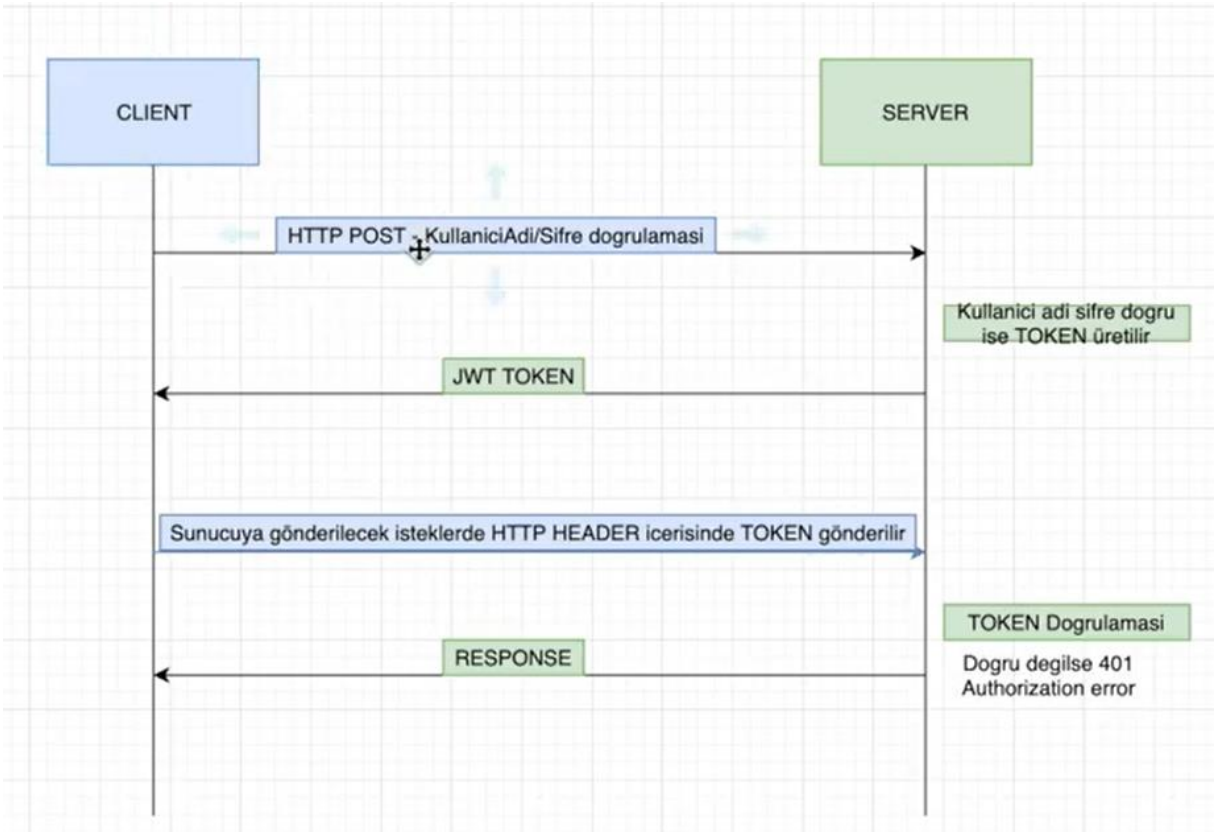
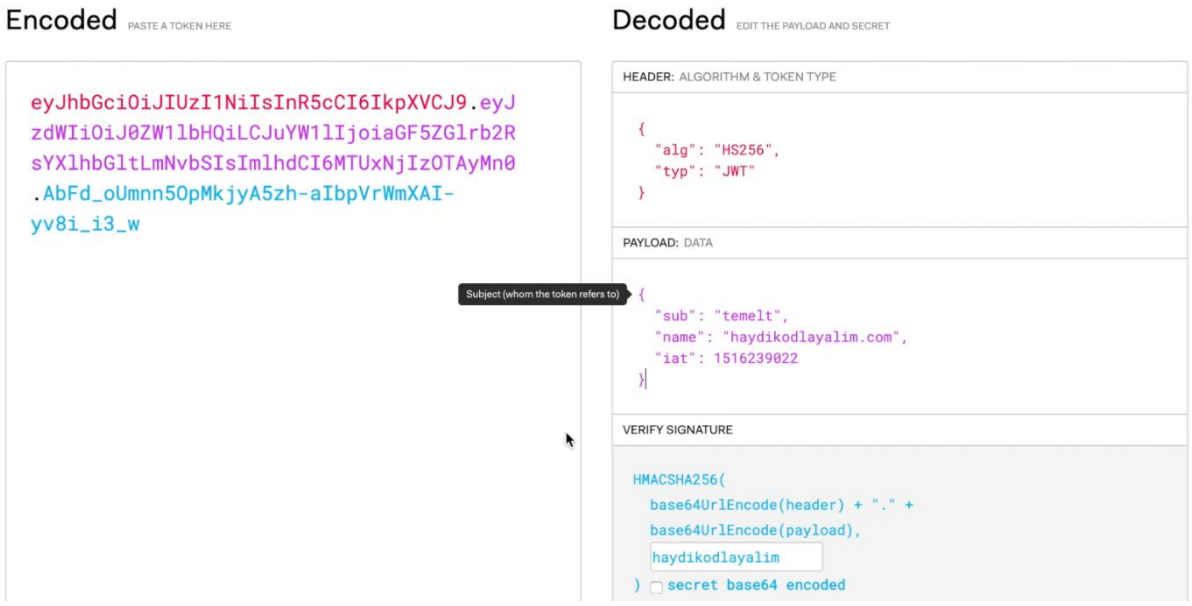
