A. Prinz, "SFTSDH: Applying Spring Security Framework With TSD-Based OAuth2 to Protect Microservice Architecture APIs," in IEEE Access, vol. 10, pp. 41914-41934, 2022, doi: 10.1109/ACCESS.2022.3165548. keywords: {Security;Authentication;Protocols;Medical services;Privacy;Electronic healthcare;Prototypes;API Security;TSD;spring framework;HTTP;OAuth2;eCoach},
2. L. Ferretti, M. Marchetti and M. Colajanni, "Verifiable Delegated Authorization for User-Centric Architectures and an OAuth2 Implementation," 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, Italy, 2017, pp. 718-723, doi: 10.1109/COMPSAC.2017.260.keywords: {Authorization;Protocols;Cryptography;Data structures;Servers;Data models;integrity;oauth;identity;correctness;outsourcing;authorization;access control;cloud},
3. Sherwin John C.Tragura, Building Python Microservices with FastAPI: Build secure, scalable, and structured Python microservices from design concepts to infrastructure , Packt Publishing, 2022.
4. Chen, Junqiao. (2023). Model Algorithm Research based on Python Fast API. Frontiers in Science and Engineering. 3. 7-10. 10.54691/fse.v3i9.5591.