TO DO APP

# Thiết kế cơ sở dữ liệu

## Xác định các thành phần dữ liệu

Todo app được phân quyền truy cập. với những người có quyền admin thì có thể thêm, sửa, xóa, xem danh sách toàn bộ các công việc. những người có quyền user chỉ có thể xem các công việc của mình.

Do đó thiết kế bảng user gồm các trường

* tblUser(id, họ tên, username, password, email, quyền)

1 Todo app cần lưu trữ thông tin về công việc gồm (Tên công việc, mô tả công việc, deadline, mức độ ưu tiên, trạng thái, người tham gia, các thẻ được gắn cho công việc)

* tblToDo(id, title, description, deadline, priorityLevel, status, member, tag)

Để đồng bộ về tên các tags, statuses, prioritylevels, roles nên sẽ có các bảng tương ứng để lưu tên

* tblTags(id, tagName)
* tblStatuses(id, statusName)
* tblprioritylevel(id, prioritylevelName)
* tblroles(id, roleName)

## Xác định mối quan hệ

1 công việc có 1 mức độ ưu tiên, 1 trạng thái tại 1 thời điểm

1 công việc có nhiều người tham gia, có nhều tag

1 người có 1 quyền

Do user và todo, tags và todo là mối quan hệ N-N nên sinh ra các bảng trung gian todo\_tag và todo\_user

A computer screen shot of a computer

Description automatically generated

# Xây dựng ứng dụng

## Application.properties

### st.app.jwtSecret

### st.app.jwtExpirationMs

## enums packages

### EpriorityLevel

### Erole

### Estatus

### ETag

## entities package

### PriorityLevel

### Role

### Status

### Tag

### ToDo

### User

## dto package

## dto.request

### LoginRequest

### SignUpRequest

### ToDoRequest

## dto.response

### JwtResponse

### MessageResponse

## repository package

### PriorityLevelRepository

### RoleRepository

### StatusRepository

### ToDoRepository

### UserRepository

## services package

### ToDoServices

### ToDoServicesImpl

### UserServices

#### UserServices.signIn(LoginRequest loginRequest)

#### UserServices.register(SignupRequest signupRequest)

### UserServicesImpl

## security package

### WebSecurityConfig

#### BEAN authenticationManager(AuthenticationConfiguration authConfig)

#### BEAN passwordEncoder()

#### BEAN authenticationJwtTokenFilter()

Tạo đối tượng AuthTokenFilter mới, kiểm tra các request có chứa token JWT bằng phương thức [doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)](#_doFilterInternal(HttpServletRequest). Nếu có thì trả về 1 Authentication mới dựa trên thông tin trong token Jwt

#### BEAN authenticationProvider()

* Khi 1 object Authentication được tạo ra (thông qua [BEAN authenticationJwtTokenFilter()](#_BEAN_authenticationJwtTokenFilter())). những thông tin này được đóng gói thành một Authentication object
* Khởi tạo 1 Bean DAOAuthenticationProvider, dùng để xác thực username và password bằng cách sử dụng UserDetailsService và PasswordEncoder
* Thiết lập UserDetailsService là một đối tượng [UserDetailsServicesImpl](#_UserDetailsServicesImpl) và dùng nó để tìm [User](#_User) bằng phương thức [UserDetailsServicesImpl.loadUserByUsername(String username)](#_UserDetailsServicesImpl.loadUserByU)
* Thiết lập passwordEncoder để mã hóa và xác thực mật khẩu, sử dụng [BEAN passwordEncoder()](#_BEAN_passwordEncoder())
* Kết quả trả về là đối tượng DaoAuthenticationProvider đã được cấu hình.

