Spring boot Security

Rajeev Gupta
rgupta.mtech@gmail.com
Java Trainer & consultant
@rajeev_gupta76

Rajeev Gupta

FreeLancer Corporate Java JEE/ Spring Trainer

New Delhi Area, India

freelance

Institution of Electronics and Telecommunication...

See contact info

See connections (500+)

Add profile section | More...

1. Expert trainer for Java 8, GOF Design patterns, OOAD, JEE 7,Spring 5, Hibernate 5 ,Spring boot, microservice, netflix oss, Spring cloud, angularjs, Spring MVC, Spring Data, Spring Security, EJB 3, JPA 2, JMS 2, Struts 1/2, Web service

2. Helping technology organizations by training their fresh and senior engineers in key technologies and processes.

3. Taught graduate and post graduate academic courses to students of professional degrees.

I am open for advance java training /consultancy/ content development/ guest lectures/online training for corporate / institutions on freelance basis

Contact :

==========

raupta.mtech@gmail.com

Clients:

==========

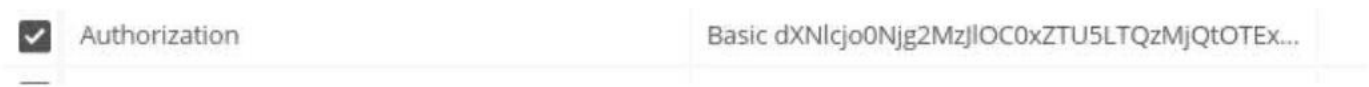
Gemalto, Noida
Cyient Ltd, Noida
Fidelity Investment Ltd
Blackrock, Gurgaon
Mahindra comviva
Steria
Bank Of America
incedo gurgaon
MakeMyTrip
Capgemini India
HCL Technologies
CenturyLink
Deloitte consulting
Nucleus Software
Ericsson Gurgaon
Avaya gurgaon
Kronos Noida
NEC Technologies
A.T. Kearney
ust global
TCS
North Shore Technologies Noida
IBM
Sapient
Accenture
Incedo
Genpact
Indian Air force
Indian railways
Vidya Knowledge Park

Spring REST security

Step 1: configuration spring rest security

- Step 1: just add spring security dependencies to boot project, an default user/password is generated

```
Using generated security password: 468632e8-1e59-4324-911f-b493308f6408
```



- Step 2: configre username/password

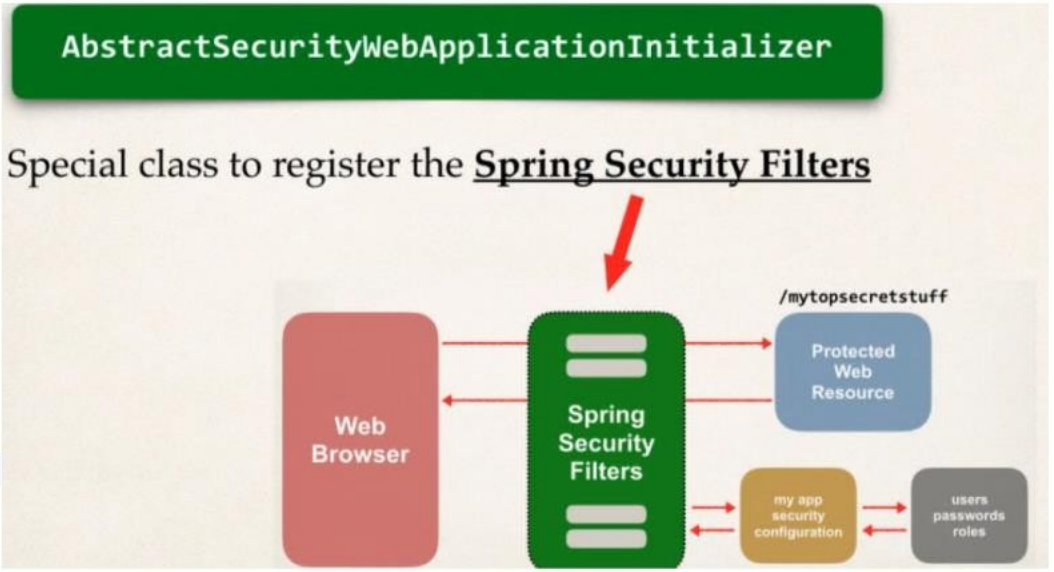
```
spring.security.user.name=admin
spring.security.user.password=admin
```

- Configuration of DelegatingFilterProxy:

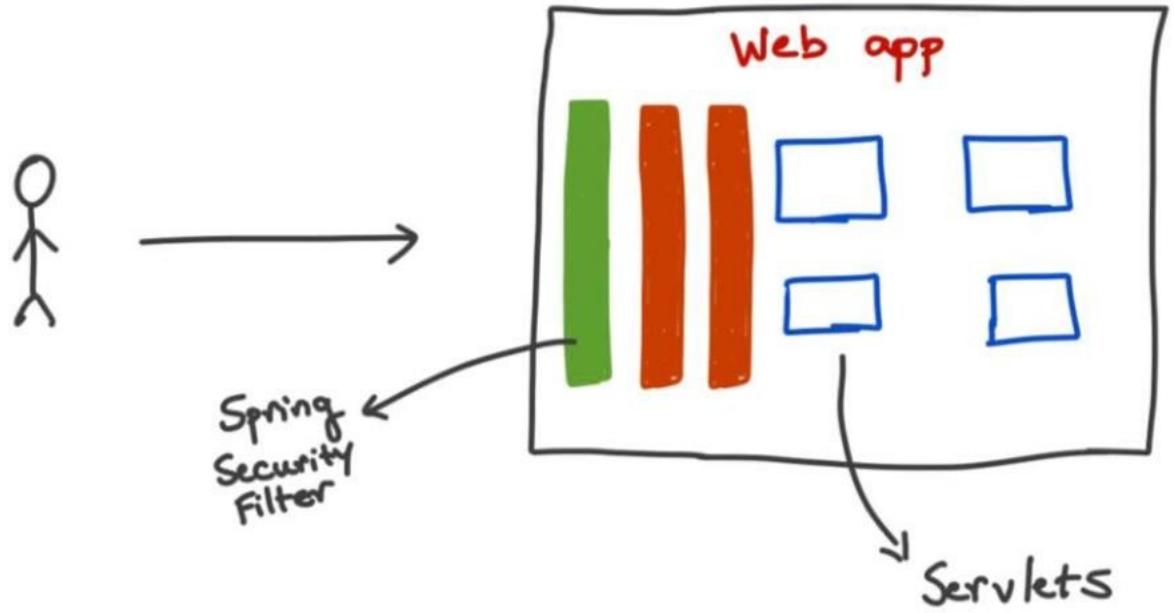
  No need as spring boot do behind

  the scenes

```
public class WebSecConfigInitilizer extends
        AbstractSecurityWebApplicationInitializer{

}
```

