

Overview of the Full Project Flow

What Happens When a User Logs In and Uses the Application?

- 1 User logs in (POST /login/authenticate) → Gets a JWT token.
- **2** Frontend stores JWT and sends it with every API request.
- Backend validates JWT before allowing access.
- Protected APIs fetch, save, update, or delete data.
- 5 Response is sent back to the frontend for display.

Step 1: User Logs In (POST /login/authenticate)

- What happens?
- The user enters their username and password.
- LoginController.java receives the request.
- Calls UserService.java to validate credentials.

Request from Frontend

```
json Copy 🎾 Edit { "username": "john_doe", "password": "securepassword" }
```

LoginController.java

java

@PostMapping("/authenticate") public ResponseEntity<Object> loginUser(@RequestBody LoginDTO loginDTO) {
 return new ResponseEntity<>(userService.loginUser(loginDTO), HttpStatus.OK); }

Calls UserService.loginUser(loginDTO) to handle login.

UserService.java

```
java

@Override public Object loginUser(LoginDTO loginDTO) { User user =
  userRepo.findByUsername(loginDTO.username()); if (user == null ||
  !passwordEncoder.matches(loginDTO.password(), user.getPassword())) { throw new RuntimeException("Invalid credentials"); } String token = jwtUtils.generateToken(user); return new LoginResponseDTO(token, user.getRole().name()); }
```

- Finds user by username in UserRepo.
- Verifies password using BCryptPasswordEncoder.matches().
- Generates JWT token using JwtUtils.generateToken(user).
- JwtUtils.java Generating JWT

```
public String generateToken(User user) { return Jwts.builder() .subject(user.getUsername()) .issuedAt(new Date(System.currentTimeMillis())) .expiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME_IN_MILLISEC)) .signWith(key) .compact(); }
```

- Creates a JWT token with:
- Username (subject)
- Expiration time
- Signed using a secret key
- Response Sent to Frontend

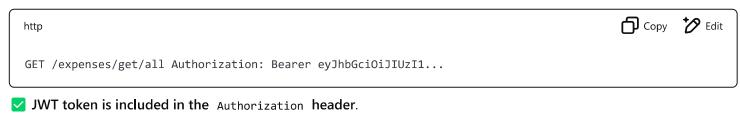
```
json Copy ★ Edit
{ "token": "eyJhbGciOiJIUzI1...", "role": "USER" }
```

Frontend stores this JWT token and includes it in all future API requests.

Step 2: Frontend Makes an API Request with JWT Token

Example: Fetching Expenses (GET /expenses/get/all)

Request from Frontend



- JwtAuthFilter.java Validating JWT
- Pefore Spring Security processes the request, JwtAuthFilter runs.

```
@Override protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException { String token = getTokenFromRequest(request); if (token != null){ String username = jwtUtils.getUsernameFromToken(token); UserDetails userDetails = customUserDetailsService.loadUserByUsername(username); if (jwtUtils.isTokenValid(token, userDetails)){ SecurityContextHolder.getContext().setAuthentication( new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities()) ); } } filterChain.doFilter(request, response); }
```

- Extracts JWT from the request header.
- ✓ Validates token using JwtUtils.
- ✓ If valid, Spring Security allows access to protected APIs.

ExpenseController.java - Fetching Expenses

```
java

@GetMapping("/get/all") public ResponseEntity<List<ExpenseResponseDTO>> getAllExpenses() {
List<ExpenseResponseDTO> expenses = expenseService.getAllExpenses(); return new ResponseEntity<>
(expenses, HttpStatus.OK); }
```

Calls ExpenseService.getAllExpenses() to fetch data.

ExpenseService.java

```
Copy bedit
 java
 @Override public List<ExpenseResponseDTO> getAllExpenses() { return expenseRepo.findAll().stream()
 .map(this::convertToExpenseResponseDTO) .collect(Collectors.toList()); }
Fetches all expenses from the database using ExpenseRepo.findAll().
Converts Expense entity to ExpenseResponseDTO for the frontend.
🔍 ExpenseRepo.java
                                                                                      Copy bedit
 java
 public interface ExpenseRepo extends JpaRepository<Expense, Long> { List<Expense> findByCategoryId(Long
 id); }
Runs SQL query:
                                                                                      Copy bedit
 sql
 SELECT * FROM expenses;
🔍 Response Sent to Frontend
                                                                                      Copy bedit
 json
```

```
json

[ { "id": 1, "description": "Lunch", "amount": 12.50, "date": "2024-02-08", "categoryName": "Food", "userId": 2, "remainingBudget": "87.50" } ]
```

Frontend receives and displays expense data.

Step 3: User Accesses a Protected Admin API (DELETE /users/delete/{id})

Admin tries to delete a user

Request from Frontend



SecurityConfig.java – Restricting Access

```
介 Copy
                                                                                                    * Edit
java
@Bean SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception { return http
.csrf(csrf -> csrf.disable()) .authorizeHttpRequests(authorize -> authorize
.requestMatchers("/login/**", "/users/add").permitAll() .requestMatchers(HttpMethod.DELETE,
"/users/delete/**").hasAuthority("ADMIN") .anyRequest().authenticated()) .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class) .build(); }
```

- Only ADMIN users can delete users.
- If a normal user tries, access is denied (403 Forbidden).

Response Sent to Frontend (If Unauthorized)

```
Copy Bdit
json
{ "error": "Access Denied", "message": "You do not have permission to access this resource" }
```

User is blocked from accessing admin-only routes.

Summary of the Full Project Flow

Step	Process	Components Involved
1 User Logs In	User enters credentials → Backend verifies → JWT	LoginController , UserService , JwtUtils
	token is returned	

		оринд Вост не Едранацон
Step	Process	Components Involved
Frontend Stores JWT	Token is saved in browser local storage & used in API calls	Frontend React App
3 User Makes an API Request	JWT is sent with API request → JwtAuthFilter validates → Backend processes request	JwtAuthFilter, SecurityConfig, ExpenseController, ExpenseService, ExpenseRepo
Protected Admin Route	If user lacks ADMIN role, access is denied (403 Forbidden)	SecurityConfig , JwtAuthFilter