#### BEAN WebSecurityConfig.SecurityFilterChain filterChain(HttpSecurity httpSecurity)

* Phương thức này được truyền vào một đối tượng httpSecurity. Đối tượng này được Spring Security tự động khởi tạo khi quét 1 class được gán annotation @EnableWebSecurity. Đối tượng này sẽ được cấu hình các quy tắc xác thực và ủy quyền cho các HTTP Request
* cấu hình csrf
* xử lý ngoại lệ bằng cách sử dụng đối tượng [AuthEntryPointJwt](#_AuthEntryPointJwt). Khi một ngoại lệ xảy ra, phương thức [AuthEntryPointJwt.commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException)](#_AuthEntryPointJwt.commence(HttpServ) đc gọi đến
* Cấu hình session: không tạo session cho các HTTP request
* Cấu hình quy tắc quy cập:
  + với request bắt đầu bằng “/auth/” đều cho phép truy cập mà không cần xác thực
  + với request có “\*/admin/\*\*” thì phải có quyền admin để truy cập
  + các request còn lại đều yêu cầu phải xác thực
* Thêm một [authenticationProvider](#_BEAN_authenticationProvider()) vào httpSecurity để xác thực các đối tượng Authentication
* Thêm filter [authenticationJwtTokenFilter](#_BEAN_authenticationJwtTokenFilter()) trước một UsernamePasswordAuthenticationFilter
* Kết quả trả về là một đối tượng SecurityFilterChain được tạo thông qua httpSecurity.build

## security.jwt package

### AuthEntryPointJwt

#### AuthEntryPointJwt.commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException)

* Khi request k đc xác thưc, Spring Security sẽ tạo một AuthenticationException và gọi đến phương thức này để xử lý. Tại đây sẽ ghi log lỗi và gửi phản hòi với trạng thái 401 (không được xác thực

### AuthTokenFilter

#### AuthTokenFilter.doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

Tại đây, request sẽ được lấy Jwt bằng [AuthTokenFilter.parseJwt(HttpServletRequest request)](#_AuthTokenFilter.parseJwt(HttpServle). Nếu chuỗi token phù hợp khi đi qua phương thức [JwtUtils.validateJwtToken(String authToken)](#_JwtUtils.validateJwtToken(String_au) thì sẽ lấy username bằng phương thức [JwtUtils.getUserNameFromJwtToken(String token)](#_JwtUtils.getUserNameFromJwtToken(St) và sử dụng username này để lấy [User](#_User) để tạo UserDetails thông qua [UserDetailsServicesImpl.loadUserByUsername(String username)](#_UserDetailsServicesImpl.loadUserByU). Tại đây, 1 đối tượng Authentication được tạo ra bằng UserNamePasswordAuthentication

Nếu xảy ra ngoại lệ thì sẽ log lỗi này lại

AuthTokenFilter.parseJwt(HttpServletRequest request)

* Phương thức này lấy Token dựa trên header “Authorization” của request gửi đến.
* Token thuộc loại Baerer Token nên phần header này có dạng “Bearer <tokenString>” nên sẽ dùng phương thức substring để lấy chuỗi token. Nếu không thì trả về null

### JwtUtils

#### JwtUtils.key()

* Giải mã chuỗi jwtSecretTạo một khóa HMAC SHA dựa vào chuỗi jwtSecret, chính là [st.app.jwtSecret](#_st.app.jwtSecret) được khai báo ở [Application.properties](#_Application.properties) từ base 64
* tạo khóa HMAC SHA từ byte được giải mã
* Kết quả trả về là một khóa để ký Jwt

#### JwtUtitls.generateJwtToken(Authentication authentication)

* Tạo một đối tượng [UserDetailsImpl](#_UserDetailsImpl) lấy thông tin người dùng từ đối tượng Authentication được truyền vào. Sử dụng Jwts.builder() để xây dựng một Jwt, đặt username làm subject, đặt thời gian jwt được phát hành, thời gian hết hạn của jwt, sử dụng khóa được tạo bằng phương thức [JwtUtils.key()](#_JwtUtils.key()) và thuật toán HS256 - một thuật toán ký số HMAC để ký jwt. Phần ký số này dùng để xác minh jwt không bị thay đổi sau khi được tạo. Cuối cùng dùng phương thức .compact() để tạo chuỗi jwt dạng String từ JwtBuilder
* Kết quả trả về là 1 chuỗi token

#### JwtUtils.getUserNameFromJwtToken(String token)

được sử dụng để phân tích (parse) một JWT (JSON Web Token) từ một chuỗi token với một khóa đã được cấu hình (key()) và lấy ra thông tin chủ sở hữu (subject) của JWT.

Jwts.parserBuilder(): Phương thức này tạo ra một JwtParserBuilder, đây là một builder được sử dụng để xây dựng một JwtParser. JwtParser là một công cụ được sử dụng để phân tích các JWT 1.

.setSigningKey(key()): Phương thức này đặt khóa được sử dụng để xác minh chữ ký của JWT. Khóa này phải là một khóa hợp lệ cho thuật toán chữ ký được tìm thấy trong tiêu đề JWT (alg header parameter) 1.