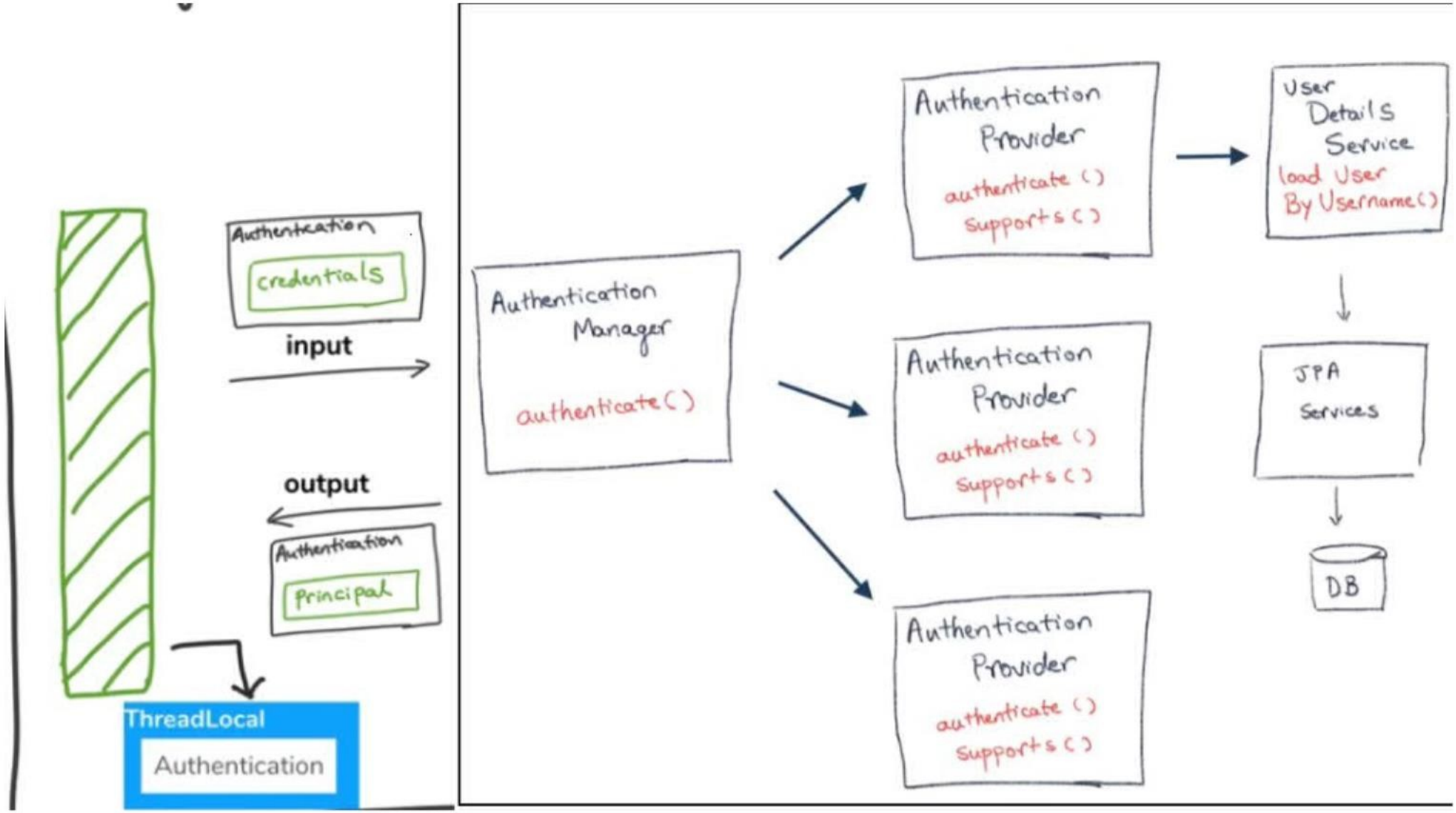


How Spring security works

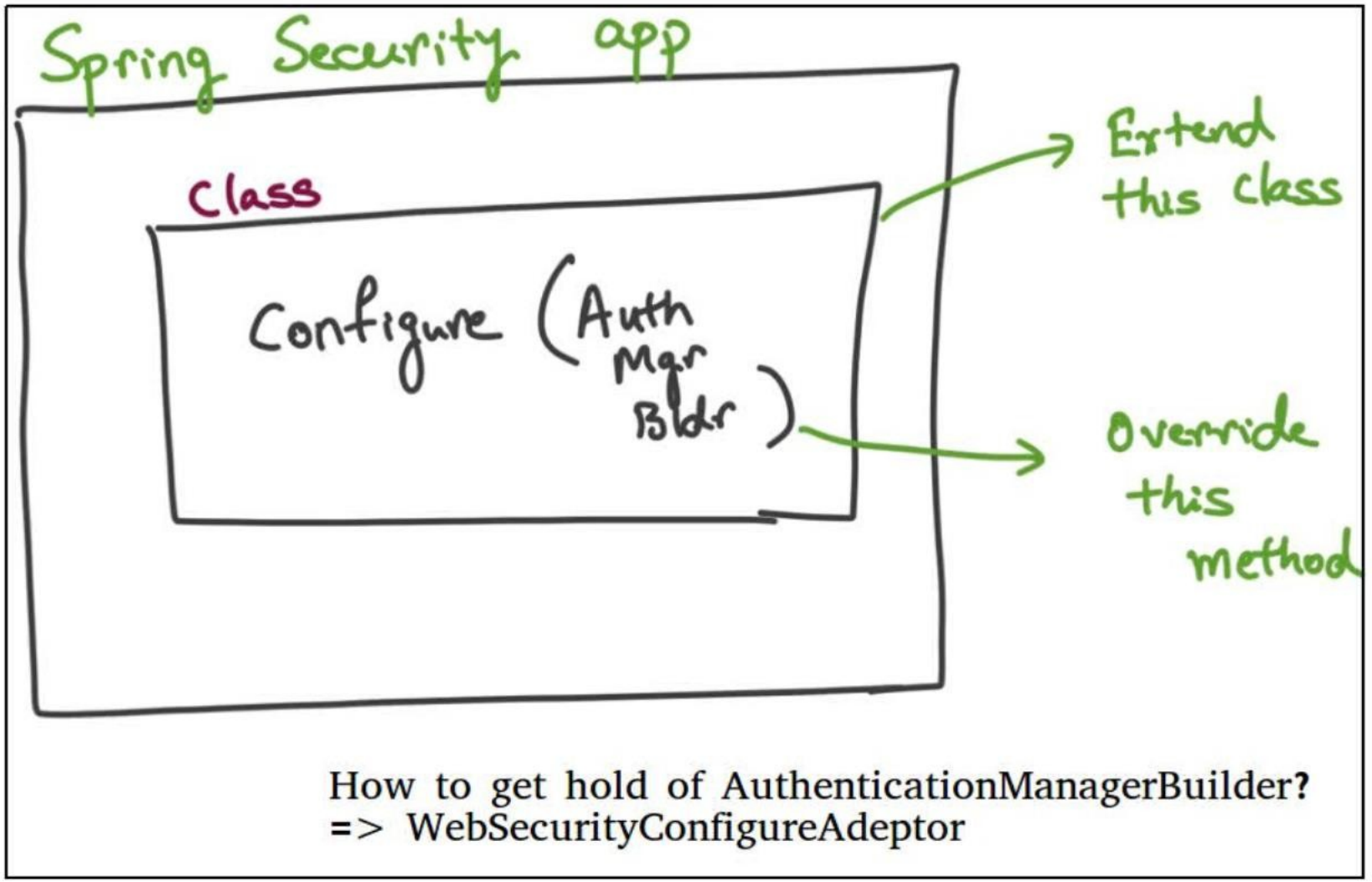


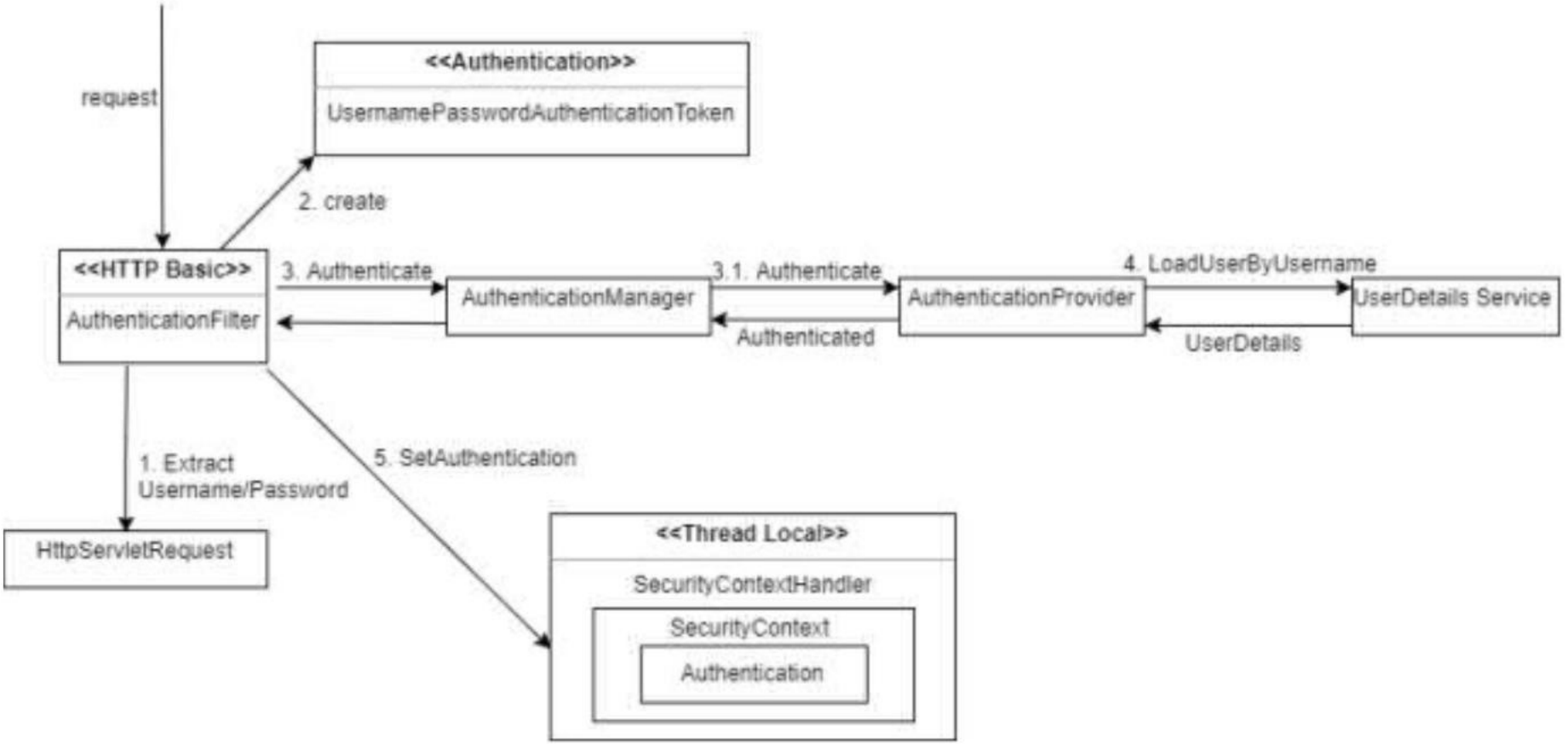

```
<filter>
<filter-name>springSecurityFilterChain</filter-name>
<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
<filter-name>springSecurityFilterChain</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

How Spring security works



How Spring security works





1 > Authentication filter creates an “Authentication Request” and passes it to Authentication manager

2 > Authentication Manager Delegates to one or more Authentication Provider

3 > Authentication Provider Uses UserDetailsService to load the UserDetails and return an “Authenticated Principle”

4 > Authentication filter sets the authentication in the security context

Configuration spring security

- Defining endpoints, for each profile

```
@RestController
public class HomeResource {
    @GetMapping("/")
    public String home() {
        return ("<h1>Welcome</h1>");
    }
    @GetMapping("/emp")
    public String homeEmp() {
        return ("<h1>Welcome emp</h1>");
    }

    @GetMapping("/mgr")
    public String homeMgr() {
        return ("<h1>Welcome mgr</h1>");
    }

    @GetMapping("/admin")
    public String homeAdmin() {
        return ("<h1>Welcome admin</h1>");
    }
}
```

Configuration spring security

```
@EnableWebSecurity
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private SecurityEntryPointHandler authEntryPoint;
    @Override
    protected void configure(AuthenticationManagerBuilder auth)throws Exception {
        auth.inMemoryAuthentication()
        .withUser("raj").password("raj121").roles("ADMIN")
        .and()
        .withUser("ekta").password("ekta121").roles("MGR")
        .and()
        .withUser("gunika").password("gunika121").roles("EMP");
    }
    @Bean
    public PasswordEncoder getPasswordEncoder(){
        return NoOpPasswordEncoder.getInstance();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
        .authorizeRequests()
        .antMatchers("/admin/**").hasAnyRole("ADMIN")
        .antMatchers("/mgr/**").hasAnyRole("ADMIN","MGR")
        .antMatchers("/emp/**").hasAnyRole("ADMIN","MGR","EMP")
        .antMatchers("/home/**").permitAll()
        .and().httpBasic().authenticationEntryPoint(authEntryPoint);
    }
}
```

```
        .antMatchers("/clerk/**").hasAnyRole("MGR","ADMIN","CLERK")
        .antMatchers("/home/**").permitAll()
        .and().httpBasic().and().sessionManagement().sessionCreationPolicy
        (SessionCreationPolicy.STATELESS);
    }
```

Configuration authenticationEntryPoint

```java
@Component
public class SecurityEntryPointHandler extends BasicAuthenticationEntryPoint{

    public void afterPropertiesSet() throws Exception {
        setRealmName("book-app");
        super.afterPropertiesSet();
    }

    public void commence(HttpServletRequest request, HttpServletResponse response,
            AuthenticationException authException) throws IOException, ServletException {
        response.addHeader("WWW-Authenticate", "Basic realm=\"" + getRealmName() + "\"");
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        PrintWriter out=response.getWriter();
        out.print("401-UNAUTHORIZED");
    }
}
```

401 Unauthorized:

> If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.

403 Forbidden:

> The server understood the request, but is refusing to fulfill it.

Configuration security Hibernate config

```
@Entity
@Table(name="user_table")
public class User {
    public static final BCryptPasswordEncoder encode=
            new BCryptPasswordEncoder();
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String username;
    private String password;
    private String email;
    private boolean status;

    private String []roles;
    public User(String username, String password, String email, boolean status,
            String[] roles) {
        this.username = username;
        setPassword(password);
        this.email = email;
        this.status = status;
        this.roles = roles;
    }

    public void setPassword(String password) {
        this.password = encode.encode(password);
    }
```

```
@Repository
public interface UserRepository extends CrudRepository<User, Integer>{
    public List<User> findByUsername(String username);
}
```

Configuration security Hibernate config

```java
public interface UserService {
    public List<User> findByUsername(String username);
    public void addUser(User user);
}
```

```java
@Service
@Transactional
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;
    @Override
    public List<User> findByUsername(String username) {
          return userRepository.findByUsername(username);
    }
    @Override
    public void addUser(User user) {
        userRepository.save(user);
    }

}
```

Configuration userDetail service

```
@Service
@Transactional
public class DetailService implements UserDetailsService {
    @Autowired
    private UserService userService;
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        User user = null;
        List<User> users = userService.findByUsername(username);
        if (users.size() == 0) {
            throw new UsernameNotFoundException("username is not found");
        } else {
            user = users.get(0);
            return new org.springframework.security.core.userdetails.User
                    (user.getUsername(),
                    user.getPassword(),
                    AuthorityUtils.createAuthorityList(user.getRoles()));
        }

    }
}
```

Configuration security Hibernate config

```
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled=true, securedEnabled=true)
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService detailService;
    @Autowired
    private SecurityEntryPointHandler authEntryPoint;
    @Override
    protected void configure(AuthenticationManagerBuilder auth)throws Exception {
        auth.userDetailsService(detailService);
    }
    @Bean
    public BCryptPasswordEncoder getBCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable().authorizeRequests().antMatchers("/admin/**")
                .hasAnyRole("ADMIN").antMatchers("/mgr/**")
                .hasAnyRole("ADMIN", "MGR").antMatchers("/emp/**")
                .hasAnyRole("ADMIN", "MGR", "EMP").antMatchers("/home/**")
                .permitAll().and().httpBasic()
                .authenticationEntryPoint(authEntryPoint);
    }
}
```

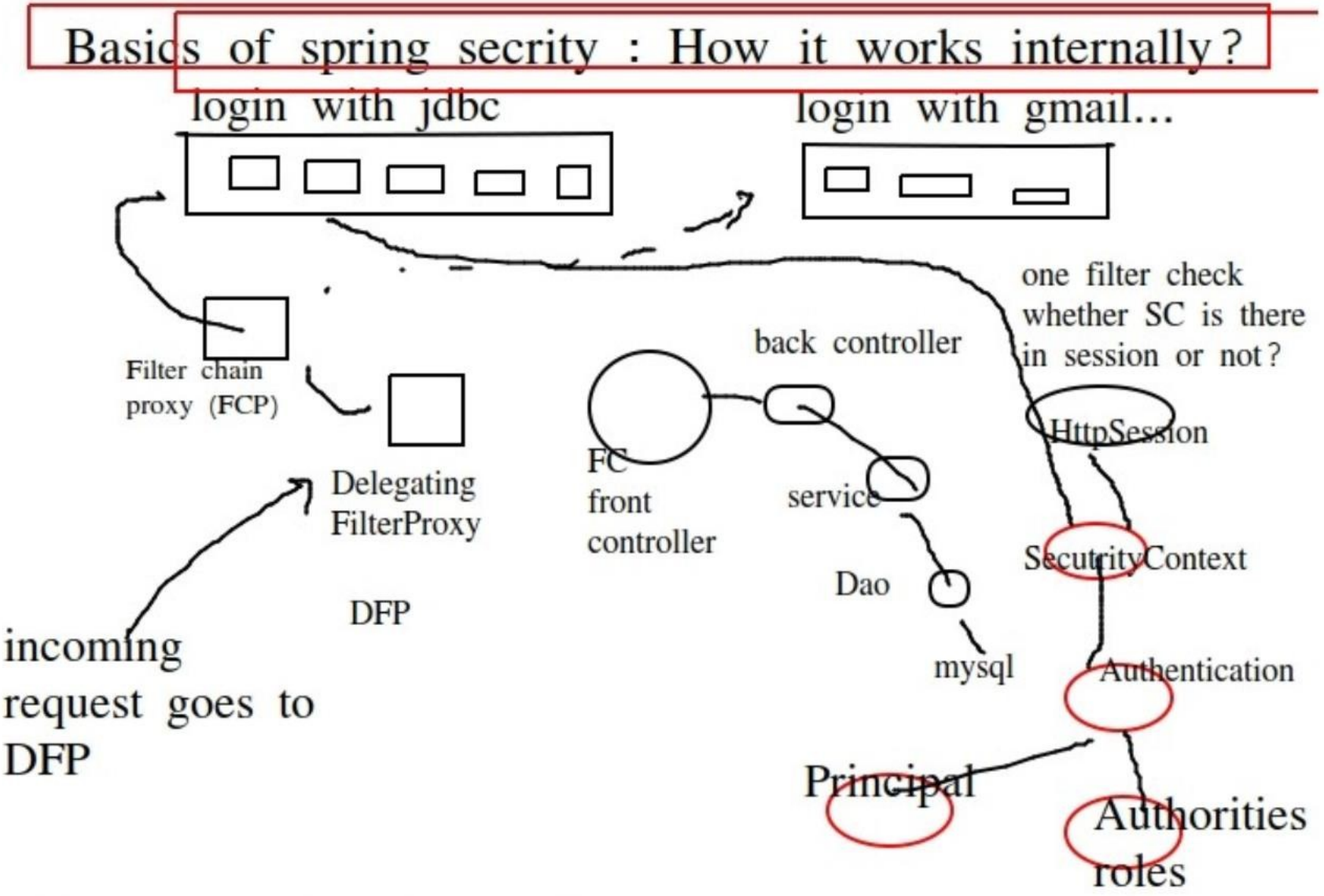
Configuration security Hibernate config-Alternative

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
    .authorizeRequests()
    .antMatchers("/admin/**").hasAnyAuthority("ROLE_ADMIN")
    .antMatchers("/mgr/**").hasAnyAuthority("ROLE_ADMIN","ROLE_MGR")
    .antMatchers("/clerk/**").hasAnyAuthority("ROLE_ADMIN","ROLE_MGR","ROLE_CLERK")
    .antMatchers("/home/**").permitAll()
    .and().httpBasic().and().sessionManagement()
    .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
}
```

Configuration security Hibernate config

```
@SpringBootApplication
public class SecDemoApplication implements CommandLineRunner{

    @Autowired
    private UserService userService;
    public static void main(String[] args) {
        SpringApplication.run(SecDemoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println("i am going to run just after spring boot project initiated...");

        User user1=new User("raj", "raj121", "raj.mtech@gmail.com", true, new String[]
                {"ROLE_EMP", "ROLE_MGR","ROLE_ADMIN"});
        User user2=new User("ekta", "ekta121", "eku.mtech@gmail.com", true, new String[]{"ROLE_MGR","ROLE_EMP"});
        User user3=new User("gunika", "gunika121", "raj.mtech@gmail.com", true, new String[]{"ROLE_EMP"});

        System.out.println("inserting an default user");
        userService.addUser(user1);
        userService.addUser(user2);
        userService.addUser(user3);
    }

}
```

