TODO Manager w/ Security

TODO Data Model

```
1. text         - String
2. authorId     - int
3. isPublic     - boolean
4. createDate   - LocalDate
```

Roles

```
1. Guest/Anonymous Role
2. Author/Standard User Role
3. Administrator Role
```

User Stories

```
As a __________, I should [not] be able to _____________.

Preconditions: what must be true for the user story to be relevant.
Postconditions: what must be true after the user story ends.
```

- [ ] As any user, I should be able to see all public Todos.
- [ ] As a guest, I should not be able to see any private Todos.
- [ ] As a guest, I should not be able to create a Todo.
- [ ] As a guest, I should not be able to remove a Todo.
- [ ] As a guest, I should not be able to edit a Todo.
- [ ] As a guest, I should be able to create an account.
- [ ] As a guest, I should be able to log into an existing account.
- [ ] As an Author, I should be able to see *my own* private Todos.
- [ ] As an Author, I should not be able to see other Author's Todos.
- [ ] As an Author, I should be able to create a Todo.
- [ ] As an Author, I should be able to remove *my own* Todos.
- [ ] As an Author, I should not be able to remove other Author's Todos.
- [ ] As an Author, I should be able to edit *my own* Todos.
- [ ] As an Author, I should not be able to edit other Author's Todos.
- [ ] As an Admin, I should be able to see all Todos.
- [ ] As an Admin, I should be able to create a Todo.
- [ ] As an Admin, I should be able to remove all Todos.
- [ ] As an Admin, I should be able to edit all Todos.
- [ ] As an Admin, I should be able to promote an Author to Admin.
- [ ] As an Admin, I should be able to remove Author users.

- [ ] As an Admin, I should be able to edit the name of Authors.
- [ ] As an Admin, I should be able to change the password of Authors.

Tasks

- [ ] Create Java API
  - [x] Create Java Project (todo-with-security)
  - [x] Modify pom.xml to include the parent tag (spring-boot-starter-parent)
  - [x] Modify pom.xml to include the following dependencies
    - [x] spring-boot-starter-security
    - [x] jjwt-api
    - [x] jjwt-impl
    - [x] jjwt-jackson
    - [x] mysql-connector-java
    - [x] spring-boot-starter-jdbc
    - [x] spring-boot-starter-web
  - [x] Create base package (todo)
    - [x] Create App class
      - [x] @SpringBootApplication
      - [x] main
        - [x] SpringApplication.run(App.class, args);
  - [x] Create application.properties file
    - [x] spring.datasource.url=jdbc:mysql://localhost:3306/todo_prod
    - [x] spring.datasource.username=root
    - [x] spring.datasource.password=top-secret-password
  - [x] Create models package
    - [x] Create AppUser class
      - [x] Extend from the User (org.springframework.security.core.userdetails)
      - [x] Add Set<String> roles field variable
      - [x] Add Integer userId field variable
      - [x] Generate getters/setters
      - [x] Generate hashCode/equals
      - [x] Add constructor which takes Integer userId, String username, String password, and Set<String> roles
        - [x] call super(username, password, roles.stream().map(r -> new SimpleGrantedAuthority("ROLE_" + r)).collect(Collectors.toList()))
        - [x] assign to this.userId
        - [x] assign to this.roles
    - [x] Create Todo class
      - [x] Create Integer todoId field variable
      - [x] Create String text field variable
      - [x] Create Integer userId field variable
      - [x] Create Boolean isPublic field variable
      - [x] Create LocalDate createDate field variable
      - [x] Generate getters/setters
      - [x] Generate hashCode/equals

