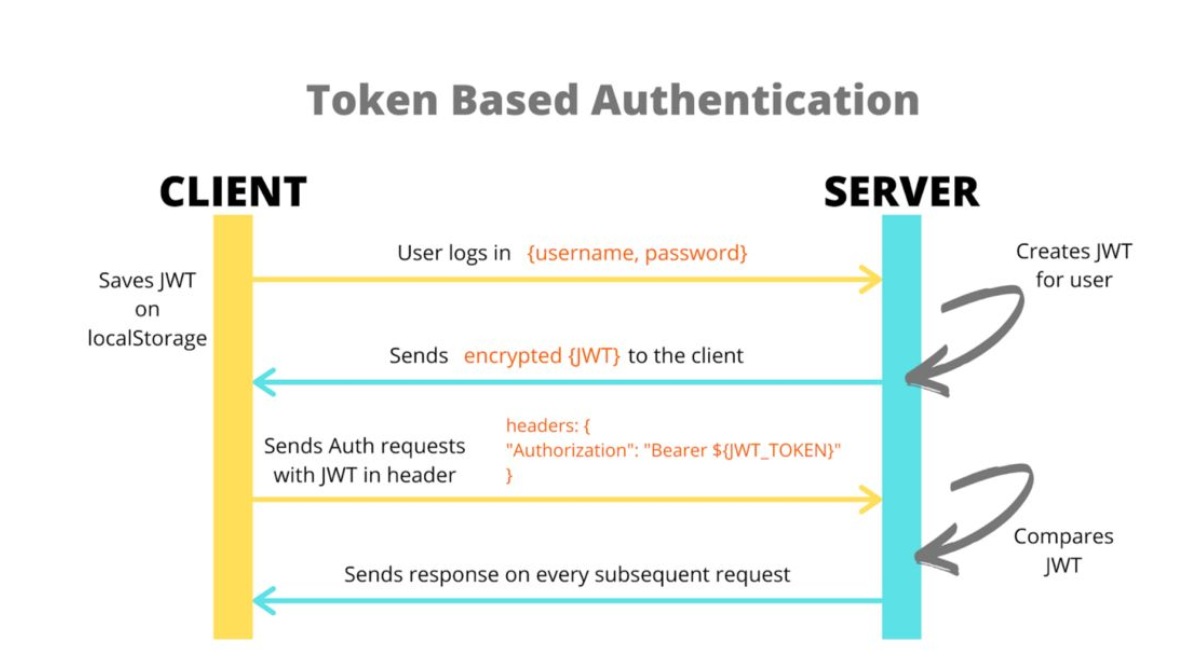
Authentication is the process of verifying a user's identity. Authentication is the first step in any security process.

Authentication – a client can make a request to with their authentication credentials (username + password) and when they are successfully authenticated the service will return a JWT.

The client will then store the JWT and each subsequent request will pass it via the Authorization header. When the Server application receives the request with the JWT it will verify that it is a valid token and if it is will allow the request to continue.

Authorization is the process of granting that user access to specific resources or functions.

Example - when you go through security in an airport, you show your ID to authenticate your identity. Then, when you arrive at the gate, you present your boarding pass to the flight attendant, so they can authorize you to board your flight and allow access to the plane.



**JSON WEB TOKENS (JWT)**

A JSON web token(JWT) is JSON Object which is used to securely transfer information over the web(between two parties). It can be used for an authentication system and can also be used for information exchange. The token is mainly composed of header, payload, signature. These three parts are separated by dots(.). JWT defines the structure of information we are sending from one party to the another, and it comes in two forms – **Serialized, Deserialized**. The Serialized approach is mainly used to transfer the data through the network with each request and response. While the deserialized approach is used to read and write data to the web token.

**Deserialized**

JWT in the deserialized form contains only the header and the payload. Both of them are plain JSON objects.