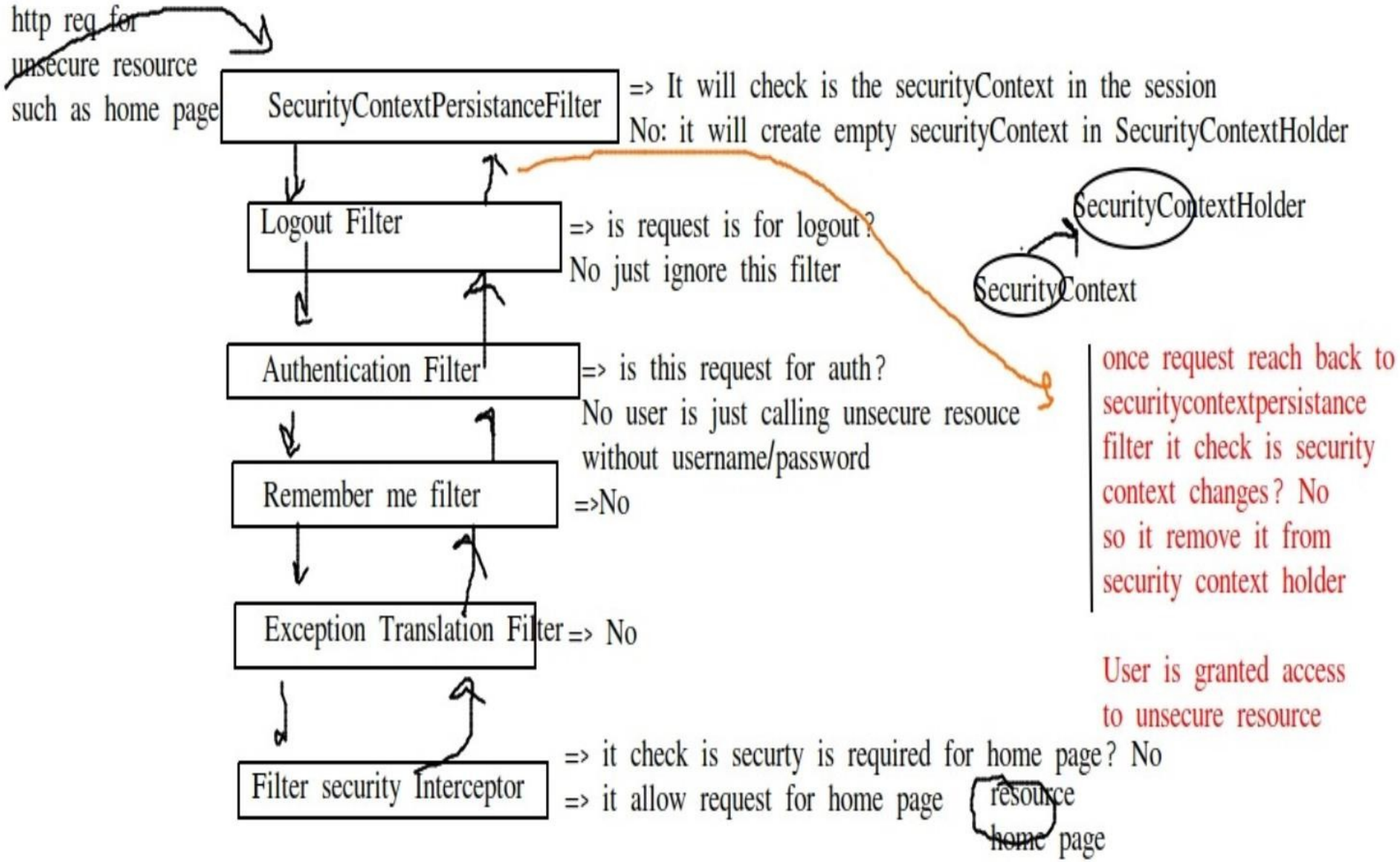
Spring security under the hood



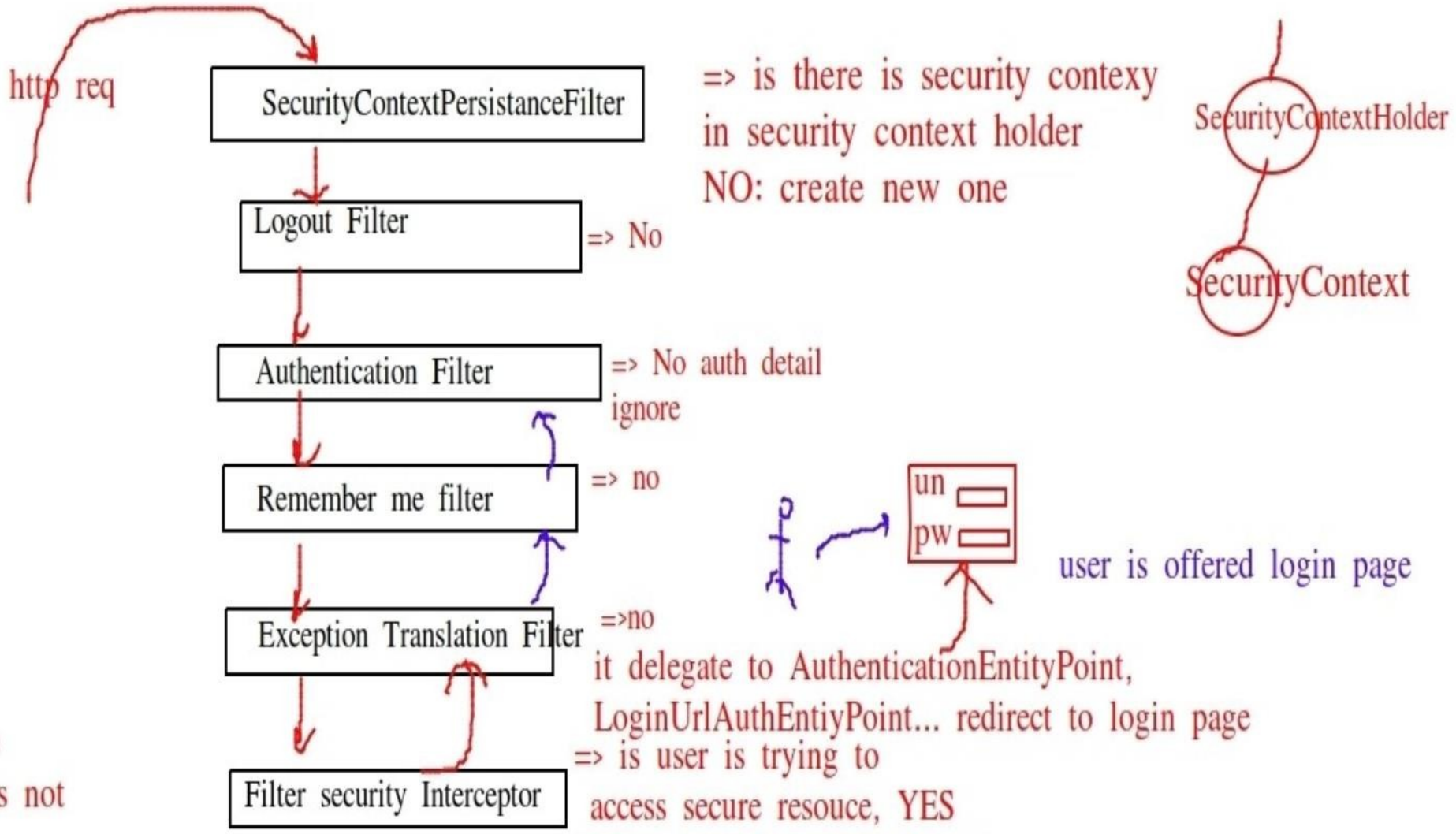
Spring security is based on servlet filter

Rather then configure many filter we need to configre fiter chain

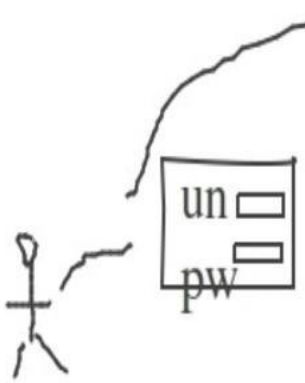
Case 1: user is trying to hit an unsecure resource priore to login



Case :2 user is trying to hit an Secure resource priore to login



Case 3: user submitting login credentials



SecurityContextPersistanceFilter

it create empty securitycontext in SCH

Logout Filter

is request for logout ?No Ignore

it will populate SC with

Authentication Filter

is req for auth ?YES

Auth Filter delegate to AuthenticationManager

Auth Mgr create Authentication object that contain details about current user

Remember me filter

Exception Translation Filter

Now filter secuity interceptor need to decide whether to allow user to access admin resources or not?

Filter security Interceptor

AcessDecisionManager to allow access or not?

SecurityContext is also kept in thread local so that we can access it anywhere in application, even in service layer

it is also kept in httpsession

TL

SecContextHolder

SecuriyContext

Authentication

Principal

Authorite

AuthenticationProvider

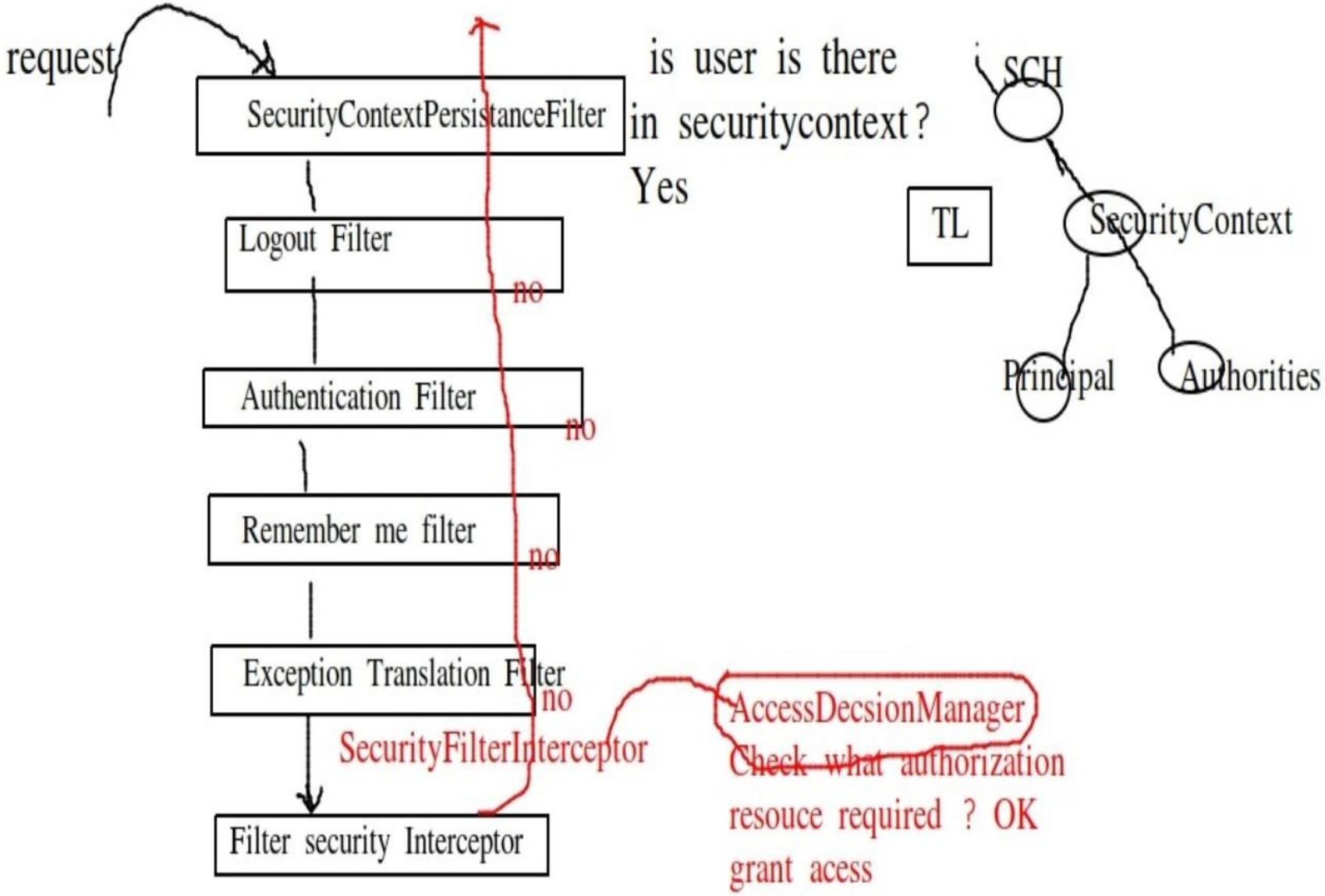
validate user

UserDetails

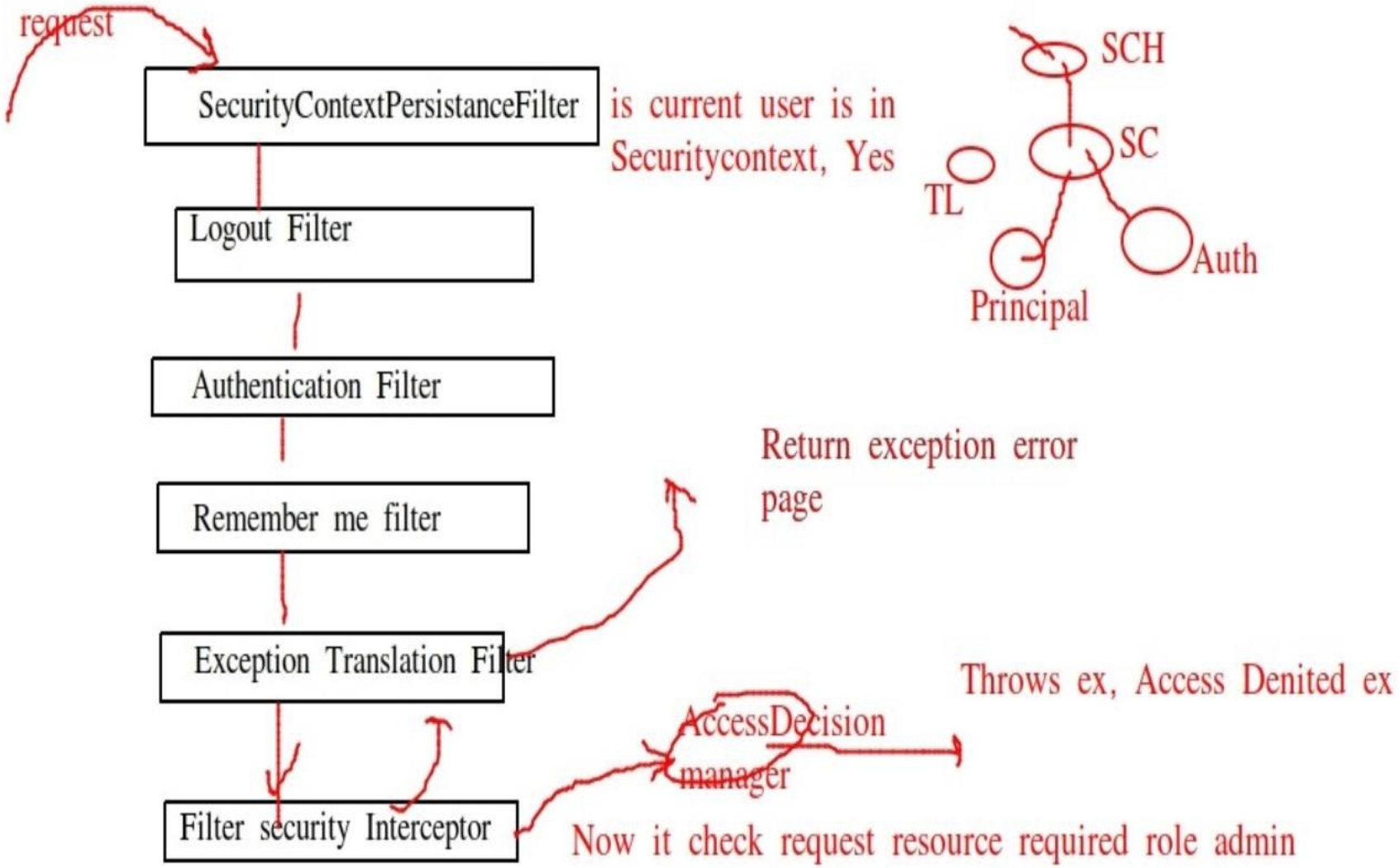
can get from jdbc , hib etc inmomory, ldap etc

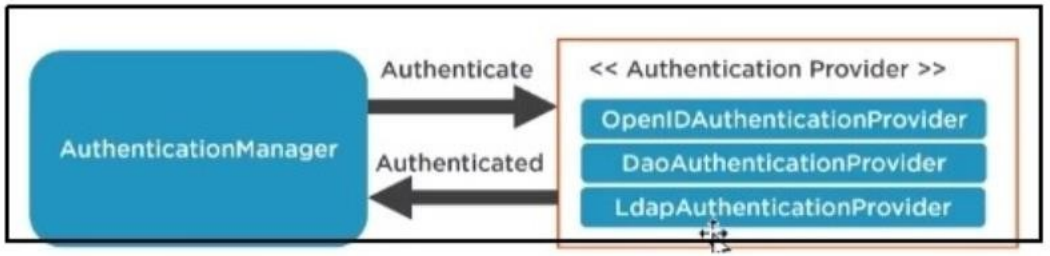
Why SC is kept in Thread local: we can git secuirity context anywhere SecurityContextHolder.getSecityContext() from thread local

Case 4: user is trying to hit secured resouces after login with requested role



Case :5 User trying to hit secured resouces after login withoout equested role






```java
public interface Authentication extends Principal, Serializable {
    // ~ Methods
    // ========================================================================================

     * Set by an <code>AuthenticationManager</code> to indicate the authorities that the
    Collection<? extends GrantedAuthority> getAuthorities();

     * The credentials that prove the principal is correct. This is usually a password,
    Object getCredentials();

     * Stores additional details about the authentication request. These might be an IP
    Object getDetails();

     * The identity of the principal being authenticated. In the case of an authentication
    Object getPrincipal();

     * Used to indicate to {@code AbstractSecurityInterceptor} whether it should present
    boolean isAuthenticated();

     * See {@link #isAuthenticated()} for a full description.
    void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException;
}
```


```java
public interface AuthenticationManager {
    // ~ Methods
    // ========================================================================================

     * Attempts to authenticate the passed {@link Authentication} object, returning a
    Authentication authenticate(Authentication authentication)
            throws AuthenticationException;
```

