AuthController – Update Authorities Part 1

Add this code below in AuthController

```
@PutMapping(value = "/users/update-authorities/{user_id}", produces =
"application/json", consumes = "application/json")
    @Operation(summary = "Update authorities")
    @ApiResponse(responseCode = "200", description = "Update authorities")
    @ApiResponse(responseCode = "401", description = "Token missing")
    @ApiResponse(responseCode = "403", description = "Token error")
    @SecurityRequirement(name = "studyeasy-demo-api")
    public AccountViewDTO update_auth(@Valid @RequestBody AuthoritiesDTO
authoritiesDTO, @PathVariable long user_id){
        Optional<Account> optionalAccount = accountService.findById(user_id);
        if(optionalAccount.isPresent()){
            Account account = optionalAccount.get();
            account.setAuthorities(authoritiesDTO.getAuthorities());
            accountService.save(account);
            AccountViewDTO accountViewDTO = new AccountViewDTO(account.getId(),
account.getEmail(), account.getAuthorities());
            return accountViewDTO;
        }
        return null;
    }
```

Create a new DTO -> AuthoritiesDTO.java

```
package org.studyeasy.SpringRestdemo.payload.auth;

import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.media.Schema.RequiredMode;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class AuthoritiesDTO {

    @NotBlank
    @Size(min = 6, max = 20)
    @Schema(description = "Authorities", example = "USER", requiredMode =
RequiredMode.REQUIRED)
```

```
    private String authorities;

}
```

Add this in AccountService.java -> below piece of code

```
 public Optional<Account> findById(long id) {
        return accountRepository.findById(id);
    }
```

In SecurityConfig.java

```
package org.studyeasy.SpringRestdemo.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.ProviderManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurit
y;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.security.oauth2.jwt.JwtEncoder;
import org.springframework.security.oauth2.jwt.NimbusJwtDecoder;
import org.springframework.security.oauth2.jwt.NimbusJwtEncoder;
import org.springframework.security.web.SecurityFilterChain;

import com.nimbusds.jose.JOSEException;
import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.JWKSource;
import com.nimbusds.jose.proc.SecurityContext;

@Configuration
@EnableWebSecurity
```

```java
public class SecurityConfig {

    private RSAKey rsaKey;

    @Bean
    public JWKSource<SecurityContext> jwkSource() {
        rsaKey = Jwks.generateRsa();
        JWKSet jwkSet = new JWKSet(rsaKey);
        return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);
    }

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

    // @Bean
    // public InMemoryUserDetailsManager users() {
    // return new InMemoryUserDetailsManager(
    // User.withUsername("chaand")
    // .password("{noop}password")
    // .authorities("read")
    // .build());
    // }

    @Bean
    public AuthenticationManager authManager(UserDetailsService
userDetailsService) {
        var authProvider = new DaoAuthenticationProvider();
        authProvider.setPasswordEncoder(passwordEncoder());
        authProvider.setUserDetailsService(userDetailsService);
        return new ProviderManager(authProvider);
    }

    @Bean
    JwtEncoder jwtEncoder(JWKSource<SecurityContext> jwks) {
        return new NimbusJwtEncoder(jwks);
    }

    @Bean
    JwtDecoder jwtDecoder() throws JOSEException {
        return NimbusJwtDecoder.withPublicKey(rsaKey.toRSAPublicKey()).build();
    }

    @Bean
```

```
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
                // CSRF configuration
                .csrf(csrf -> csrf
                        .disable()) // Disable CSRF for stateless JWT
authentication
                // Frame options for H2 console
                .headers(headers -> headers
                        .frameOptions(frameOptions -> frameOptions.sameOrigin()))
                // Authorization configuration
                .authorizeHttpRequests(authorize -> authorize
                        .requestMatchers("/auth/token").permitAll()
                        .requestMatchers("/auth/users/add").permitAll()
                        .requestMatchers("/auth/users").hasAuthority("SCOPE_ADMIN
")
                        .requestMatchers("/auth/users/update-
authorities/**").hasAuthority("SCOPE_ADMIN")
                        .requestMatchers("/auth/profile").authenticated()
                        .requestMatchers("/auth/profile/update-
password").authenticated()
                        .requestMatchers("/swagger-ui/**").permitAll()
                        .requestMatchers("/v3/api-docs/**").permitAll()
                        .requestMatchers("/test").authenticated() // `/test`
requires authentication
                )
                // JWT-based authentication
                .oauth2ResourceServer(oauth2 -> oauth2
                        .jwt(Customizer.withDefaults()))
                // Stateless session management
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .headers(headers -> headers
                        .frameOptions(frameOptions ->
frameOptions.sameOrigin())); // Required for H2 console

        return http.build();
    }

}
```

