# API – Key Code Structure :

Security Configuration class

@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
@Autowired  
 private ApiKeyAuthFilter apiKeyAuthFilter;  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf().disable() // Disabling CSRF protection  
 .addFilterBefore(apiKeyAuthFilter, UsernamePasswordAuthenticationFilter.class) // Add our custom filter  
 .authorizeRequests()  
 .anyRequest().authenticated(); // All requests must be authenticated  
 }  
}

Filter class to check in header if the api key is present or not

@Component  
public class ApiKeyAuthFilter extends OncePerRequestFilter {  
@Value("${api.key}")  
 private String apiKey;  
 @Value("${api.secret}")  
 private String apiSecret;  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)  
 throws ServletException, IOException {  
   
 // Get the API key and secret from request headers  
 String requestApiKey = request.getHeader("X-API-KEY");  
 String requestApiSecret = request.getHeader("X-API-SECRET");  
 // Validate the key and secret  
 if (apiKey.equals(requestApiKey) && apiSecret.equals(requestApiSecret)) {  
 // Continue processing the request  
 filterChain.doFilter(request, response);  
 } else {  
 // Reject the request and send an unauthorized error  
 response.setStatus(HttpStatus.UNAUTHORIZED.value());  
 response.getWriter().write("Unauthorized");  
 }  
 }  
}

# Basic Authentication using Username and Password:

userDetailsService class to load the user from the DB

@Service  
public class UserDetail implements UserDetailsService {

@Autowired   
 UserRepository userRepo;

@Override  
 public UserDetails loadUserByUsername(String username)throws

UsernameNotFoundException

{  
 `User user = userRepo.findByUserNameOrEmail(username, username);  
 if(user==null){  
 throw new UsernameNotFoundException("User not exists by Username");  
 }  
   
 Set<GrantedAuthority> authorities = user.getRoles().stream()  
 .map((role) -> new

SimpleGrantedAuthority(role.getName()))  
 .collect(Collectors.toSet());

return new org.springframework.security.core.userdetails.User(username,user.getPassword(),authorities);  
 }  
}

Login API Controller :

@PostMapping("/login")  
 public ResponseEntity<String> authenticateUser(@RequestBody LoginDto loginDto) {  
 Authentication authentication = authenticationManager  
 .authenticate(new UsernamePasswordAuthenticationToken(loginDto.getUsername(), loginDto.getPassword())); SecurityContextHolder.getContext().setAuthentication(authentication);  
 return new ResponseEntity<>("User login successfully!...", HttpStatus.OK);  
 }

Config Class for this security mechanism

@Configuration  
public class SecurityConfig {

@Bean  
 public static PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }

@Bean  
 public AuthenticationManager authenticationManager

(AuthenticationConfiguration configuration) throws Exception {  
 return configuration.getAuthenticationManager();  
 }

@Bean  
 SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http.csrf().disable()  
 .authorizeRequests()

.antMatchers("/api/\*\*", "/h2-console/\*\*")

.permitAll()  
 .anyRequest().authenticated();  
 http.headers().frameOptions().disable();

return http.build();  
 }  
}

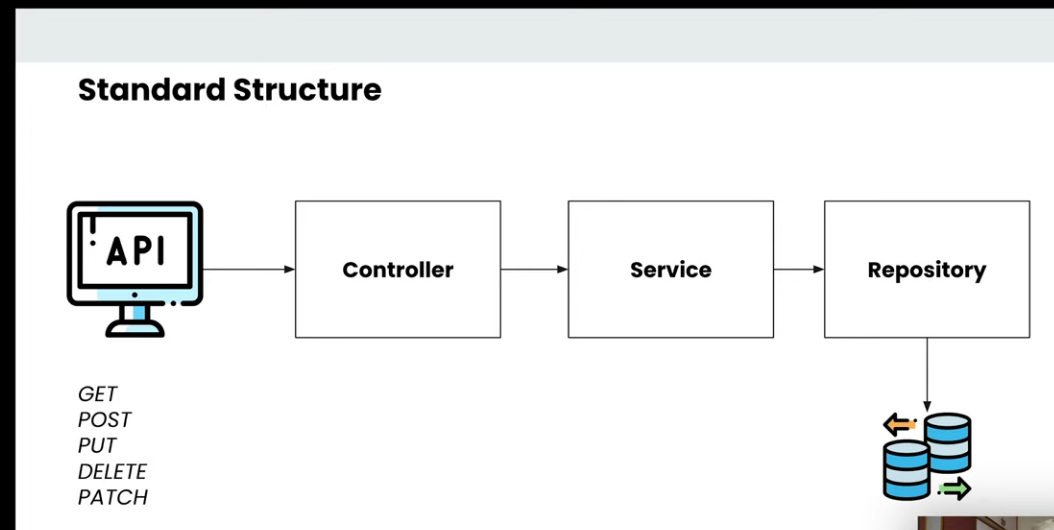
# What is stateless and Stateful beans

* Stateless beans are those beans which have ony one instances throughout the project
* Example of stateless beans are Singelton class.
* Stateful beans are those beans where every time we create a new instances .