Spring JSP security

Step 20: Spring security with JSP

- No changes in hibernate configuration just need to change configure method

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
    .authorizeRequests()
    .antMatchers("/admin/**").hasAnyRole("ADMIN")
    .antMatchers("/mgr/**").hasAnyRole("ADMIN","MGR")
    .antMatchers("/clerk/**").hasAnyRole("ADMIN","MGR","CLERK")
    .antMatchers("/home/**").authenticated()
    .and().formLogin()
    .loginPage("/login").loginProcessingUrl("/myloginprocessor")
    .usernameParameter("username").passwordParameter("password")
    .defaultSuccessUrl("/home")
    .permitAll()
    .and()
    .httpBasic()
    .and()
    .exceptionHandling().accessDeniedPage("/accessdenied");

}
```

Custom login page

```jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<body>

    <c:if test="${not empty errorMessge}">
        <div style="color:red">${errorMessge}</div>
    </c:if>

    <form name='login' action="login" method='POST'>
        <table>
            <tr>
                <td>UserName:</td>
                <td><input type='text' name='name' value=''></td>
            </tr>
            <tr>
                <td>Password:</td>
                <td><input type='password' name='pass' /></td>
            </tr>
            <tr>
                <td colspan='2'><input name="submit" type="submit" value="submit" /></td>
            </tr>
        </table>
       <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
    </form>
</body>
</html>
```

Custom login get mapping

```
@RequestMapping(value = "/login", method = RequestMethod.GET)
public String loginPage(@RequestParam(value = "error", required = false) String error,
                        @RequestParam(value = "logout", required = false) String logout,
                        Model model) {
    String errorMessge = null;
    if(error != null) {
        errorMessge = "Username or Password is incorrect !!";
    }
    if(logout != null) {
        errorMessge = "You have been successfully logged out !!";
    }
    model.addAttribute("errorMessge", errorMessge);
    return "login";
}
```

Using @AuthenticationPrincipal

- Using @AuthenticationPrincipal to get user details in an controller, it will only works if SecUser must have implementated UserDetails interface

```
@GetMapping(path="home")
public String home(@AuthenticationPrincipal SecUser user){
    System.out.println(user);
    return "home";
}
```

```
@GetMapping(path="clerk")
public String helloClerk(@AuthenticationPrincipal SecUser user){
    System.out.println(user);
    return "clerk_home";
}
```

```
public class SecUser implements UserDetails{
    private static final long serialVersionUID = 2469986169861297608L;
    private User user;
    public SecUser(User user) {
        this.user = user;
    }
    public SecUser() {}
```

Concurrent Session management

- Let's say you charge a monthly fee per user for your software application
- What happens if one user purchases a license, then shares that account with all their friends?
- This is where concurrent session management can help you out!
- concurrent session management allows you to forcibly logout any duplicate users

Ex:

1. User A logs in via Computer A
2. User A then logs in via Computer B
3. User A's session on Computer A is invalidated and they are logged out
4. User A on Computer B doesn't notice anything

Concurrent Session management

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
    .authorizeRequests()
    .antMatchers("/admin/**").hasAnyRole("ADMIN")
    .antMatchers("/mgr/**").hasAnyRole("ADMIN","MGR")
    .antMatchers("/clerk/**").hasAnyRole("ADMIN","MGR","CLERK")
    .antMatchers("/home/**").authenticated()
    .and().formLogin()
    .loginPage("/login").loginProcessingUrl("/myloginprocessor")
    .usernameParameter("username").passwordParameter("password")
    .defaultSuccessUrl("/home")
    .permitAll()
    .and()
    .httpBasic()
    .and()
    .exceptionHandling().accessDeniedPage("/accessdenied").and()
    .sessionManagement().maximumSessions(1);

}
```

```
This session has been expired (possibly due to multiple concurrent logins being attempted as the same user).
```

Note: SecUser must implements equal() and hashcode() methods

Spring security Display content based on roles

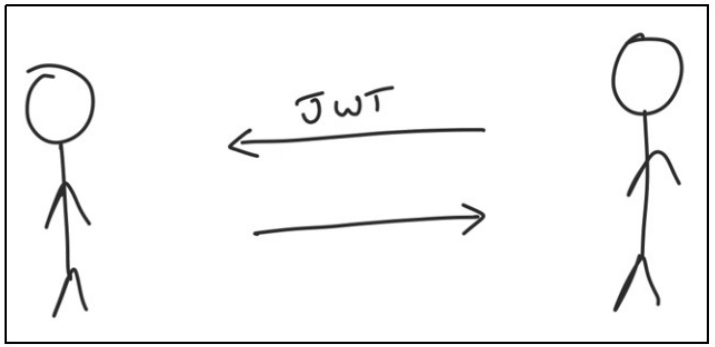
```
< %@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
< %@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
< %@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
< %@ taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
```

```
<sec:authorize access="hasAnyRole('MGR','ADMIN')">
<a href="mgr">MGR</a><br/>
</sec:authorize>

<sec:authorize access="hasRole('ADMIN')">
<a href="admin">admin</a><br/>
</sec:authorize>
```

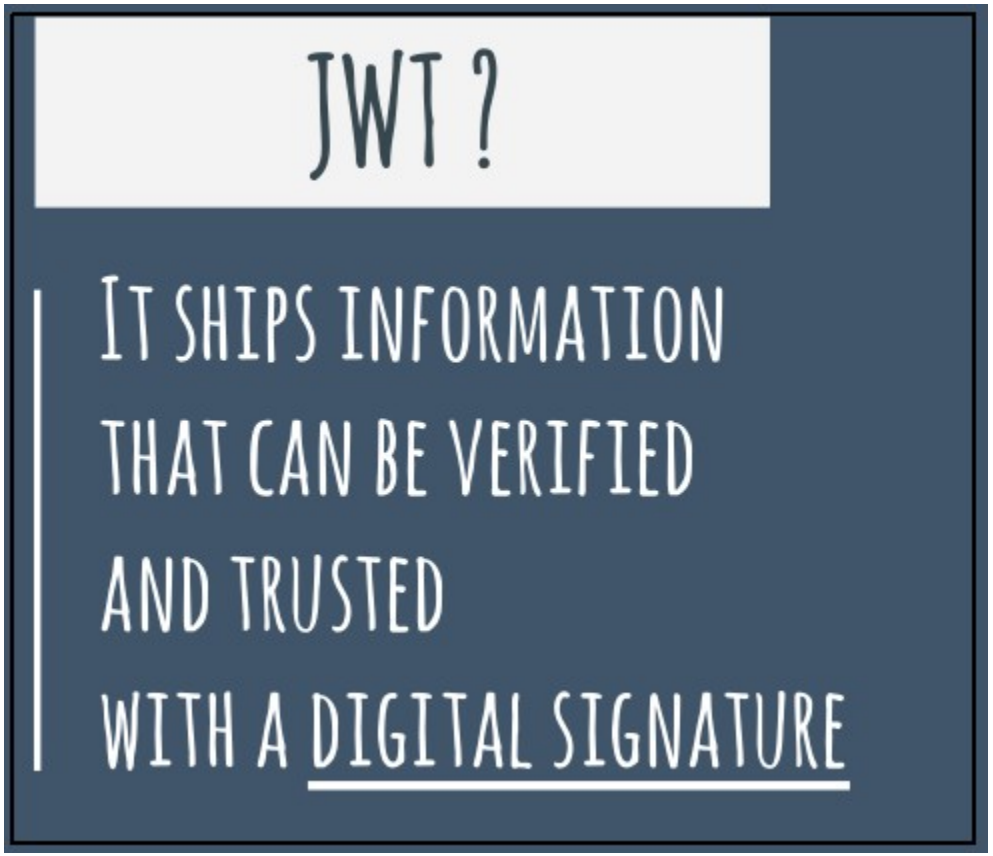
```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
</dependency>
```

JWT based security

rgupta.mtech@gmail.com



What is JWT?


JWT ?
IT SHIPS INFORMATION
THAT CAN BE VERIFIED
AND TRUSTED
WITH A DIGITAL SIGNATURE

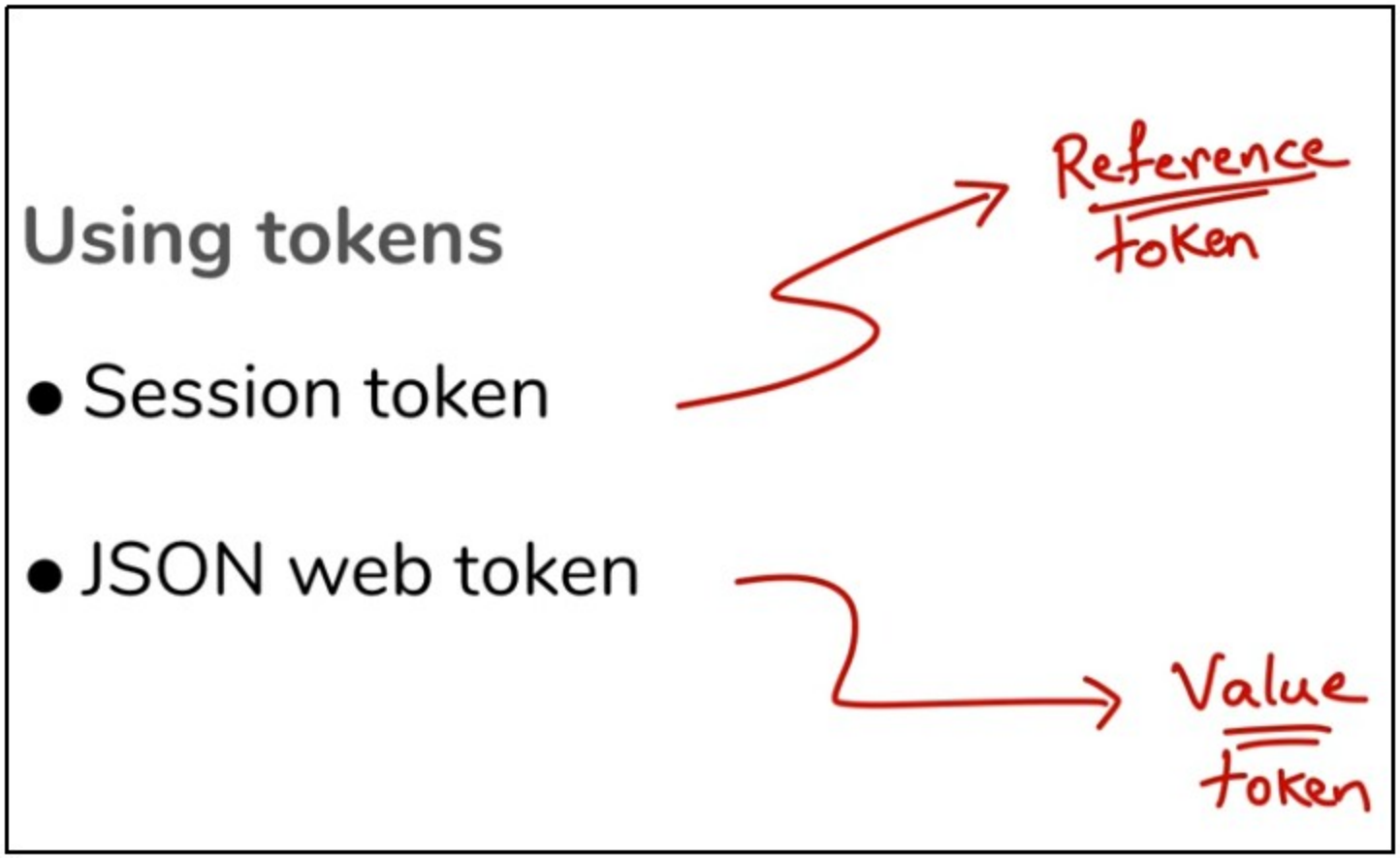
Why JWT?

Stateless

JWT allow the server to verify the information contained in the JWT without necessarily storing state on the server.