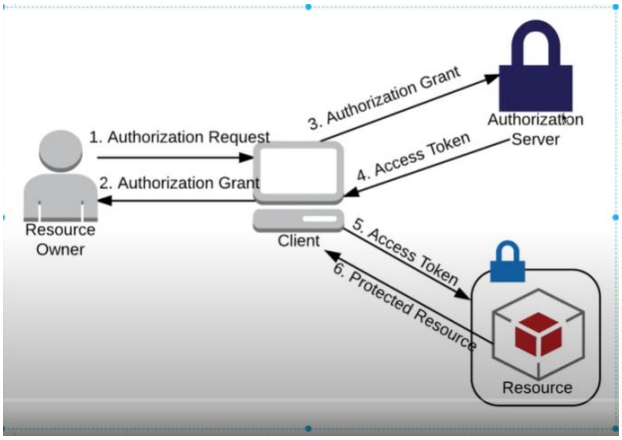
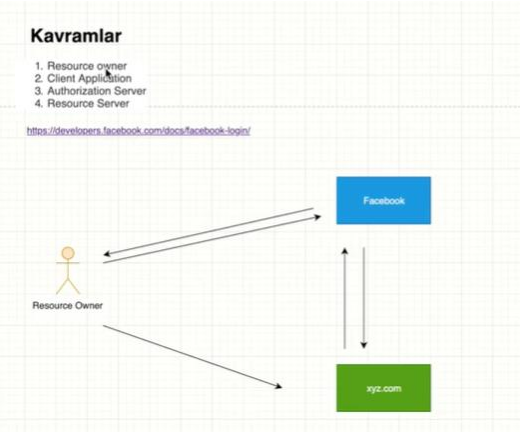
JWT