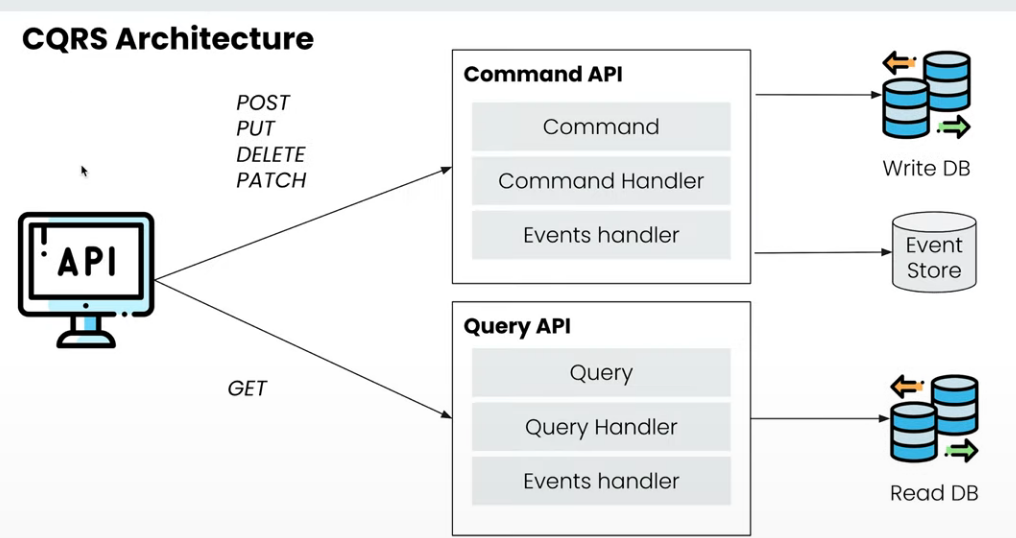
# Why we need @SpringBootApplications annotations :

Event Driven Desing Pattern

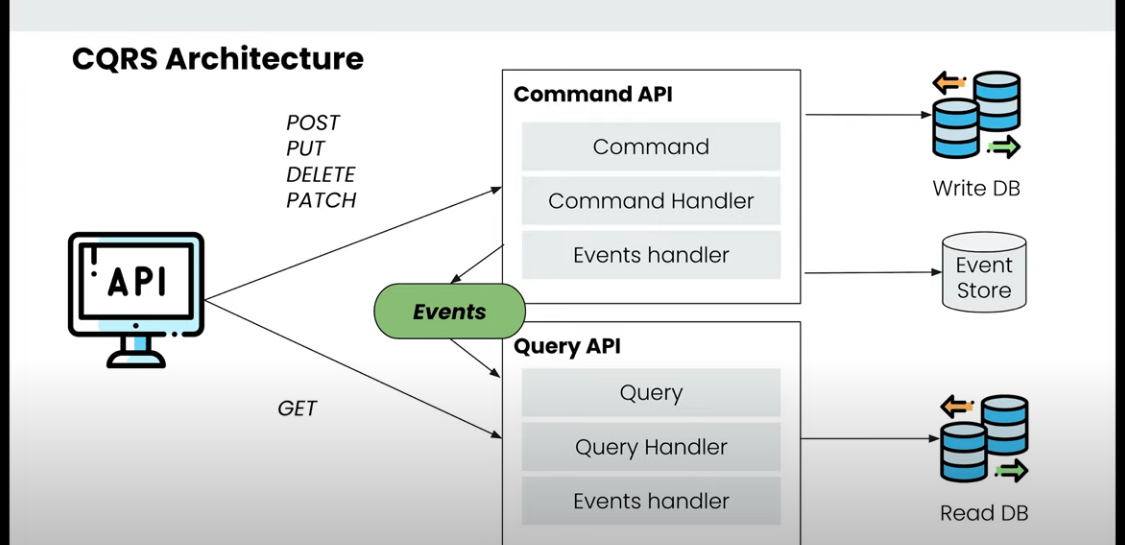
Standard structure of the services in the application



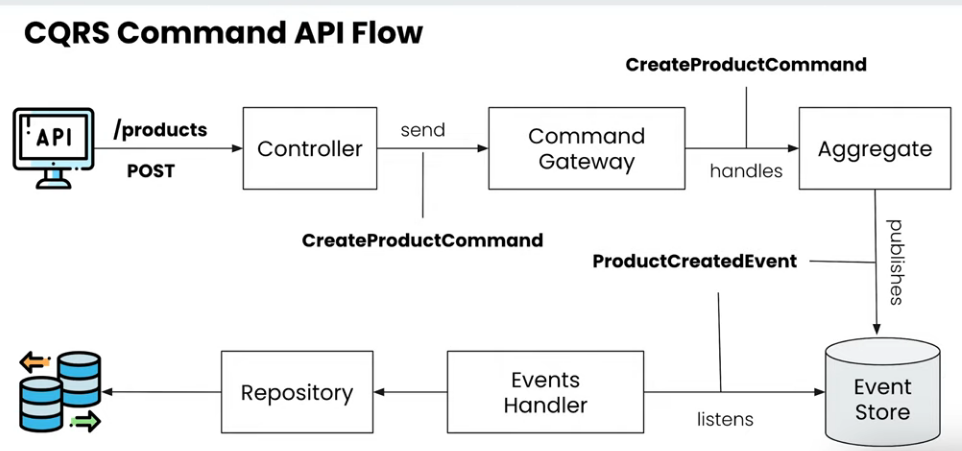
CQRS Desing pattern implementation diagram :



We need to create events to store the data from the write db to read db and make in sync that data



Here we are going to impl the “AXON Server To “ store the all event from the controller



This is the flow we are going to use and flow :