http://secure.indas.on.ca

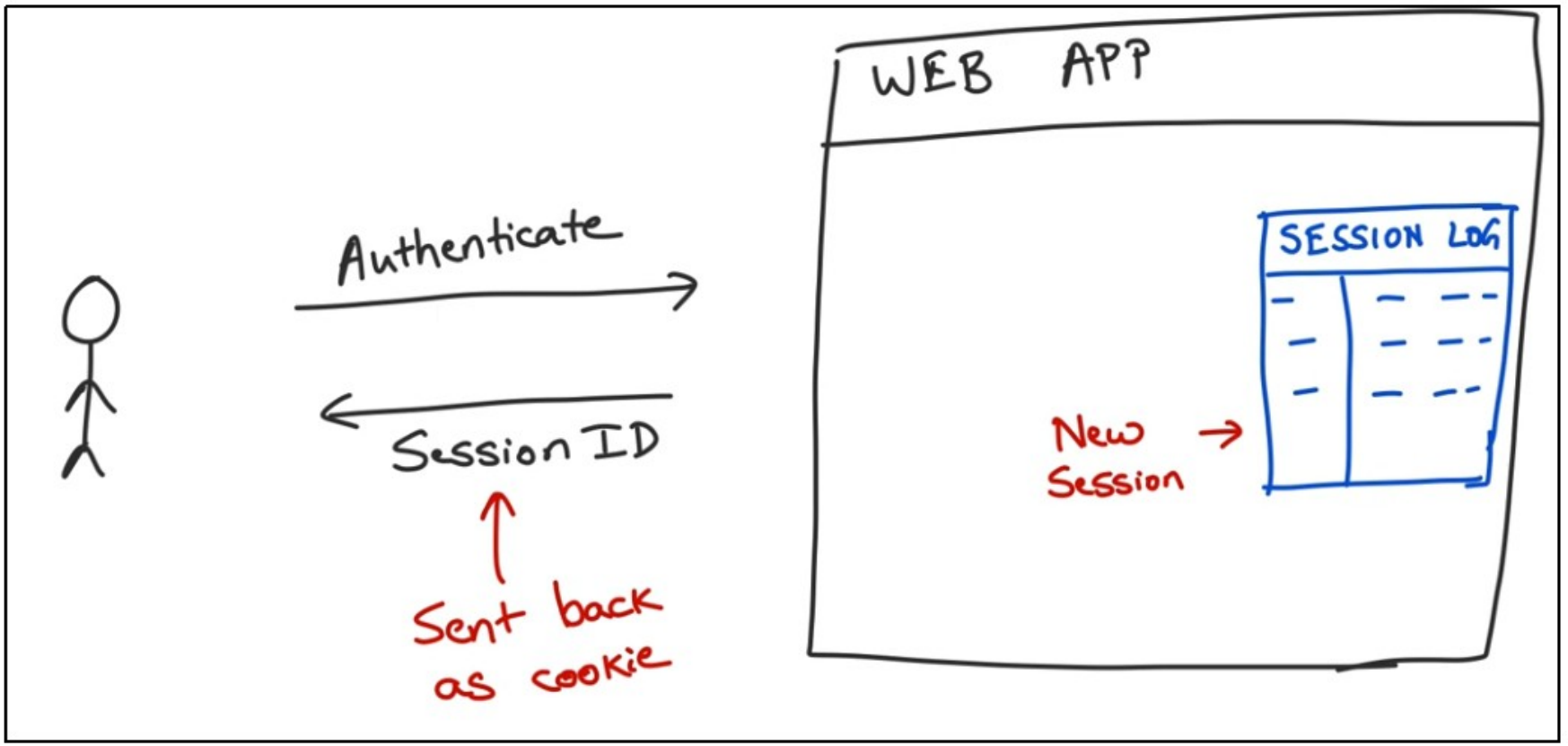
Authorization strategies



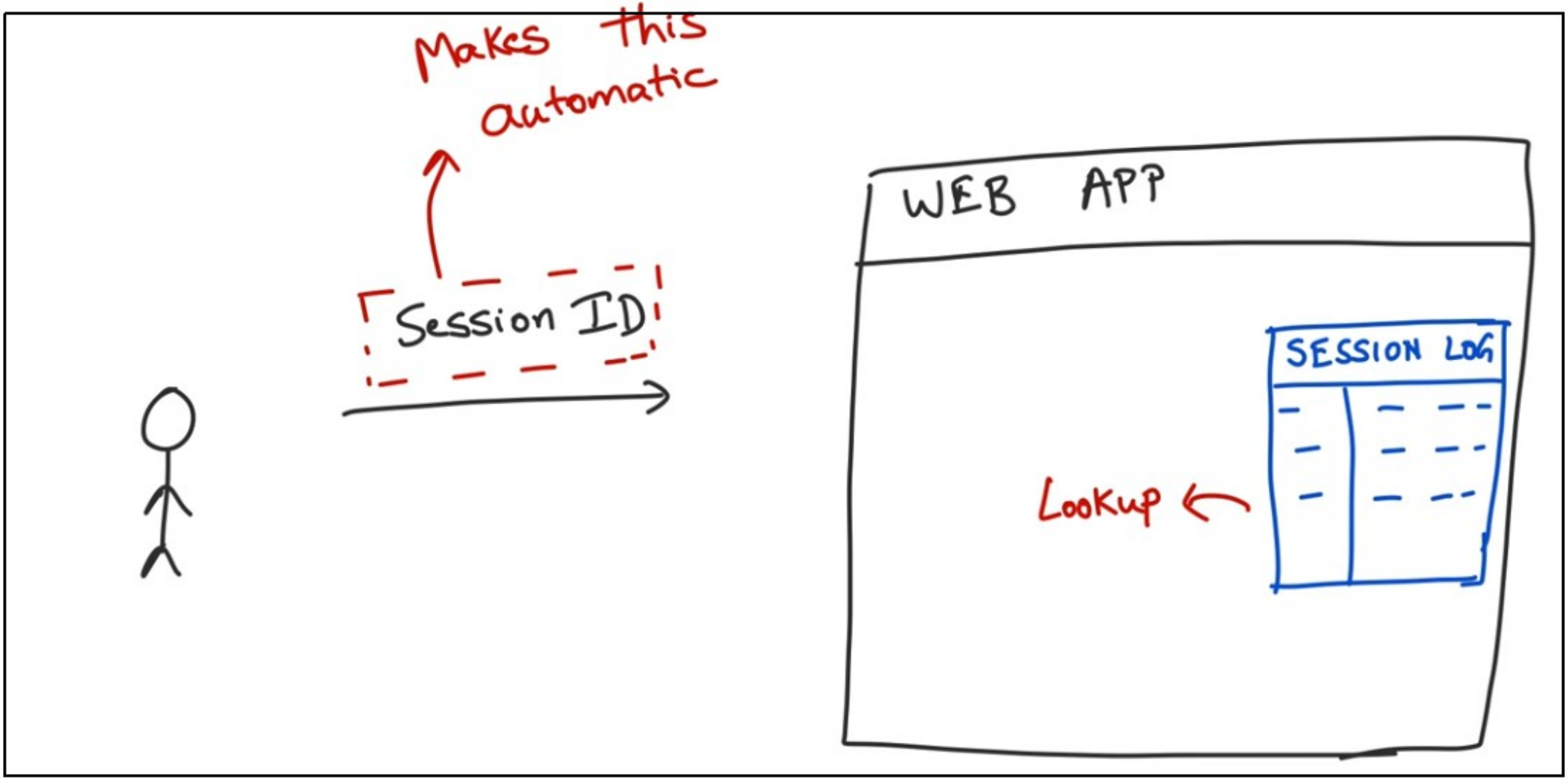
Session ID + Cookies

Most popular mechanism for authorization

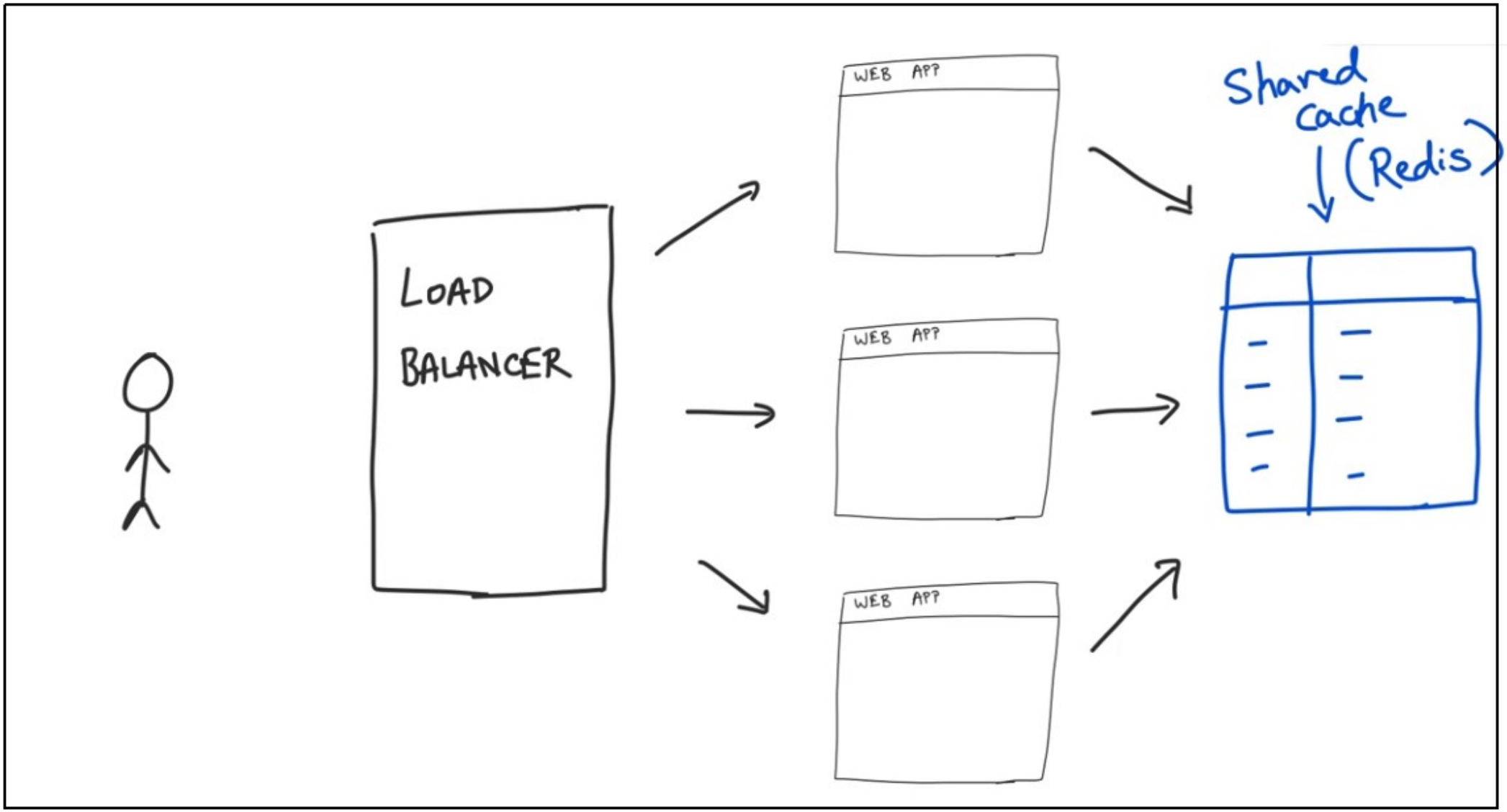
Session based authorization



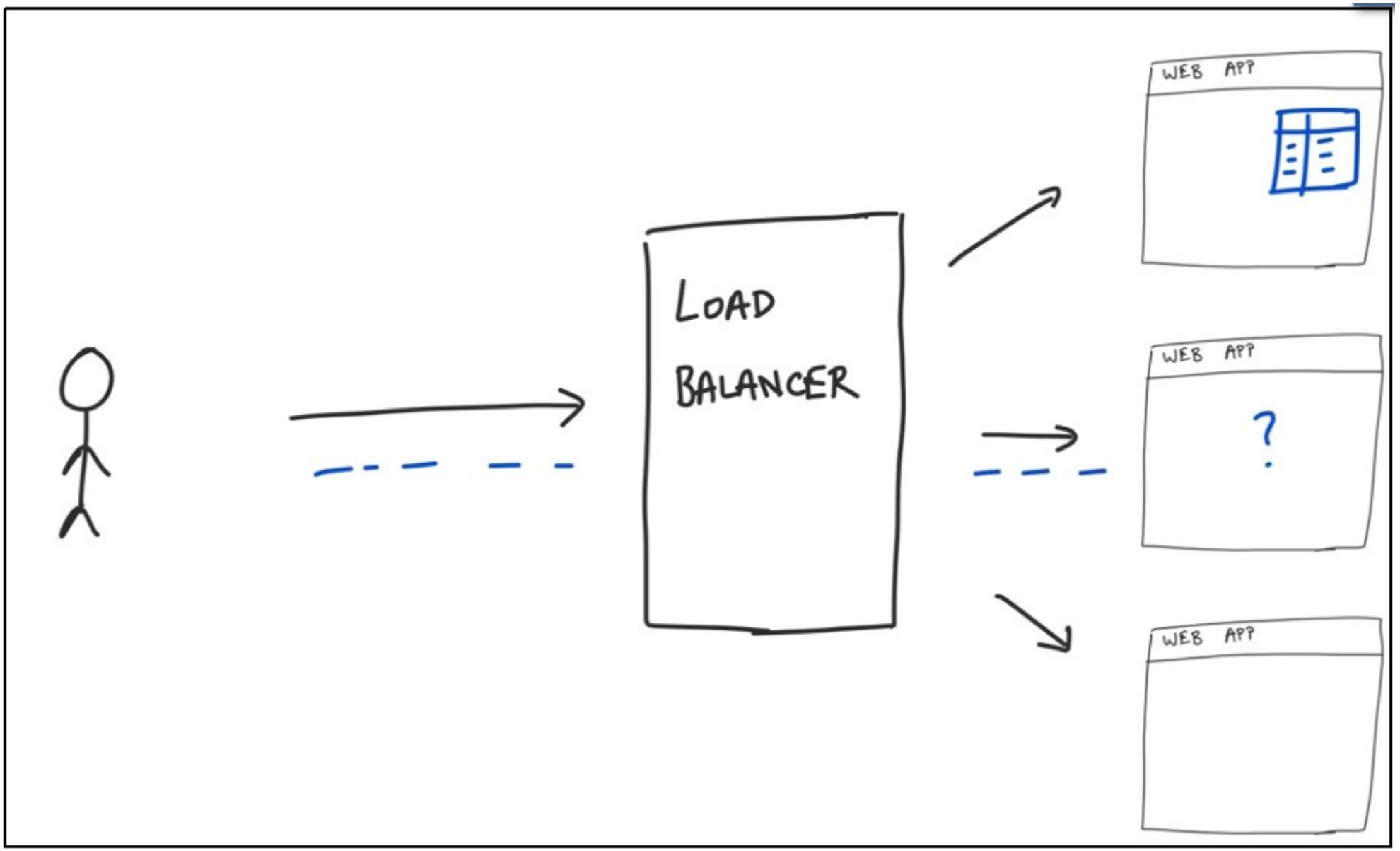
Session based authorization



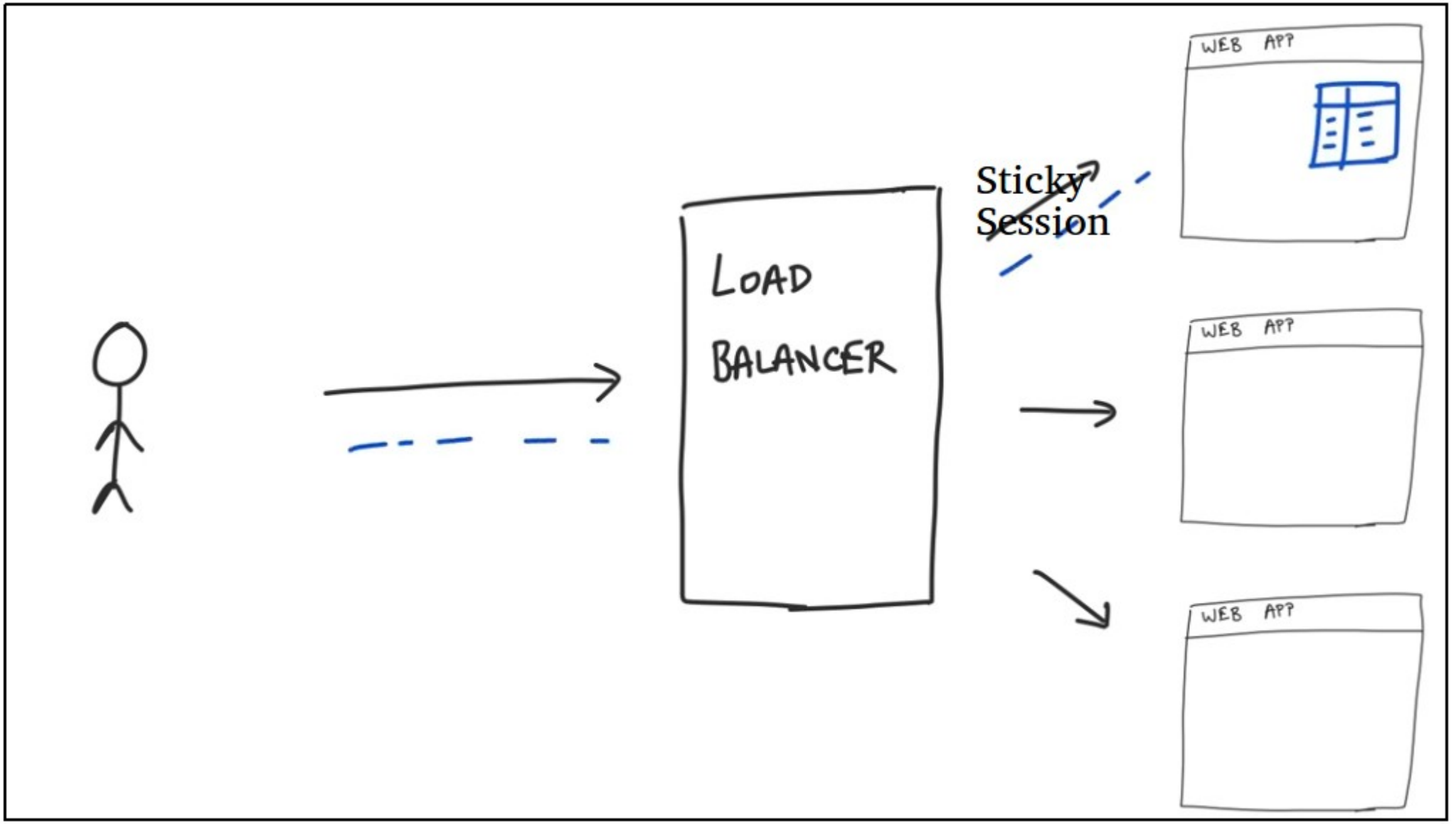
Session based authorization



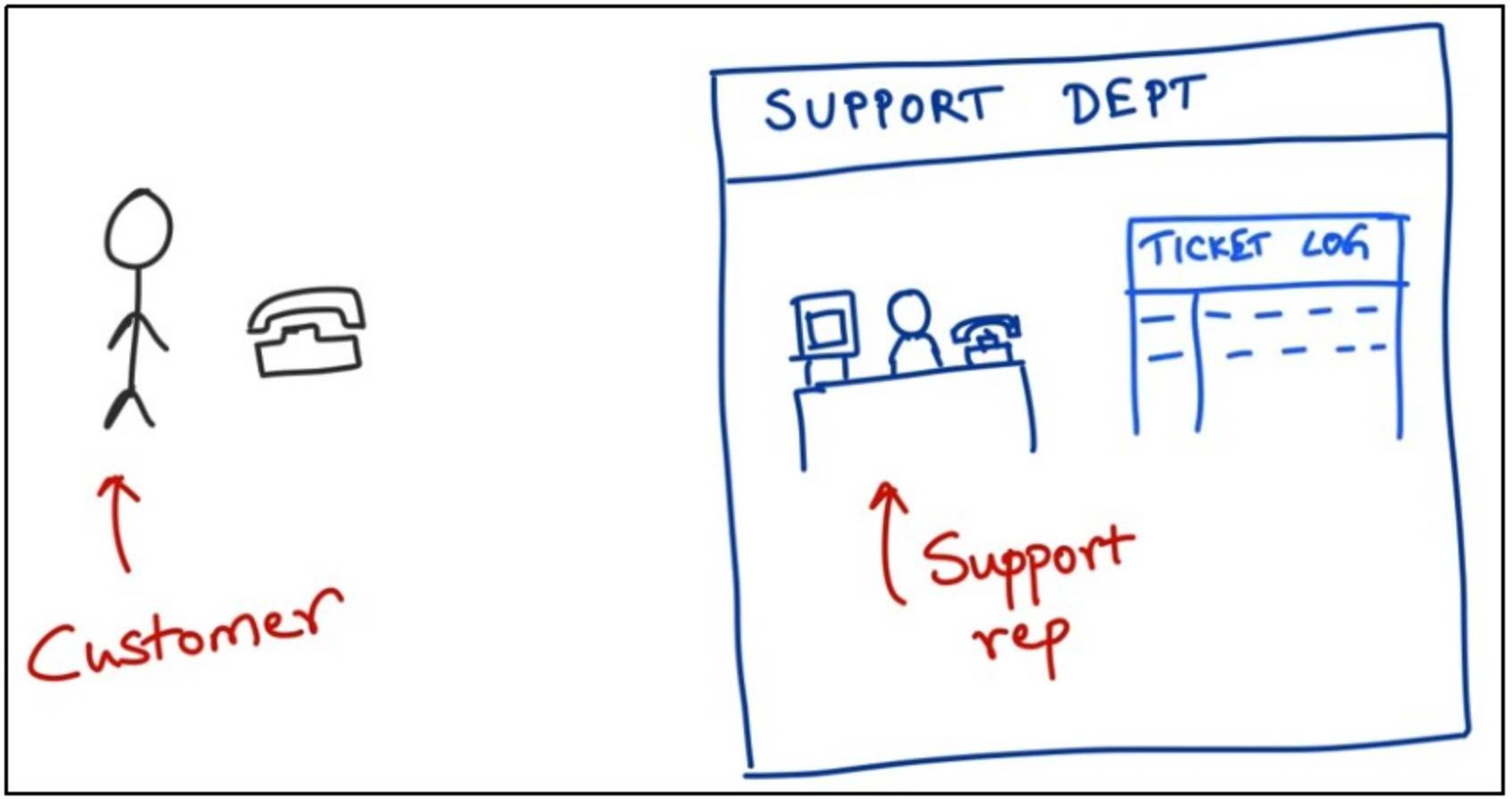
Session based authorization



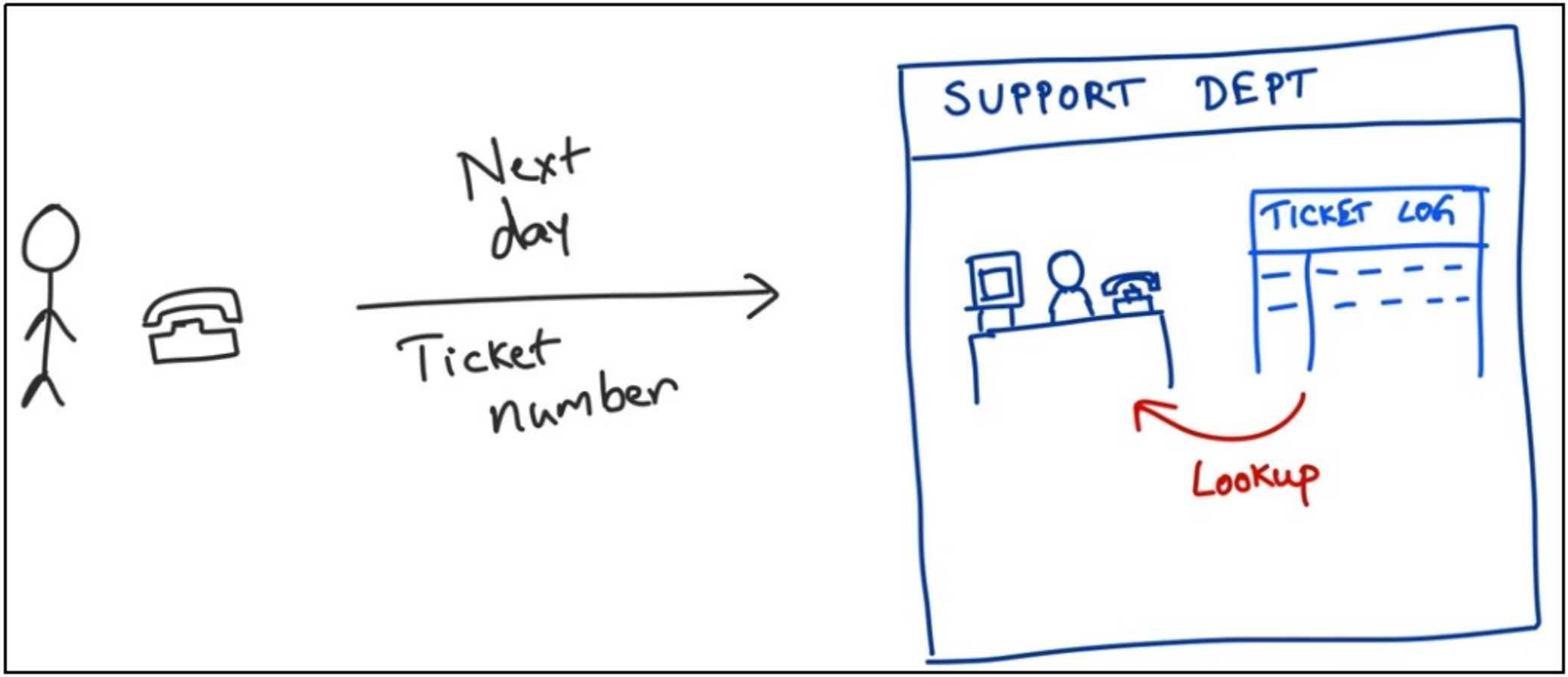
Session based authorization



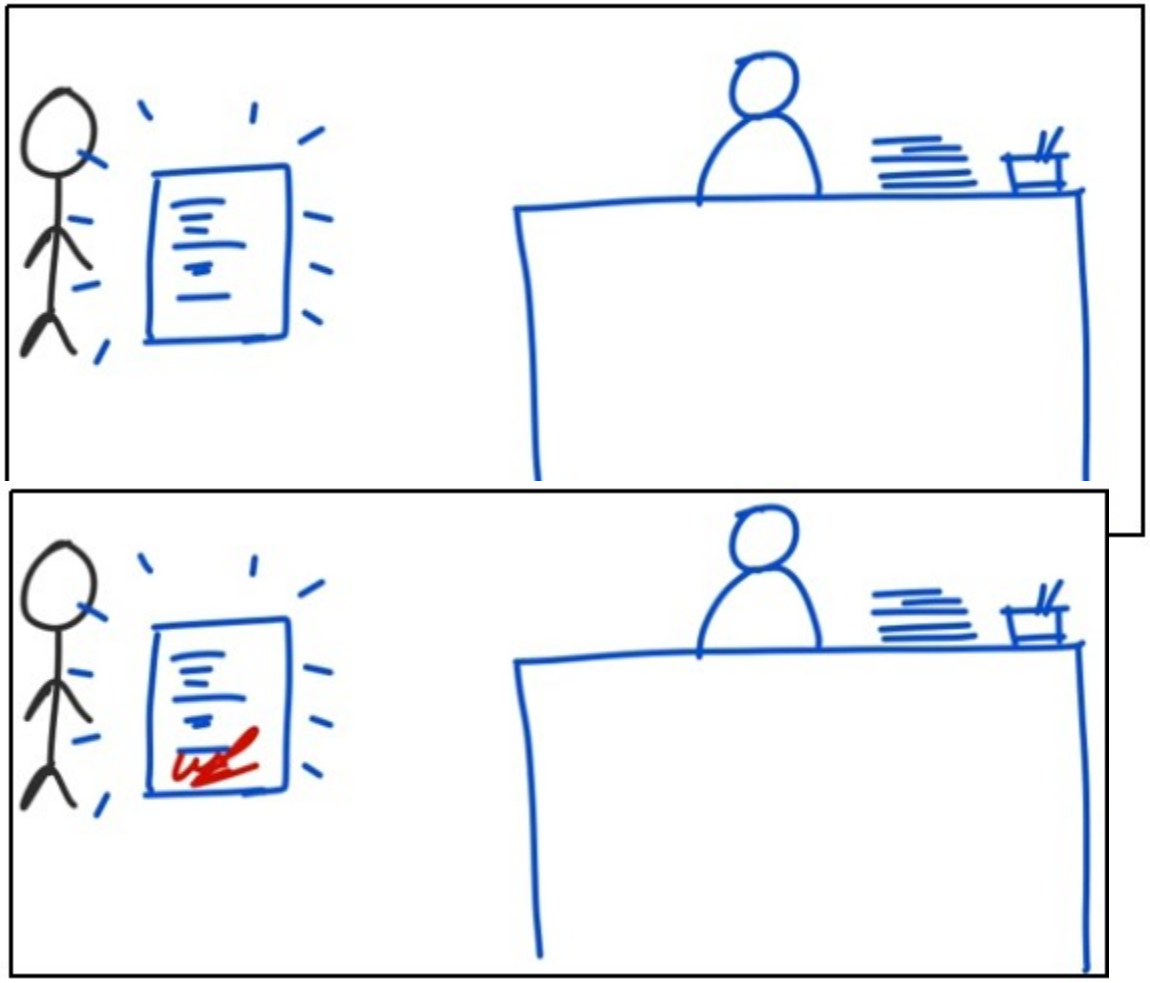
Session based authorization



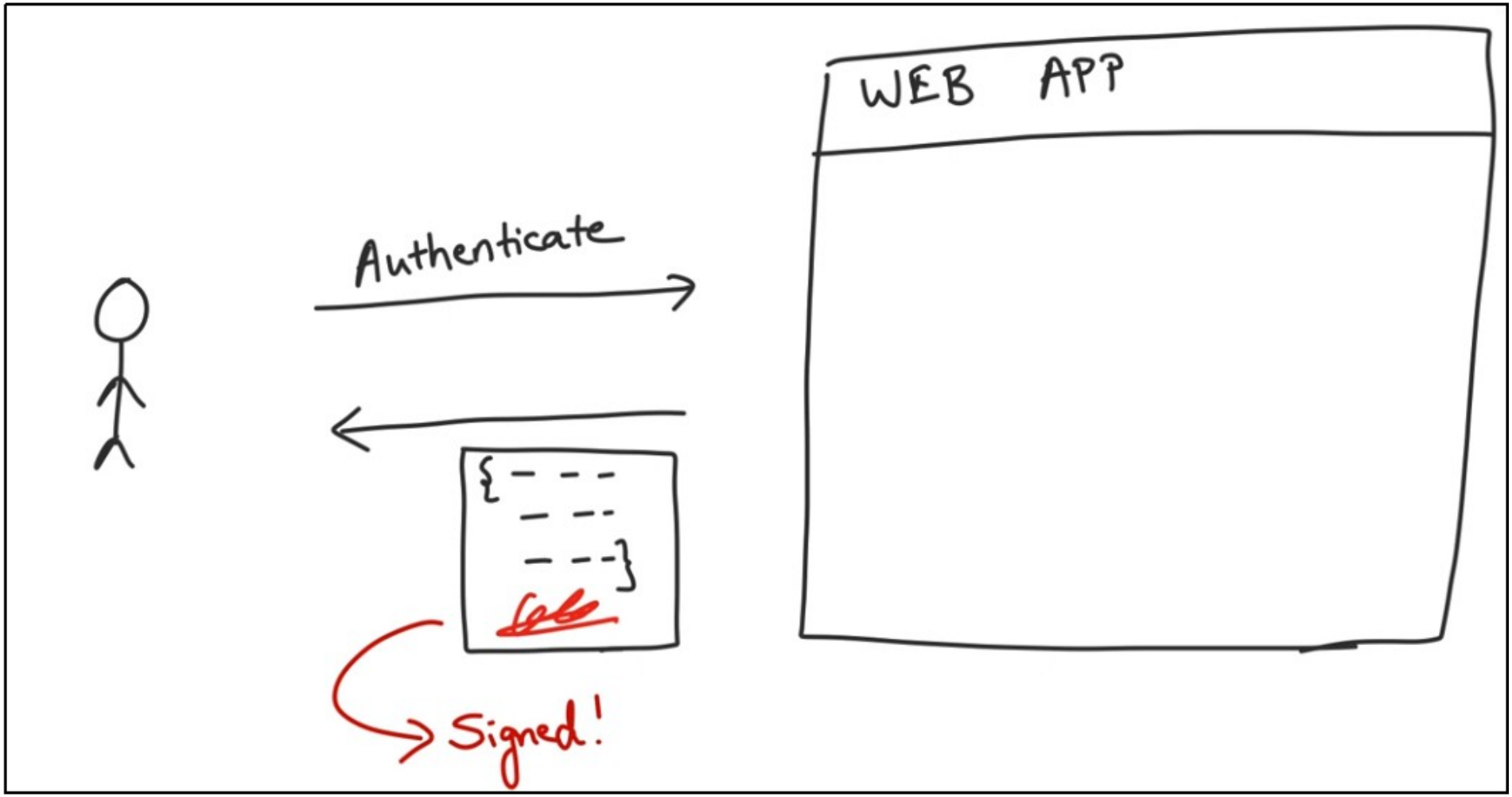
Session based authorization



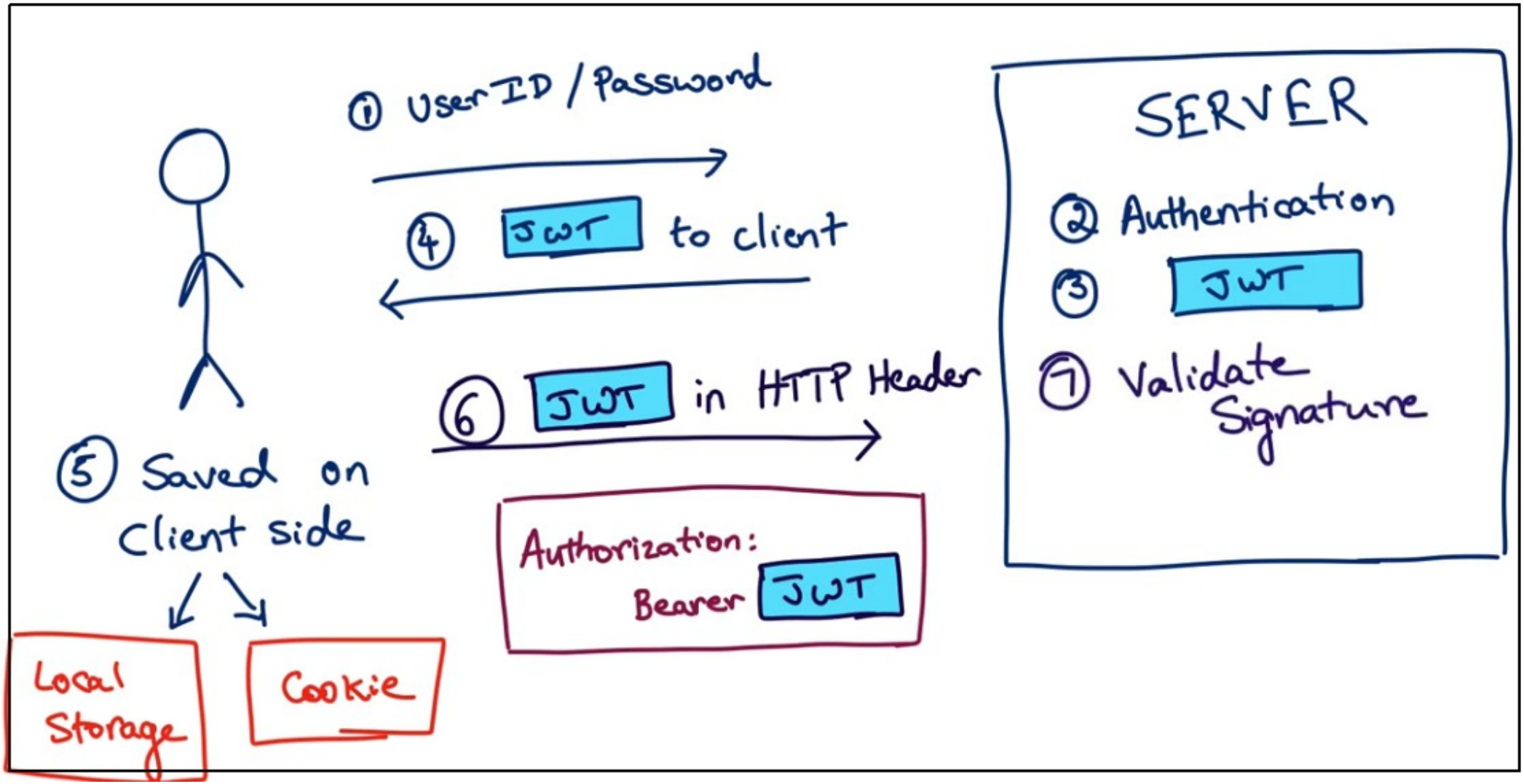
JWT based token



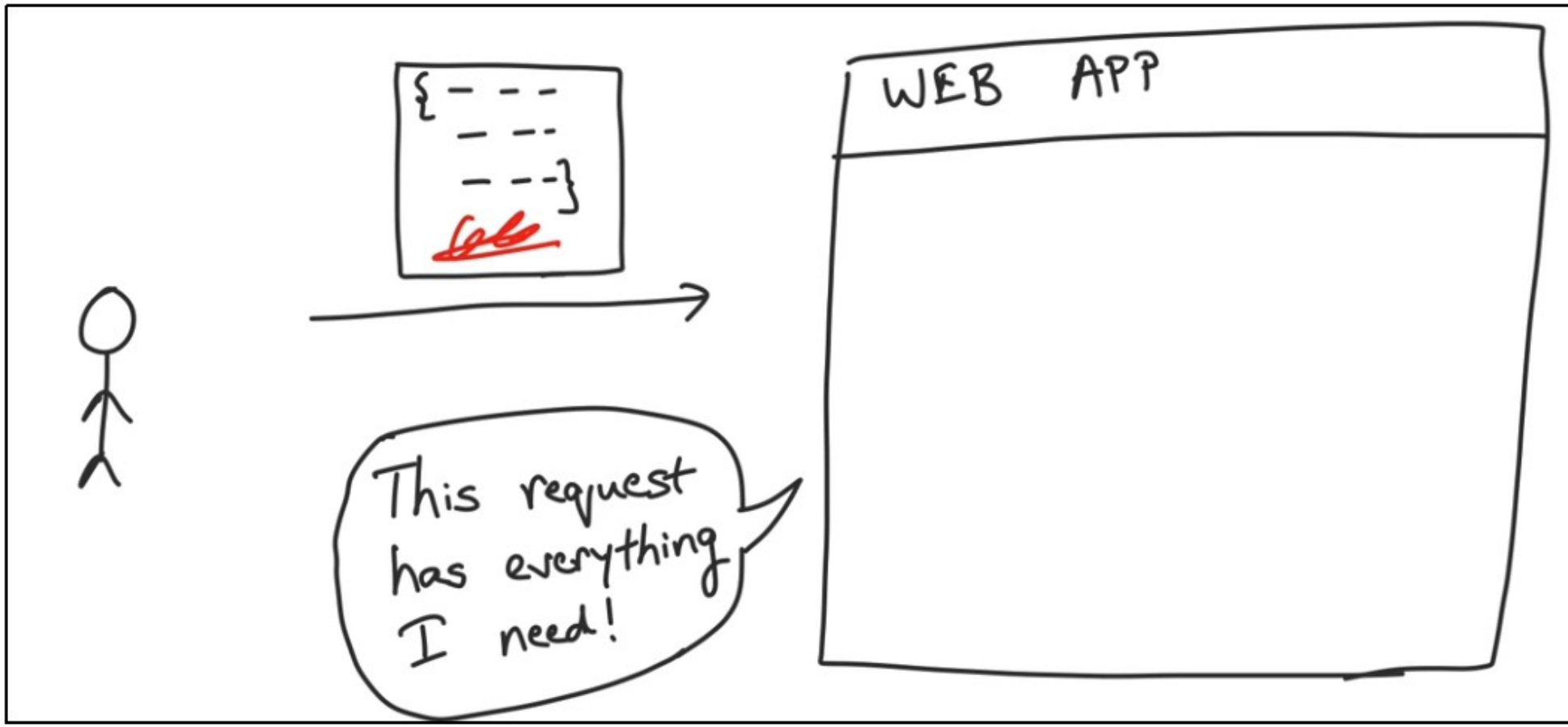
JWT based token



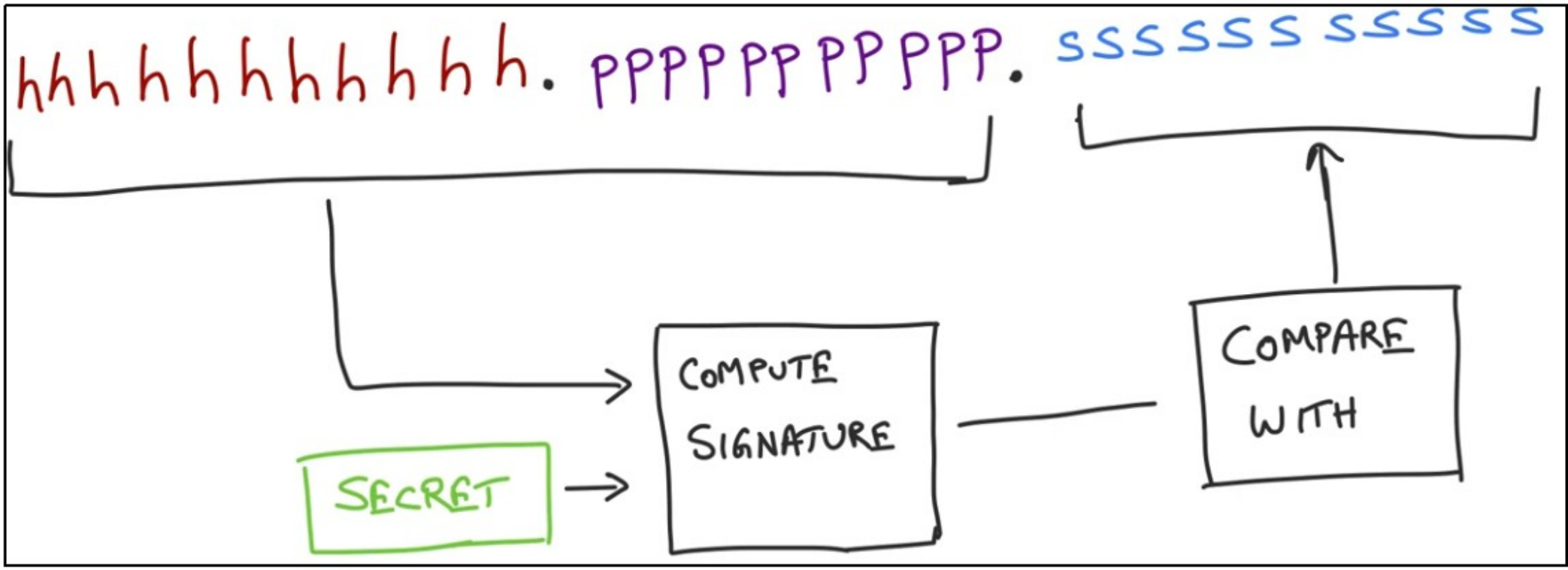
JWT token flow

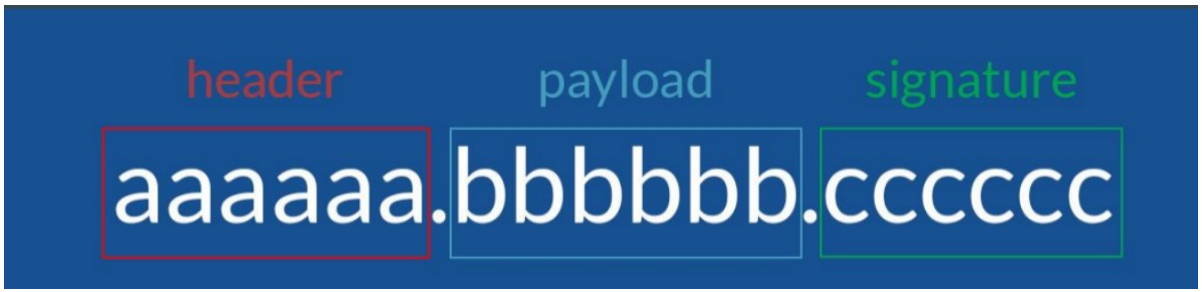


JWT Token flow

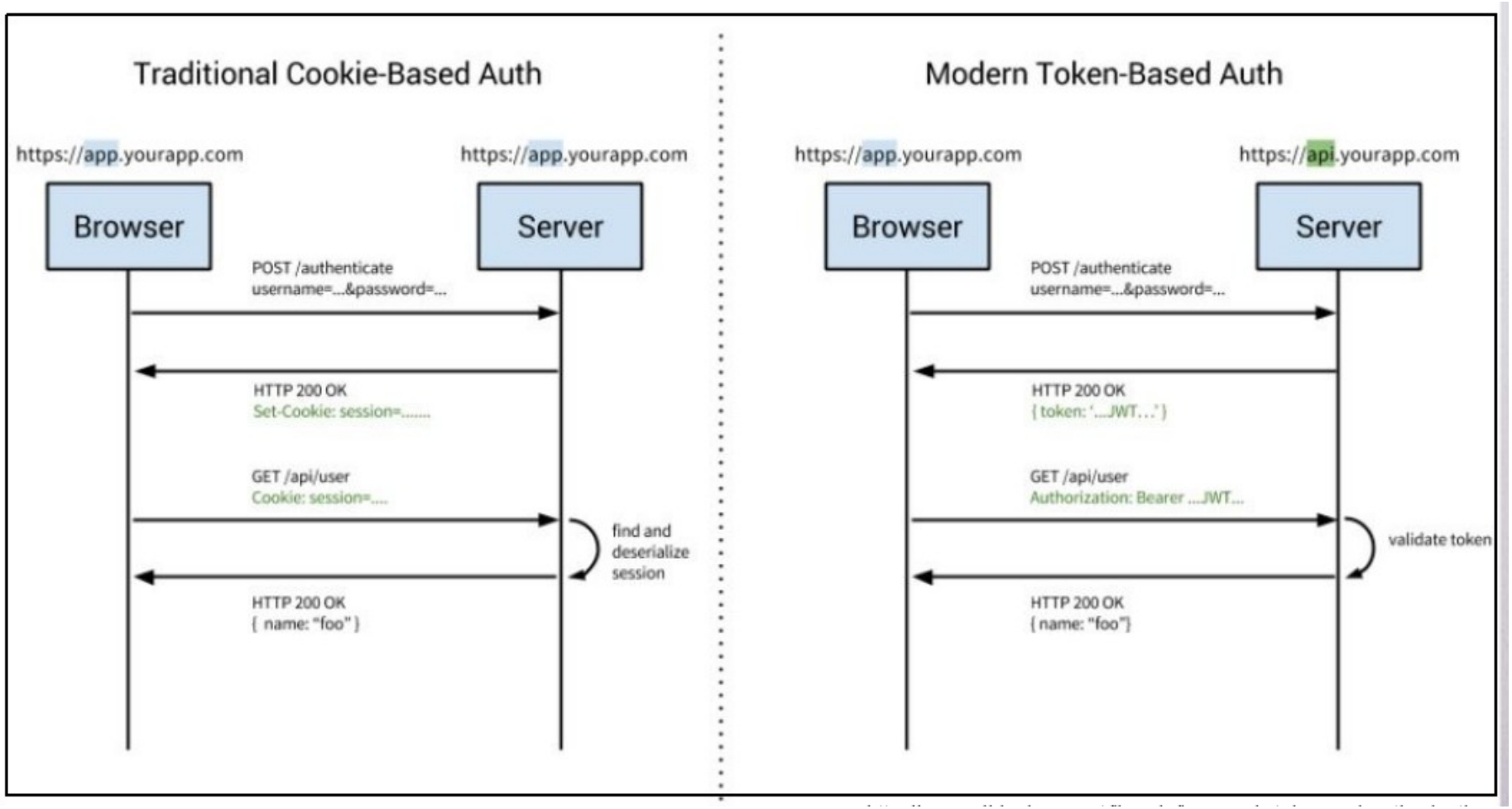


JWT token

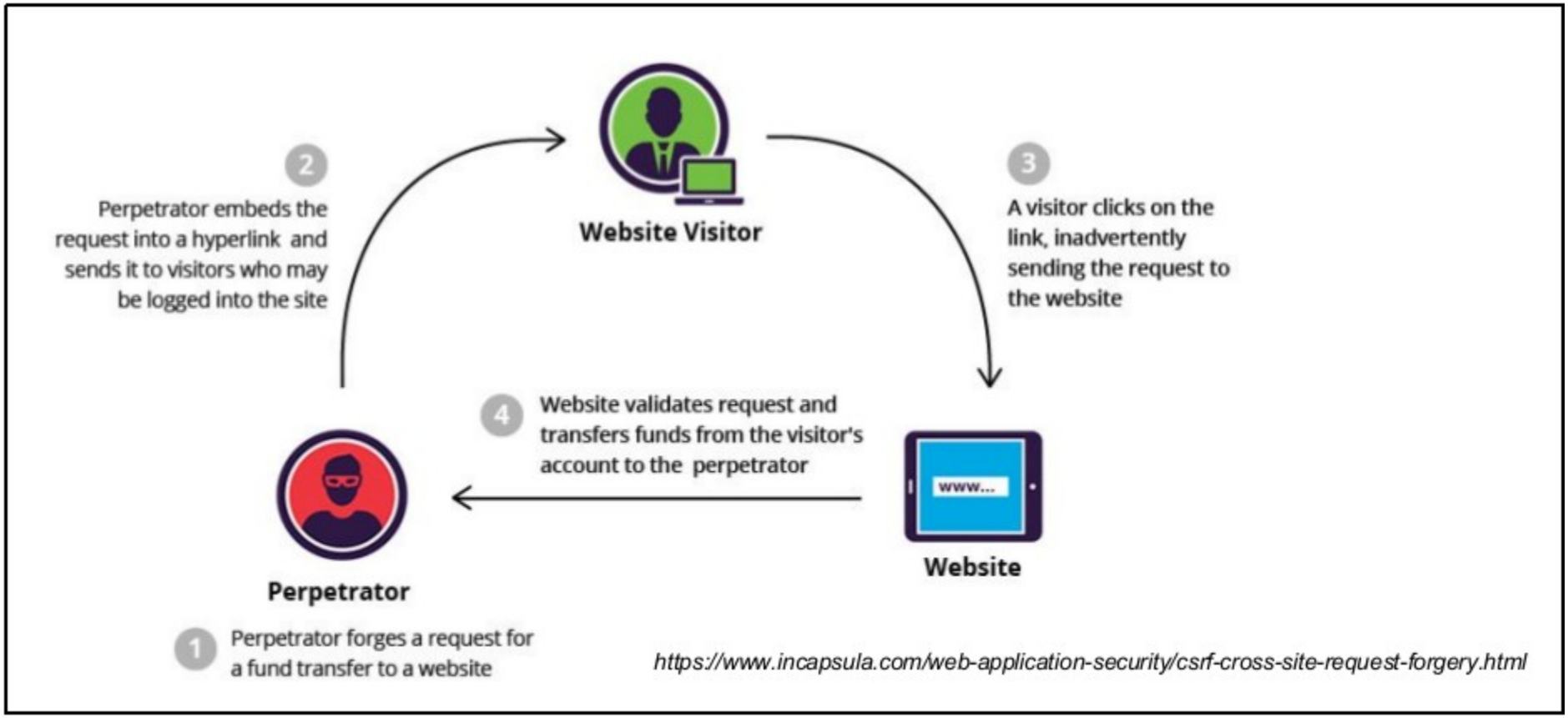

hhhhhhhhhhhh. ppppppppppp. sssssss sssss s
COMPUTE SIGNATURE
SECRET
COMPARE WITH


header
payload
signature
aaaaaa.bbbbbb.cccccc

Cookie vs JWT based authorization



No need to protect against CSRF



Header

Parts of the header :

- declaring the type, which is JWT
- the hashing algorithm to use

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

Common JWT Signing Algorithms

HS256	HMAC using SHA-256
RS256	RSASSA-PKCS1-v1_5 using SHA-256
ES256	ECDSA using P-256 and SHA-256

Payload

Carry the information that we want to transmit, also called the JWT Claims.


```
{
  "iss": "scotch.io",
  "exp": 1300819380,
  "name": "Chris Sevilleja",
  "admin": true
}
```

Signature

Made up of a hash of the following components:

- the header
- the payload
- secret

```
var encodedString =
base64UrlEncode(header) + "." +
base64UrlEncode(payload);


HMACSHA256(encodedString,'secret');
```

Full JSON of JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. HEADER