**Serialized**

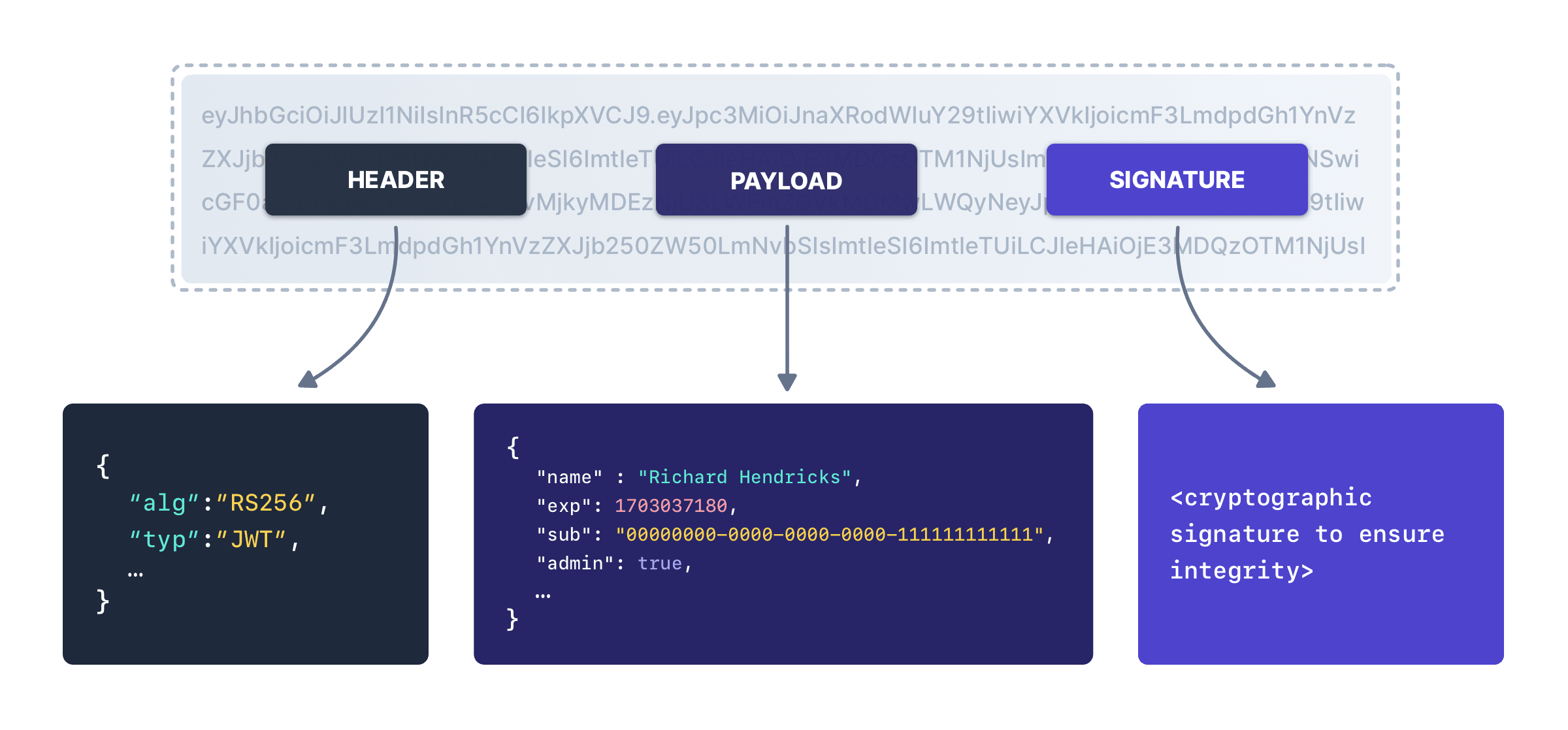
JWT in the serialized form represents a string of the following format:

[header].[payload].[signature]

all these three components make up the serialized JWT.

A JSON Web Token is an open method for representing claims securely between two parties. A JWT is a set of claims (JSON property–value pairs) that together make up a JSON object. It consists of three parts:

* Header: Consists of two properties: { "alg": "HS256", "typ": "JWT" }. alg is the algorithm that is used to encrypt the JWT.
* Payload: This is where the data to be sent is stored; this data is stored as JSON property–value pairs.
* Signature: This is created by encrypting, with the algorithm specified in the header: (i) the base64Url-encoded header, (ii) base64Url-encoded payload, and (iii) a secret (or a private key):



This is the third part of JWT and used to verify the authenticity of token. BASE64URL encoded header and payload are joined together with dot(.) and it is then hashed using the hashing algorithm defined in a header with a secret key. This signature is then appended to header and payload using dot(.) which forms our actual token header.payload.signature

**Syntax :**

*HASHINGALGO( base64UrlEncode(header) + “.” + base64UrlEncode(payload),secret)*

in your configuration.

​

#### SIGNING JSON WEB TOKENS

The next step is to create a new JwtDecoder bean but I think we need to talk about what we are going to do here. As you learned earlier there are 3 parts to the JWT, the header, payload, and signature. The signature is created using by encrypting the header + payload and a secret (or private key).

A JWT can be encrypted using either a symmetric key (shared secret) or asymmetric keys (the private key of a private-public pair).

Symmetric key: The same key is used for both encryption (when the JWT is created) and decryption (MobileTogether Server uses the key to verify the JWT). The symmetric key—also known as the shared secret—is stored as a setting in MobileTogether Server. See Symmetric Key: Shared Secret for details of working with symmetric keys.

Asymmetric keys: Different keys are used for encryption (private key) and decryption (public key). The public key is stored as a setting in MobileTogether Server so that the JWT can be verified. For information about using asymmetric encryption for JWTs, see Asymmetric Keys: Public Key.

Steps to create your security project.

1. Create User class
2. Create UserRepostiory that extends JPA Repository
3. Create Login Controller and mapping for /welcome
4. Now run your application into postman
5. It is secured and not allowing you to get in. You will get 401 unauthorized error.
6. Create a Securityconfig class to define security configuration
7. Annotate class with @Configuration and @EnableWebSecurity.
8. Define security.jwt.secret-key and security.jwt.issuer in application.properties file
9. Use security key in security config file to get the key value for decoding JWT token.
10. Now run your application and you will able to access your welcome page
11. Now we need to configure the authentication manager to read the users from the database.
12. So, Create a service class that implements UserServiceDetails and write your findByUserName and findByEmail methods.
13. Now create Authentication manager to use this service class . So register a bean there.
14. This authentication manager will use this JWT decoder to decode the JWT token.
15. Now a RestController and write a method to create JWT token.
16. Now create a SignUp model class to get the signUp request from client.
17. Now create a signUp method annotate with PostMapping to allow for signup for the user.
18. This method uses a BindingResult object as a parameter to check the submit data is valid or not.
19. If we have a validation error this method will return bad request responseentity otherwise will create a signupuser.
20. We will also check the database if that user or email is already present or not, if present then return bad request other wise will save that user into the database and will create a jwttoken with token and userdetails and return the response. If we have exception then will return a bad response to the user.
21. The go to postman and hit the http endpoint for the signup and sent the requestbody of signup.
22. This will create a user into the database and will return a JWT Token.
23. Steps to authenticate a user.
24. Now create a login model class that should have NotEmpty fields username and password.
25. Now we need authentication manager to authenticate user, that we already created in our SpringConfig file.
26. And create Login method with PostMapping of SignInRequest Body and BindingResult objec , check user validation add JWTtoken and return the result.
27. After login take your JWT token and access other api .