----------xxxx----------

AuthController – Update Authorities Part 2

For update_auth method – You should follow this convention

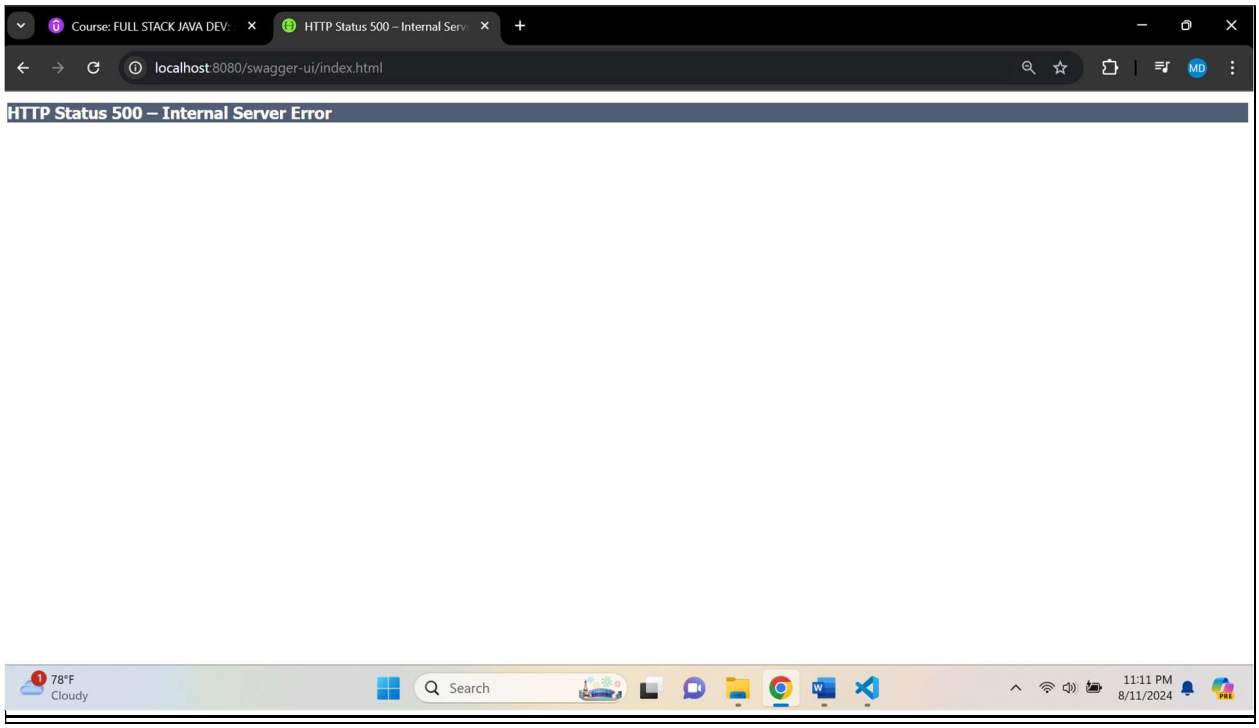
```
@PutMapping(value = "/users/update-authorities/{user_id}", produces =
"application/json", consumes = "application/json")
```

to

```
 @PutMapping(value = "/users/{user_id}/update-authorities", produces =
"application/json", consumes = "application/json")
```

In SecurityConfig.java -> method -> securityFilterChain -> update

```
.requestMatchers("/auth/users/{user_id}/update-
authorities").hasAuthority("SCOPE_ADMIN")
```