eyJpc3MiOiJzY290Y2guaW8iLCJleHAiOjEz

MDA4MTkzODAsIm5hbWUiOiJDaHJpcyB CLAIMS

TZXZpbGxlamEiLCJhZG1pbiI6dHJ1ZX0.

03f329983b86f7d9a9f5fef85305880101d5e302 SIGNATURE

afafa20154d094b229f757

Step1: start with hello world spring boot security configuration

```java
@RestController
public class Hello {
    @GetMapping(path = "/hello")
    public String hello() {
        return "hello world";
    }
```

```java
@Service
public class DetailService implements UserDetailsService{
    @Override
    public UserDetails loadUserByUsername(String username)throws UsernameNotFoundException {
        return new User("raj", "raj", AuthorityUtils.createAuthorityList("ADMIN","MGR"));
    }

}
```

```java
@EnableWebSecurity
public class SecurityConfigurer extends WebSecurityConfigurerAdapter{

    @Autowired
    private UserDetailsService userDetailsService;


    @Override
    protected void configure(AuthenticationManagerBuilder auth)
            throws Exception {
        auth.userDetailsService(userDetailsService);
    }
    @Bean
    public PasswordEncoder getPasswordEncoder(){
        return NoOpPasswordEncoder.getInstance();
    }
}
```

Step2: put jwt dependency and util

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

```java
@Service
public class JwtUtil {

    private String SECRET_KEY = "secret";

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }
    private Claims extractAllClaims(String token) {
        return Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBody();
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }
```

```java
public String generateToken(UserDetails userDetails) {
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, userDetails.getUsername());
}

private String createToken(Map<String, Object> claims, String subject) {

    return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis(
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10))
            .signWith(SignatureAlgorithm.HS256, SECRET_KEY).compact();
}

public Boolean validateToken(String token, UserDetails userDetails) {
