# Securing APIs with Spring Security

In this section, you built a complete, real-world authentication and authorization system using Spring Security and JWTs.

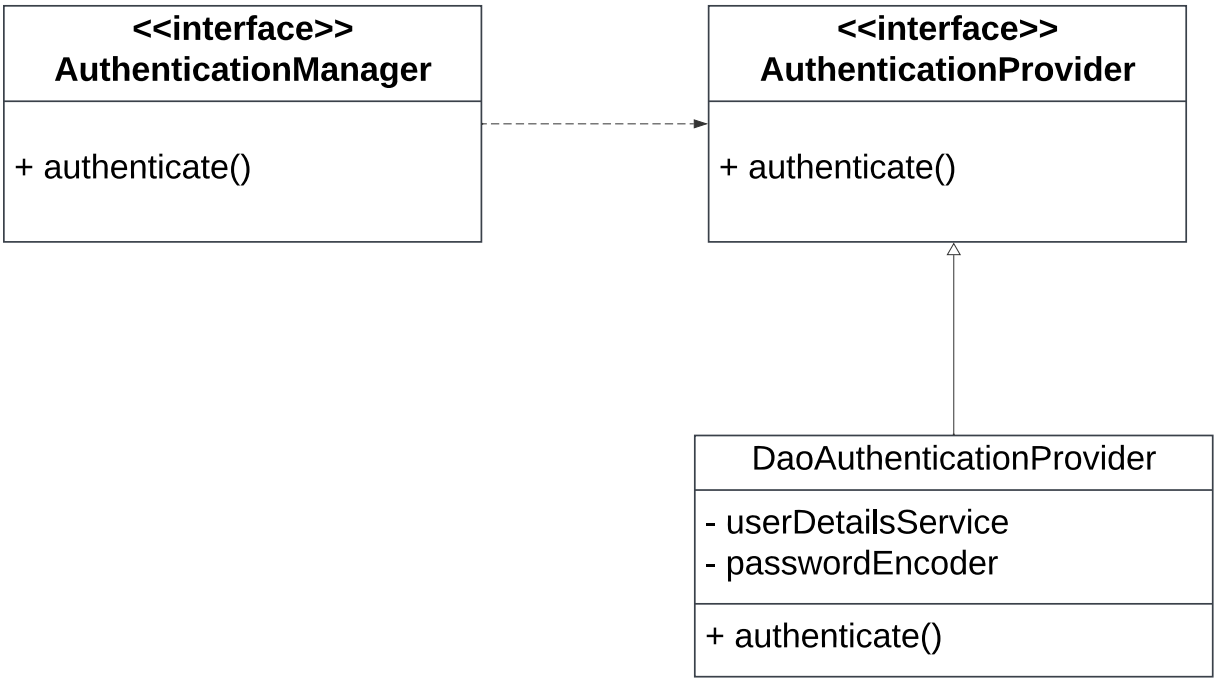
## Authentication Fundaments

We have two main authentication methods to choose from:

- **Session-based authentication** stores session data on the server—great for traditional web apps, but not ideal for APIs.
- **Token-based authentication** uses stateless JWTs, which are a better fit for REST APIs.

## User Login and Password Security

- Spring Security provides the `PasswordEncoder` interface for hashing passwords.
- To authenticate users, we work with Spring's built-in `AuthenticationManager`, which delegates to an `AuthenticationProvider`.



## JSON Web Tokens

- A JSON Web Token (JWT) is a compact, URL-safe string used to securely transmit information about a user between a client and a server.
- It has three parts, separated by dots (.): `<Header>.<Payload>.<Signature>`
  - **Header** – Specifies the signing algorithm and token type.
  - **Payload** – Contains the actual data (called claims) such as user ID, email, role, etc.
  - **Signature** – A cryptographic hash of the header and payload, signed with a secret key. This ensures the token hasn't been tampered with.
- In a typical token-based authentication flow, we generate two types of tokens:
  - **Access Token**
    - Used to access protected API endpoints
    - Sent by the client on every request to the server
    - Short-lived (usually 15 minutes or less)
    - If compromised, its short lifespan limits damage
    - Can be stored in memory (safer but cleared on page reload), or in localStorage (persistent but accessible to JavaScript).
  - **Refresh Token**
    - Used to get a new access token when the current one expires
    - Long-lived (typically 7 days or more)
    - Reduces the need for the user to log in repeatedly
    - Should be delivered as a secure HttpOnly cookie so it’s not accessible from JavaScript.