Updated SecurityConfig.java

```
package org.studyeasy.SpringRestdemo.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```java
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.ProviderManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurit
y;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.security.oauth2.jwt.JwtEncoder;
import org.springframework.security.oauth2.jwt.NimbusJwtDecoder;
import org.springframework.security.oauth2.jwt.NimbusJwtEncoder;
import org.springframework.security.web.SecurityFilterChain;

import com.nimbusds.jose.JOSEException;
import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.JWKSource;
import com.nimbusds.jose.proc.SecurityContext;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private RSAKey rsaKey;

    @Bean
    public JWKSource<SecurityContext> jwkSource() {
        rsaKey = Jwks.generateRsa();
        JWKSet jwkSet = new JWKSet(rsaKey);
        return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);
    }

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

    // @Bean
    // public InMemoryUserDetailsManager users() {
    // return new InMemoryUserDetailsManager(
```

```
    // User.withUsername("chaand")
    // .password("{noop}password")
    // .authorities("read")
    // .build());
    // }

    @Bean
    public AuthenticationManager authManager(UserDetailsService
userDetailsService) {
        var authProvider = new DaoAuthenticationProvider();
        authProvider.setPasswordEncoder(passwordEncoder());
        authProvider.setUserDetailsService(userDetailsService);
        return new ProviderManager(authProvider);
    }

    @Bean
    JwtEncoder jwtEncoder(JWKSource<SecurityContext> jwks) {
        return new NimbusJwtEncoder(jwks);
    }

    @Bean
    JwtDecoder jwtDecoder() throws JOSEException {
        return NimbusJwtDecoder.withPublicKey(rsaKey.toRSAPublicKey()).build();
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
                // CSRF configuration
                .csrf(csrf -> csrf
                        .disable()) // Disable CSRF for stateless JWT
authentication
                // Frame options for H2 console
                .headers(headers -> headers
                        .frameOptions(frameOptions -> frameOptions.sameOrigin()))
                // Authorization configuration
                .authorizeHttpRequests(authorize -> authorize
                        .requestMatchers("/auth/token").permitAll()
                        .requestMatchers("/auth/users/add").permitAll()
                        .requestMatchers("/auth/users").hasAuthority("SCOPE_ADMIN
")
                        .requestMatchers("/auth/users/{user_id}/update-
authorities").hasAuthority("SCOPE_ADMIN")
                        .requestMatchers("/auth/profile").authenticated()
```

```
                        .requestMatchers("/auth/profile/update-
password").authenticated()
                        .requestMatchers("/swagger-ui/**").permitAll()
                        .requestMatchers("/v3/api-docs/**").permitAll()
                        .requestMatchers("/test").authenticated() // `/test`
requires authentication
                )
                // JWT-based authentication
                .oauth2ResourceServer(oauth2 -> oauth2
                        .jwt(Customizer.withDefaults()))
                // Stateless session management
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .headers(headers -> headers
                        .frameOptions(frameOptions ->
frameOptions.sameOrigin())); // Required for H2 console

        return http.build();
    }

}
```

Update AccountDTO.java

```
package org.studyeasy.SpringRestdemo.payload.auth;

import io.swagger.v3.oas.annotations.media.Schema;
import io.swagger.v3.oas.annotations.media.Schema.RequiredMode;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Setter
@Getter
@AllArgsConstructor
@NoArgsConstructor
public class AccountDTO {

    @Email
```

```
    @Schema(description = "Email Address", example = "admin@studyeasy.org",
requiredMode = RequiredMode.REQUIRED)
    private String email;

    @Size(min = 6, max = 20)
    @Schema(description = "Password", example = "Password", requiredMode =
RequiredMode.REQUIRED, maxLength = 20, minLength = 6)
    private String password;
}
```