Request ve response larımız üzerinde servere giden isteklerin aynı kişiden gidip gitmediğini doğrulamak için kullandığımız yöntem









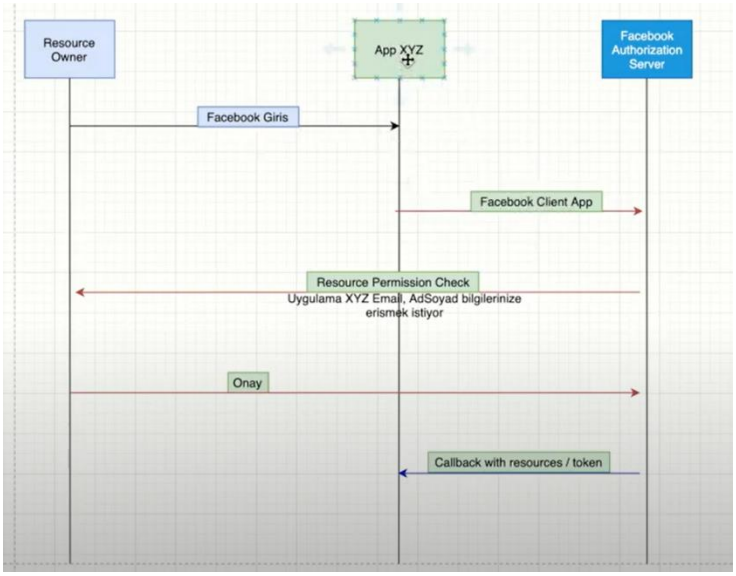
Twitter örneği üzerinde

Resource owner : kullanıcımız,

Client application: geliştirdiğimiz uygulama

Authorization server: yetkilendirme sunucusu,

Resource sercer :Twitlerin saklı olduğu sunucu



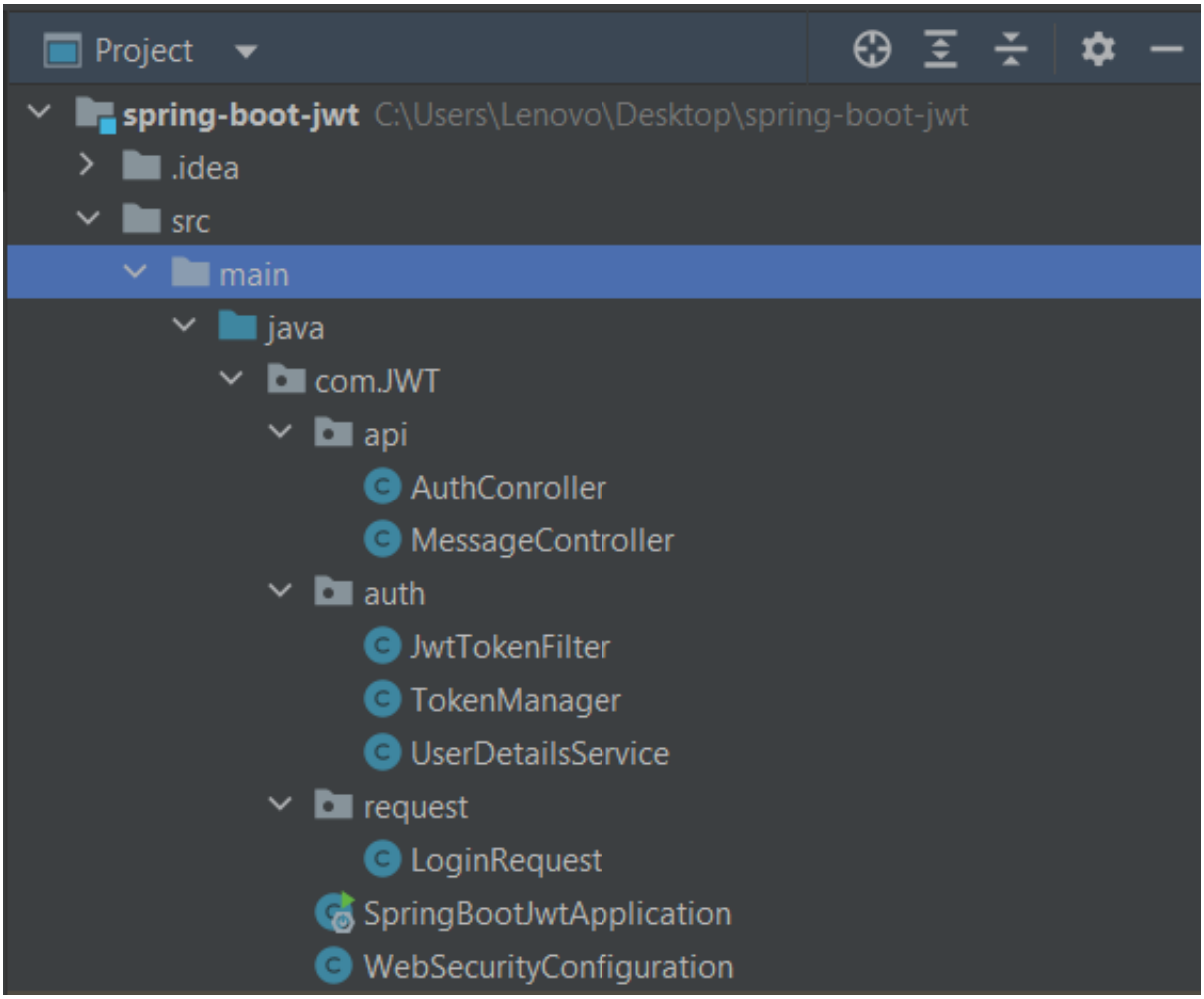
JWT yi request body ye eklemiyoruz, path variable a eklemiyoruz.

Requestte jwt yi ekleyeceğimiz yer header. Header daki authorization parametresi.

403 veya 401 görüyosan jwt ile ilgili bir sorun var demektir.

Db ye passwordlerin açık açık yazılmaması lazım ayrıca.

Mailin daha önce db de kayıt olup olmadığına bakacaksın.