    final String username = extractUsername(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
}
```

Step3: create endpoint to accept jwt token

- Accept username and password
- Return jwt token in response

- => create bean to send request

```
public class AuthRequest {
    private String username;
    private String password;
    public AuthRequest() {}
```

=> create bean to get response

```
public class AuthResponse {
    private String jwtToken;

    public AuthResponse(String jwtToken) {
        this.jwtToken = jwtToken;
    }
```

Step3: create endpoint to accept jwt token

```java
@RestController
public class Hello {
    @Autowired
    private AuthenticationManager authManager;
    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping(path = "/authenticate")
    public ResponseEntity<AuthResponse> createAuthToken(@RequestBody AuthRequest authRequest) throws Exception {
        try{
        authManager.authenticate(
                new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPassword())
        );
        }catch(BadCredentialsException ex){
            throw new Exception("user name is invalid", ex);
        }
        UserDetails userDetails=userDetailsService.loadUserByUsername(authRequest.getUsername());

        final String jwtToken=jwtUtil.generateToken(userDetails);

        return ResponseEntity.ok().body(new AuthResponse(jwtToken));
    }
```

Step4: Permit all /authenticate to accept jwt token

Dont forget to Override this method otherwise @Autowire AuthenticationManger will fail

```
@Override
@Bean
public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
}
@Override
public void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
    .authorizeRequests().antMatchers("/authenticate/**").permitAll()
        .anyRequest().authenticated()
        .and()
    .formLogin().and()
    .httpBasic();
}
```

Step 5

- Intercept all incoming request
  - Extract JWT token for the header
  - Validate and set in execution context