Updated AuthController.java

```
package org.studyeasy.SpringRestdemo.controller;

import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import org.studyeasy.SpringRestdemo.model.Account;
import org.studyeasy.SpringRestdemo.payload.auth.AccountDTO;
import org.studyeasy.SpringRestdemo.payload.auth.AccountViewDTO;
import org.studyeasy.SpringRestdemo.payload.auth.AuthoritiesDTO;
import org.studyeasy.SpringRestdemo.payload.auth.PasswordDTO;
import org.studyeasy.SpringRestdemo.payload.auth.ProfileDTO;
import org.studyeasy.SpringRestdemo.payload.auth.TokenDTO;
import org.studyeasy.SpringRestdemo.payload.auth.UserLoginDTO;
```

```java
import org.studyeasy.SpringRestdemo.service.AccountService;
import org.studyeasy.SpringRestdemo.service.TokenService;
import org.studyeasy.SpringRestdemo.util.constants.AccountError;
import org.studyeasy.SpringRestdemo.util.constants.AccountSuccess;

import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.security.SecurityRequirement;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import lombok.extern.slf4j.Slf4j;

@RestController
@RequestMapping("/auth")
@Tag(name = "Auth Controller", description = "Controller for Account management")
@Slf4j
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private TokenService tokenService;

    @Autowired
    private AccountService accountService;

    public AuthController(TokenService tokenService, AuthenticationManager
authenticationManager) {
        this.tokenService = tokenService;
        this.authenticationManager = authenticationManager;
    }

    @PostMapping("/token")
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity<TokenDTO> token(@Valid @RequestBody UserLoginDTO
userLogin) throws AuthenticationException {
        try {
            Authentication authentication = authenticationManager
                    .authenticate(
                            new
UsernamePasswordAuthenticationToken(userLogin.getEmail(),
userLogin.getPassword()));
            return ResponseEntity.ok(new
TokenDTO(tokenService.generateToken(authentication)));
```

```
        } catch (Exception e) {
            log.debug(AccountError.TOKEN_GENERATION_ERROR.toString() + ": " +
e.getMessage());
            return new ResponseEntity<>(new TokenDTO(null),
HttpStatus.BAD_REQUEST);
        }

    }

    @PostMapping(value = "/users/add", produces = "application/json")
    @ResponseStatus(HttpStatus.CREATED)
    @ApiResponse(responseCode = "400", description = "Please enter a valid email
and Password length between 6 to 20 characters")
    @ApiResponse(responseCode = "200", description = "Account Added")
    @Operation(summary = "Add a new user")
    public ResponseEntity<String> addUser(@Valid @RequestBody AccountDTO
accountDTO) {
        try {
            Account account = new Account();
            account.setEmail(accountDTO.getEmail());
            account.setPassword(accountDTO.getPassword());
            accountService.save(account);
            return ResponseEntity.ok(AccountSuccess.ACCOUNT_ADDED.toString());
        } catch (Exception e) {
            log.debug(AccountError.ADD_ACCOUNT_ERROR.toString() + ": " +
e.getMessage());
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(null);
        }
    }

    @GetMapping(value = "/users", produces = "application/json")
    @Operation(summary = "List user api")
    @ApiResponse(responseCode = "200", description = "List of users")
    @ApiResponse(responseCode = "401", description = "Token missing")
    @ApiResponse(responseCode = "403", description = "Token error")
    @SecurityRequirement(name = "studyeasy-demo-api")
    public List<AccountViewDTO> Users() {
        List<AccountViewDTO> accounts = new ArrayList<>();
        for (Account account : accountService.findall()) {
            accounts.add(new AccountViewDTO(account.getId(), account.getEmail(),
account.getAuthorities()));
        }
        return accounts;
    }
```

```
    @GetMapping(value = "/profile", produces = "application/json")
    @Operation(summary = "View Profile")
    @ApiResponse(responseCode = "200", description = "View Profile")
    @ApiResponse(responseCode = "401", description = "Token missing")
    @ApiResponse(responseCode = "403", description = "Token error")
    @SecurityRequirement(name = "studyeasy-demo-api")
    public ProfileDTO profile(Authentication authentication) {
        String email = authentication.getName();
        Optional<Account> optionalAccount = accountService.findByEmail(email);
        Account account = optionalAccount.get();
        ProfileDTO profileDTO = new ProfileDTO(account.getId(),
account.getEmail(), account.getAuthorities());
        return profileDTO;
    }

    @PutMapping(value = "/profile/update-password", produces =
"application/json", consumes = "application/json")
    @Operation(summary = "Update Profile")
    @ApiResponse(responseCode = "200", description = "Update Profile")
    @ApiResponse(responseCode = "401", description = "Token missing")
    @ApiResponse(responseCode = "403", description = "Token error")
    @SecurityRequirement(name = "studyeasy-demo-api")
    public AccountViewDTO update_password(@Valid @RequestBody PasswordDTO
passwordDTO, Authentication authentication) {
        String email = authentication.getName();
        Optional<Account> optionalAccount = accountService.findByEmail(email);
        if (optionalAccount.isPresent()) {
            Account account = optionalAccount.get();
            account.setPassword(passwordDTO.getPassword());
            accountService.save(account);
            AccountViewDTO accountViewDTO = new AccountViewDTO(account.getId(),
account.getEmail(),
                    account.getAuthorities());
            return accountViewDTO;
        }
        return null;
    }

    @PutMapping(value = "/users/{user_id}/update-authorities", produces =
"application/json", consumes = "application/json")
    @Operation(summary = "Update authorities")
    @ApiResponse(responseCode = "200", description = "Update authorities")
    @ApiResponse(responseCode = "401", description = "Token missing")
    @ApiResponse(responseCode = "403", description = "Token error")
    @ApiResponse(responseCode = "400", description = "Invalid user ID")
```

```
    @SecurityRequirement(name = "studyeasy-demo-api")
    public ResponseEntity<AccountViewDTO> update_auth(@Valid @RequestBody
AuthoritiesDTO authoritiesDTO,
            @PathVariable long user_id) {
        Optional<Account> optionalAccount = accountService.findById(user_id);
        Account account = optionalAccount.get();
        account.setAuthorities(authoritiesDTO.getAuthorities());
        accountService.save(account);
        AccountViewDTO accountViewDTO = new AccountViewDTO(account.getId(),
account.getEmail(),
                account.getAuthorities());
        return ResponseEntity.ok(accountViewDTO);
    }
}
```