MAIN CLASS

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootJwtApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootJwtApplication.class, args);
    }
}
/**
 POST request --> localhost:8080/login

 raw-json -->
                {
                "username" : "burak",
                "password" : "123"
                }

 tokenı üretir

 GET request bearer token --> localhost:8080/message

 tokenı kullanır.
 */
```

WebSecurityConfiguration

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    private JwtTokenFilter jwtTokenFilter;

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    public void configurePasswordEncoder(AuthenticationManagerBuilder builder) throws
Exception {

builder.userDetailsService(userDetailsService).passwordEncoder(getBCryptPasswordEncoder());
    }

    @Bean
    public BCryptPasswordEncoder getBCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }

   @Bean
    public AuthenticationManager getAuthenticationManager() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
                .authorizeRequests().antMatchers("/login").permitAll()
                .anyRequest().authenticated()

.and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        http.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);
    }
}
```

LoginRequest

```
public class LoginRequest {

    private String username;

    private String password;

    public LoginRequest() {
    }

    public LoginRequest(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

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

```
//                                    AuthController

@RestController
@RequestMapping("/login")
public class AuthConroller {

    @Autowired
    private TokenManager tokenManager;

    @Autowired
    private AuthenticationManager authenticationManager;

    @PostMapping
    public ResponseEntity<String> login(@RequestBody LoginRequest loginRequest) {
        try {
            authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(loginRequest.getUsername(),
loginRequest.getPassword()));

            return ResponseEntity.ok(tokenManager.generateToken(loginRequest.getUsername()));
        } catch (Exception e) {
            throw e;
        }
    }
}
```

MessageController

```
@RestController
@RequestMapping("/message")
public class MessageController {

    @GetMapping
    public ResponseEntity<String> getMessage() {
        return ResponseEntity.ok("Merhaba JWT");
    }
}
```

JwtTokenFilter

```
@Component
public class JwtTokenFilter extends OncePerRequestFilter {

    @Autowired
    private TokenManager tokenManager;

    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest,
                                    HttpServletResponse httpServletResponse,
                                    FilterChain filterChain) throws ServletException, IOException {

        /**
         * "Bearer 123hab2355"
         */
        final String authHeader = httpServletRequest.getHeader("Authorization");

        String username = null;
        String token = null;


        if (authHeader != null && authHeader.contains("Bearer")) {
            token = authHeader.substring(7);
            try {
                username = tokenManager.getUsernameToken(token);
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }

        if (username != null && token != null
                && SecurityContextHolder.getContext().getAuthentication() == null) {
            if (tokenManager.tokenValidate(token)) {
                UsernamePasswordAuthenticationToken upassToken =
                        new UsernamePasswordAuthenticationToken(username, null, new ArrayList<>());
                upassToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(httpServletRequest));
                SecurityContextHolder.getContext().setAuthentication(upassToken);
            }
        }

        filterChain.doFilter(httpServletRequest, httpServletResponse);
    }
}
```

TokenManager

```
@Service
public class TokenManager {

    private static final int validity = 10 * 1000;
    Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuer("www.JWT.com")
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + validity))
                .signWith(key)
                .compact();
    }

    public boolean tokenValidate(String token) {
        if (getUsernameToken(token) != null && isExpired(token)) {
            return true;
        }
        return false;
    }

    public String getUsernameToken(String token) {
        Claims claims = getClaims(token);
        return claims.getSubject();
    }

    public boolean isExpired(String token) {
        Claims claims = getClaims(token);
        return claims.getExpiration().after(new Date(System.currentTimeMillis()));
    }

    private Claims getClaims(String token) {
        return Jwts.parser().setSigningKey(key).parseClaimsJws(token).getBody();
    }

}
```

UserDetailsService

```java
@Service
public class UserDetailsService implements
org.springframework.security.core.userdetails.UserDetailsService {

    private Map<String, String> users = new HashMap<>();

    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    @PostConstruct
    public void init() {
        users.put("burak", passwordEncoder.encode("123"));
    }


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        if (users.containsKey(username)) {
            return new User(username, users.get(username), new ArrayList<>());
        }

        throw new UsernameNotFoundException(username);
    }
}
```

https://ugurcemozturk.medium.com/json-web-tokens-jwts-ile-spring-boot-uygulamas%C4%B1n%C4%B1-g%C3%BCvenceye-alma-38577fb5354b

https://devnot.com/2017/json-web-token-jwt-standardi/

https://medium.com/@furkanbegen/oauth2-basit-anlat%C4%B1m-c5f9b542a0f2

https://docs.spring.io/spring-security/site/docs/5.0.x/reference/html/csrf.html