.build(): Phương thức này xây dựng một JwtParser từ JwtParserBuilder 1.

.parseClaimsJws(token): Phương thức này sử dụng JwtParser để phân tích JWT từ chuỗi token. Nếu JWT có chữ ký hợp lệ, phương thức này sẽ trả về một Jws object, đại diện cho JWT đã được phân tích. Nếu JWT không có chữ ký hợp lệ, phương thức này sẽ ném ra một JwtException 1.

.getBody().getSubject(): Phương thức này lấy ra thông tin chủ sở hữu (subject) từ JWT đã được phân tích. Thông tin này thường được sử dụng để xác định người dùng đã được xác thực 5.

#### JwtUtils.validateJwtToken(String authToken)

được sử dụng để xác minh chữ ký của một JWT phân tích (parse) một JWT (JSON Web Token) từ một chuỗi authToken với một khóa đã được cấu hình (key()).

Jwts.parserBuilder(): Phương thức này tạo ra một JwtParserBuilder, đây là một builder được sử dụng để xây dựng một JwtParser. JwtParser là một công cụ được sử dụng để phân tích các JWT 4.

.setSigningKey(key()): Phương thức này đặt khóa được sử dụng để xác minh chữ ký của JWT. Khóa này phải là một khóa hợp lệ cho thuật toán chữ ký được tìm thấy trong tiêu đề JWT (alg header parameter) 4.

.build(): Phương thức này xây dựng một JwtParser từ JwtParserBuilder 4.

.parse(authToken): Phương thức này sử dụng JwtParser để phân tích JWT từ chuỗi authToken. Nếu JWT có chữ ký hợp lệ, phương thức này sẽ trả về một Jws object, đại diện cho JWT đã được phân tích. Nếu JWT không có chữ ký hợp lệ, phương thức này sẽ ném ra một JwtException 4.

## security.services

### UserDetailsImpl

#### UserDetailsImpl.build(User user)

### UserDetailsServicesImpl

#### UserDetailsServicesImpl.loadUserByUsername(String username)

* Sử dụng [UserRepository](#_UserRepository) để lấy về user bằng cách truyền vào username. Nếu không tìm thấy thì ném ra 1 ngoại lệ UsernameNotFoundException
* Kết quả về 1 đối tượng [UserDetailsImpl](#_UserDetailsImpl) bằng cách gọi phương thức [UserDetailsImpl.build(User user)](#_UserDetailsImpl.build(User_user))

## Controllers

### AuthControllers

#### userServices

#### AuthControllers. registerUser(SignupRequest signupRequest)

#### AuthControllers.authenticateUser(LoginRequest loginRequest)

### TestControllers

## Chức năng

### Đăng ký user

* Quyền truy cập: ALL
* Sử dụng phương thức HTTP POST
* đối tượng [signupRequest](#_SignUpRequest) được truyền vào thông qua @RequestBody
* Thông qua PostMapping, request này sẽ được phương thức [AuthControllers. registerUser(SignupRequest signupRequest)](#_AuthControllers._registerUser(Signu) tiếp nhận
* tại đây, thông qua [userServices](#_private_UserServices_userServices;), gọi phương thức [UserServices.register(SignupRequest signupRequest)](#_UserServices.register(SignupRequest). Phương thức này được thực thi thông qua [UserServicesImpl](#_UserServicesImpl)
* Trong phương thức này, đặt ra 1 chuỗi regex để kiểm tra mật khẩu (tối thiểu 8 ký tự, bao gồm chữ hoa, chữ thường, số và ký tự đặc biệt), kiểm tra họ tên có trống hay không, dùng [UserRepository](#_UserRepository) để kiểm tra username, email đã tồn tại hay chưa. Nếu không vi phạm thì tạo 1 đối tượng [User](#_User) với các thuộc tính fullName, userName, email, và mật khẩu được mã hóa bằng đối tượng [PasswordEncoder](#_PasswordEncoder) được khởi tạo là @Bean tại class [WebSecurityConfig](#_WebSecurityConfig) sử dụng hàm băm BCrypt. Thuộc tính [Role](#_Role) của đối tượng [User](#_User) sẽ được gán là ROLE\_USER nếu để trống. Đối tượng User sẽ được lưu lại thông qua phương thức save của [UserRepository](#_UserRepository). Kết quả trả về một ResponseEntity với HttpStatusCode và body là một [MessageResponse](#_MessageResponse) thông báo thực hiện thành công hoặc vi phạm điều kiện trong bước kiểm tra