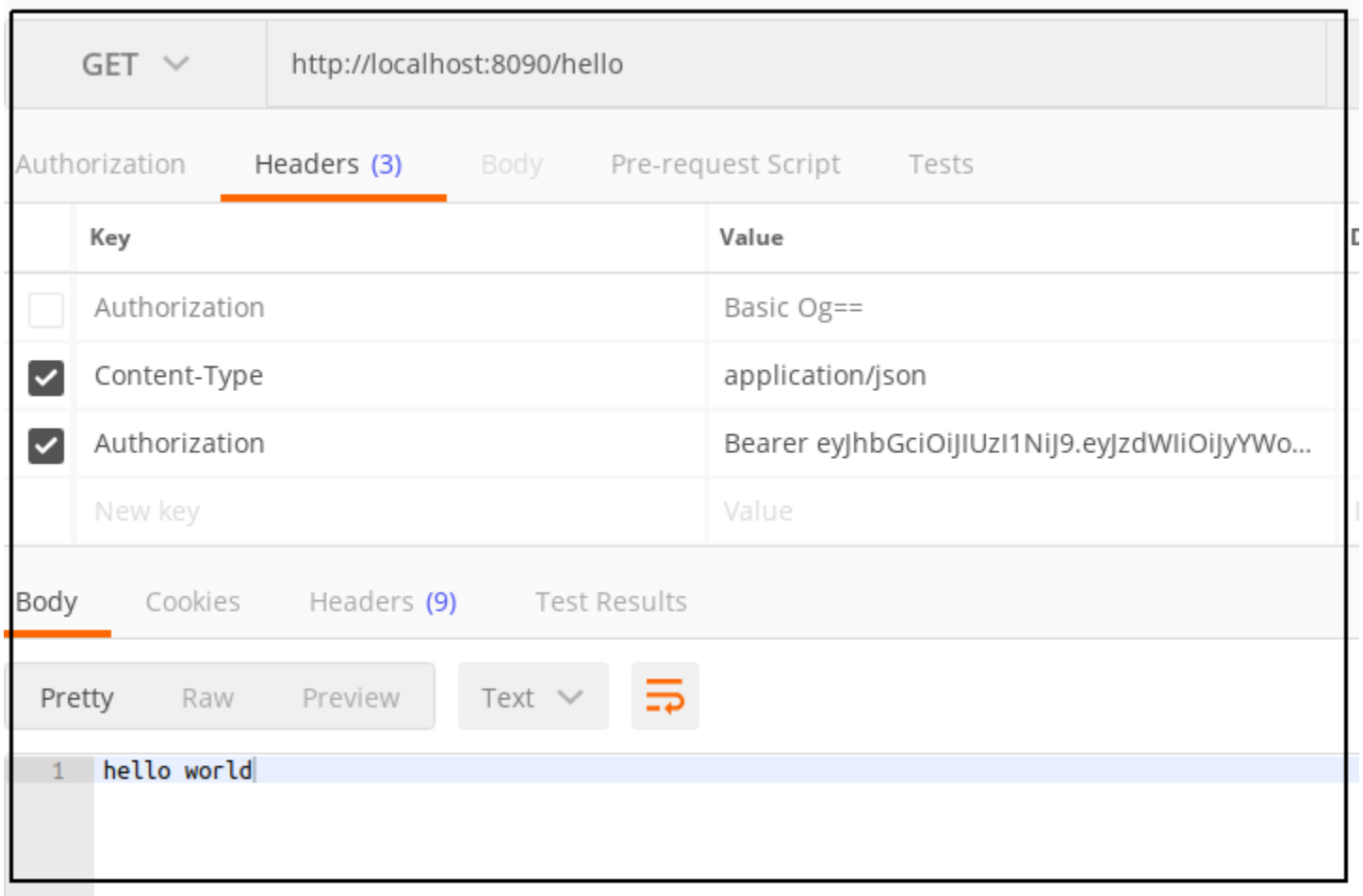
- Intercept all incoming request

```java
@Component
public class JwtRequestFilter extends OncePerRequestFilter{
    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private JwtUtil jwtUtil;
    @Override
    protected void doFilterInternal(HttpServletRequest request,HttpServletResponse response,
            FilterChain filterChain)throws ServletException, IOException {
        final String authHeader=request.getHeader("Authorization");
        String jwt=null;
        String username=null;
        //it must contain Bearer and and valid jwt token for authorization
        if(authHeader!=null && authHeader.startsWith("Bearer ")){
            jwt=authHeader.substring(7);
            username=jwtUtil.extractUsername(jwt);
        }
        //now extract userDetails related to username
        if(username!=null && SecurityContextHolder.getContext().getAuthentication()==null){
            UserDetails userDetails=this.userDetailsService.loadUserByUsername(username);
            if(jwtUtil.validateToken(jwt, userDetails)){
                UsernamePasswordAuthenticationToken
                authenticationToken=new UsernamePasswordAuthenticationToken(userDetails,
                        null, userDetails.getAuthorities());
                authenticationToken.setDetails(new WebAuthenticationDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authenticationToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

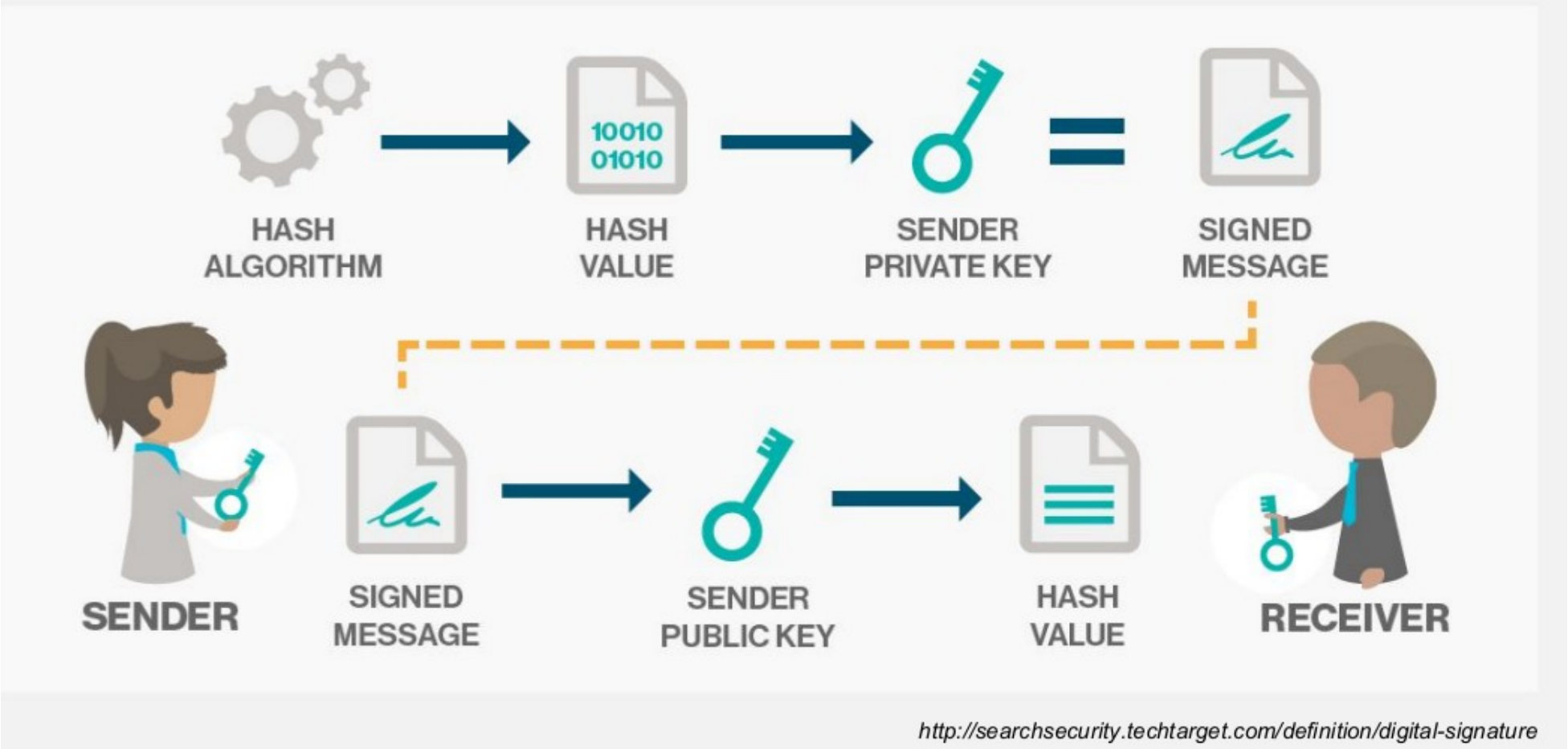
Appying filter to customized security

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http.csrf().disable()
    .authorizeRequests().antMatchers("/authenticate/**").permitAll()
        .anyRequest().authenticated()
        .and().sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.addFilterBefore(jwtRequestFilter,
            UsernamePasswordAuthenticationFilter.class);
}
```

Testing with jwt token



Digital signature process



http://searchsecurity.techtarget.com/definition/digital-signature