- [ ] Create data package
  - [x] Create TodoRepo interface
    - [x] List<Todo> findAllPublic()
    - [x] List<Todo> findByUserId(Integer userId)
    - [x] Todo findById(Integer todoId)
    - [x] Todo add(Todo toAdd)
    - [x] boolean remove(Integer todoId)
    - [x] void edit(Todo updated)
  - [x] Create UserRepo interface
    - [x] AppUser findByUsername(String username)
    - [x] AppUser add(AppUser toAdd)
    - [x] boolean remove(Integer userId)
    - [x] void edit(User updated)
  - [x] Create TodoMapper class
    - [x] implements RowMapper<Todo>
    - [x] Generate interface method
      - [x] Todo toReturn = new Todo();
      - [x] toReturn.setTodoId(rs.getInt("todoId"));
      - [x] toReturn.setText(rs.getString("todoText"));
      - [x] toReturn.setUserId(rs.getInt("authorId"));
      - [x] toReturn.setPublic(rs.getBoolean("isPublic"));
      - [x] toReturn.setCreateDate(LocalDate.parse(rs.getString("createDate")));
      - [x] return toReturn;
  - [x] Create TodoDbRepo class
    - [x] Add @Repository
    - [x] add @Autowired JdbcTemplate template field variable
    - [x] implements TodoRepo
      - [x] generate functions automatically
      - [x] implement findAllPublic()
        - [x] String sql = "SELECT * FROM todos where isPublic = 1;"
        - [x] return template.query(sql, new TodoMapper());
      - [ ] implement findById()
        - [ ] String sql = return template.query("select * from todos where todoId = ?", new TodoMapper(), todoId).stream().findAny().orElse(null);
  - [x] Create UserMapper class
    - [x] create Set<String> roles field variable
    - [x] create UserMapper constructor which takes in the Set of roles and sets the field variable
    - [x] implements RowMapper<AppUser>
    - [x] auto-generate methods
      - [x] AppUser toBuild = new AppUser(userId, username, password, roles);
  - [x] Create UserDbRepo class
    - [x] Add @Repository
    - [x] implements UserRepo
      - [x] Add @Autowired JdbcTemplate template field variable
      - [x] generate functions automatically

- [x] create private Set<String> findRolesByUsername(String username)
  - [x] String sql = "SELECT roleName FROM users u inner join userroles ur on ur.userId = u.userId inner join roles r on ur.roleId = r.roleId where username = ?"
  - [x] return template.query(sql, (rowData, rowNum)->rowData.getString("roleName"), username).stream().collect(Collectors.toSet())
- [x] implement findByUsername(String username)
  - [x] String sql = "select userId, username, password from users where username = ?"
  - [x] return template.query(sql, new UserMapper(findRolesByUsername(username)), username).stream().findAny().orElse(null);

- [ ] Create domain package
  - [ ] Create InvalidUserException
    - [ ] create constructor that takes in String message, call super(message)
    - [ ] create constructor that takes in String message, Throwable innerException calls super(message, innerException)
  - [ ] Create UserService class
    - [x] mark with @Service
    - [x] implements UserDetailsService
    - [x] add UserRepo field variable
    - [x] add PasswordEncoder field variable
    - [x] add constructor which takes in a UserRepo & PasswordEncoder
    - [x] @Override loadUserByUsername (can return AppUser as a UserDetails object)
      - [x] use the repo to pass along the user
      - [x] add //TODO: validate (later we'll check to make sure username isn't null/empty/etc)
      - [x] if user is not found (we get a null) throw new UsernameNotFoundException(username + " not found")
      - [x] otherwise, return the user
    - [x] add AppUser create(String username, String password)
      - [x] for now just return null
  - [x] Create TodoService class
    - [x] mark as @Service
    - [ ] add @Autowired TodoRepo tRepo field variable
    - [ ] add @Autowired UserRepo uRepo field variable
    - [x] -- should have autogenerated getPublicTodos method from controller --
      - [x] return repo.findAllPublic();
    - [ ] create public void deleteById(Integer todoId, Principal user) throws InvalidUserException {
      - [ ] Todo toDelete = tRepo.findById(todoId);
      - [ ] AppUser requester = uRepo.findByUsername(user.getName());
      - [ ] if(requester.getRoles().contains("ADMIN") || requester.getUserId().intValue() == toDelete.getUserId().intValue()){
        - [ ] tRepo.remove(todoId);
      - [ ] } else { throw new InvalidUserException("Only admins and the author of the todo may delete it."); }
- [ ] Create security package
  - [ ] create SecurityConfig class
    - [x] @EnableWebSecurity
    - [x] extends WebSecurityConfigurerAdapter
    - [x] @Override protected void configure(HttpSecurity http) throws Exception