### Đăng nhập

* Quyền truy cập: ALL
* Sử dụng phương thức HTTP POST
* Đối tượng [SignUpRequest](#_SignUpRequest) được truyền vào thông qua @RequestBody
* Thông qua PostMapping, request này sẽ được phương thức [AuthControllers. authenticateUser (LoginRequest loginRequest)](#_AuthControllers._registerUser(Signu) tiếp nhận
* tại đây, thông qua [userServices](#_private_UserServices_userServices;), gọi phương thức [UserServices. signIn(LoginRequest loginRequest)](#_UserServices.register(SignupRequest). Phương thức này được thực thi thông qua [UserServicesImpl](#_UserServicesImpl)
* Trong phương thức này,
  + Khởi tạo 1 đối tượng Authentication chứa thông tin người dùng đã được xác thực. dùng kết quả trả về của phương thức authenticate của Bean [AuthenticationManager](#_AuthenticationManager) được định nghĩa tại [WebSecurityConfig](#_WebSecurityConfig). Đối tượng được truyền vào phương thức authenticate là một đối tượng UsernamePasswordAuthenticationToken mới với tên người dùng và mật khẩu từ yêu cầu đăng nhập. Class này đại diện cho một yêu cầu xác thực đơn giản dựa trên tên người dùng và mật khẩu
  + Thiết lập đối tượng Authentication này trong SecurityContext - một đối tượng giữ thông tin về người dùng hiện tại đang tương tác với hệ thống - thông qua SecurityContextHolder - một lớp utililty trong Spring Security cung cấp quyền truy cập đến SecurityContext
  + Tạo token thông qua [JwtUtitls.generateJwtToken(Authentication authentication)](#_JwtUtitls.generateJwtToken(authenti) với đối tượng truyền vào là authentication đã tạo ở trên
  + Tạo đối tượng [UserDetailsImpl](#_UserDetailsImpl) được gán là principal của đối tượng authentication trên. Principal này là đối tượng đại diện cho người dùng hiện tại
  + Lấy role dạng String từ đối tượng userDetails bên trên
  + Kết quả trả về là một ResponseEntity với phần thân là một [JwtResponse](#_JwtResponse)

### Xác thực và phân quyền

Mỗi request được gửi đến sẽ đi qua các filter trong chuỗi filter của SpringSecurity. Request này đi qua filter được cấu hình trong [WebSecurityConfig](#_WebSecurityConfig)

### Thêm mới ToDo

* Quyền truy cập: ADMIN
* Sử dụng phương thức HTTP POST
* Annotation Authorize cho biết người dùng có quyền nào thì được truy cập đến phương thức
* gọi đến phương thức addNewToDo(toDoRequest)

### Sửa ToDo

* Quyền truy cập: ADMIN
* Sử dụng phương thức HTTP PUT
* Annotation Authorize cho biết người dùng có quyền nào thì được truy cập đến phương thức

### Xóa ToDo

* Quyền truy cập: ADMIN
* Sử dụng phương thức HTTP DELETE
* Annotation Authorize cho biết người dùng có quyền nào thì được truy cập đến phương thức

### Lấy danh sách tất cả ToDo

* Quyền truy cập: ADMIN
* Sử dụng phương thức HTTP GET
* Annotation Authorize cho biết người dùng có quyền nào thì được truy cập đến phương thức

### Lấy danh sách ToDo theo userId

* Quyền truy cập: ADMIN, USER
* Sử dụng phương thức HTTP GET
* Annotation Authorize cho biết người dùng có quyền nào thì được truy cập đến phương thức

## Phân quyền

* Mỗi Request sẽ phải đi qua SecurityFilterChain đc cấu hình ở [WebSecurityConfig.SecurityFilterChain filterChain(HttpSecurity httpSecurity)](#_WebSecurityConfig.SecurityFilterCha)
* Theo như cấu hình này, request sẽ đi qua bộ lọc [BEAN authenticationJwtTokenFilter()](#_BEAN_authenticationJwtTokenFilter()) để kiểm tra xem request đó có token hay không. Nếu có thì tạo 1 đối tượng Authentication
* Sau khi đi qua [BEAN authenticationJwtTokenFilter()](#_BEAN_authenticationJwtTokenFilter()), dựa theo url, lọc xem request cần có quyền nào thông qua các requestMatchers tương ứng. Nếu có quyền tương ứng sẽ cho phép request đi đến tầng controller