- [x] http.csrf().disable()
- [x] http.cors()
- [x] http.authorizeRequests()
  - [x] .antMatchers(HttpMethod.POST, "/api/security/login").permitAll()
  - [x] .antMatchers(HttpMethod.GET, "/api/todo/public").permitAll()
  - [ ] .antMatchers(HttpMethod.DELETE, "/api/todo/*").hasAnyRole("AUTHOR", "ADMIN")
  - [x] .antMatchers("/**").denyAll()
  - [x] .and()
  - [x] .sessionManagement()
    - [x] .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
- [x] public PasswordEncoder getEncoder(){ return new BCryptPasswordEncoder(); }
  - [x] mark with @Bean
- [x] @Override protected AuthenticationManager authenticationManager() throws Exception
  - [x] just return super.authenticationManager();
  - [x] mark with @Bean

- [x] Create JwtConverter class
  - [x] Mark as @Component
  - [x] add a Key field variable (secretKey) assign Keys.secretKeyFor(SignatureAlgorithm.HS256)
  - [x] add public String getTokenFromUser(User toConvert)
    - [x] generate comma separated string of authorities granted to the user (retrieve those with .getAuthorities())
    - [x] return Jwts.builder()
      - [x] .setIssuer("todo-app")
      - [x] .setSubject(toConvert.getUsername())
      - [x] .claim("authorties", commaSeparatedString)
      - [x] .setExpiration(new Date(System.currentTimeMillis() + 15 * 60 * 1000))
      - [x] .signWithKey(secretKey)
      - [x] .compact();
  - [x] add public User getUserFromToken(String token)
    - [x] try/catch (JwtException)
      - [x] JwtParser parser = Jwts.parserBuilder().requireIssuer("todo-app").setSigningKey(secretKey).build();
      - [x] Jws<Claims> claims = parser.parseClaimsJws(token.substring(7));
      - [x] String username = claims.getBody().getSubject();
      - [x] String authorities = (String)claims.getBody().get("authorities");
      - [x] String [] authSplit = authorities.split(",");
      - [x] List<GrantedAuthority> grantedAuthorities = new ArrayList<>();
      - [x] for(String auth : authSplit){ grantedAuthorities.add(new SimpleGrantedAuthority(auth)); }
      - [x] return new User(username, username, grantedAuthorities);
      - [x] catch(JwtException ex) {
        - [x] ex.printStackTrace(System.err);
        - [x] return null; }
- [ ] Create JwtRequestFilter class
  - [x] extends BasicAuthenticationFilter
  - [x] Add a JwtConverter field

- [x] Add a constructor that takes in a JwtConvert and AuthenticationManager
  - [x] super(authManager)
  - [x] store the JwtConverter in the field variable
- [x] @Override protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
  - [x] String authHeader = request.getHeader("Authorization ");
  - [x] if(authHeader != null && authHeader.startsWith("Bearer ")){
    - [x] User converted = converter.getUserFromToken(authHeader);
    - [x] if(converted != null){
      - [x] UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(converted.getUsername(), null, convertedUser.getAuthorities());
      - [x] SecurityContextHolder.getContext().setAuthentication(token);
    - [x] } else {
      - [x] response.setStatus(403); }
  - [x] chain.doFilter(request, response);
- [x] IN SecurityConfig.java
  - [x] add @Autowired JwtConverter field variable
  - [x] right after the .and() call .addFilter(new JwtReqestFilter())

- [ ] Create controllers package
  - [ ] Add AuthController class
    - [x] mark as @RestController
    - [x] add @RequestMapping("/api/security")
    - [x] add AuthenticationManager field variable
    - [x] add JwtConverter field variable
    - [x] add UserService field variable
    - [x] add a constructor that takes in all field variables and sets them
    - [x] add ResponseEntity login(@RequestBody Map<String,String> credentials)
      - [x] mark as @PostMapping("/login")
      - [x] create UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(credentials.get("username"), credentials.get("password"));
      - [x] in a try/catch(AuthenticationException ex) block...
        - [x] Authentication authResult = authManager.authenticate(token);
        - [x] if(authResult.isAuthenticated()){
          - [x] String jwt = converter.getTokenFromUser((User)authResult.getPrincipal());
          - [x] Map<String,String> tokenWrapper = new HashMap<>();
          - [x] tokenWrapper.put("jwt_token", jwt);
          - [x] return ResponseEntity.ok(tokenWrapper);
        - [x] }
        - [x] catch(AuthenticationException ex){
          - [x] ex.printStackTrace(System.err); }
        - [x] return new ResponseEntity(HttpStatus.FORBIDDEN);
  - [x] Add TodoController class
    - [x] mark as @RestController

- [x] @RequestMapping("/api/todo")
- [x] add @Autowired TodoService field variable (service)
- [x] add a GET endpoint ("/public") for retrieving all todos
  - [x] List<Todo> pubTodos = service.getPublicTodos() (doesn't exist yet...)
  - [x] generate TodoService.getPublicTodos()
  - [x] return ResponseEntity.ok(pubTodos);
- [x] add a DELETE endpoint ("/{todoId}")
  - [x] public ResponseEntity delete(@PathVariable Integer todoId, Principal user){
    - [x] service.deleteById(todoId, user);
    - [x] generate TodoService.deleteById()
    - [x] return ResponseEntity.ok().build();

- [ ] Create mysql schemas (test/prod)
  - [x] create sql folder in project folder
  - [x] create todo-test.sql
  - [x] create todo-prod.sql
  - [x] drop database if exists todo_X
  - [x] create database todo_X
  - [x] use todo_X
  - [x] create table users
    - [x] userId int primary key auto_increment
    - [x] username varchar(300) not null unique
    - [x] password varchar(2048) not null,
  - [x] create table todos
    - [x] todoId int primary key auto_increment
    - [x] todoText text not null
    - [x] authorId int not null
    - [x] isPublic bit(1) not null
    - [x] createDate date not null
    - [x] constraint fk_todos_users foreign key (authorId) references users(userId)
  - [x] create table roles
    - [x] roleId int primay key auto_increment
    - [x] roleName varchar(20) not null unique
  - [x] create table userroles
    - [x] userId int not null,
    - [x] roleId int not null,
    - [x] constraint pk_userroles (userId, roleId),
    - [x] constraint fk_users_userroles foreign key (userId) references users(userId)
    - [x] constraint fk_roles_userroles foreign key (roleId) references roles(roleId)
  - [x] insert into users (username, password) values ('bob', '$2a$12$HqaU3VlN09ufZ60R8VrLHuIX8H6b1iFDA9AG./vzThpIzhxEIF8nC'); -- pw is password
  - [x] insert into users (username, password) values ('june', '$2a$12$k2TB.cQ1TLHLOYn.pbbiTuQ5HoUxozWkl.ZgFZ.9eioAeMxndT5AS'); -- pw is admin-password
  - [x] insert into roles (roleName) VALUES ('AUTHOR'), ('ADMIN');
  - [x] insert into userroles (userId, roleId) VALUES (1,1), (2,2);

- [x] insert into todos (todoText, authorId, isPublic, createDate) values ('this is a private todo', 1, 0, '2020-04-06'), ('this is a public todo', 2, 1, '2020-04-05');
- [x] generate reset stored procedure in db (set_known_good_state)
  - [x] delete from userroles;
  - [x] delete from users;
  - [x] alter table users auto_increment = 1;
  - [x] delete from roles;
  - [x] alter table roles auto_increment = 1;
  - [x] delete from todos;
  - [x] alter table todos auto_increment = 1;
  - [x] (copy all inserts from prod)
- [x] at end of test schema call set_known_good_state();

- [ ] Create React Front-End
  - [x] From the terminal, inside of your Java application
    - [x] `npx create-react-app client`
    - [x] `cd client`
    - [x] `code .` [optional - open in VSCode]
  - [x] Delete cruft
    - [x] ./public/favicon.ico
    - [x] ./public/logo192.png
    - [x] ./public/logo512.png
    - [x] ./public/manifest.json
    - [x] ./public/robots.txt
    - [x] ./src/App.css
    - [x] ./src/App.test.js
    - [x] ./src/logo.svg
    - [x] ./src/reportWebVitals.js
    - [x] ./src/setupTests.js
    - [x] Update ./public/index.html
      - [x] From default file, delete:
        - [x] Lines 4-26
        - [x] Change Title to `Todo App`
        - [x] Delete any additional comments here
    - [x] Update ./src/App.js
      - [x] From default file, delete:
        - [x] Lines 7-20
        - [x] Lines 1-2
    - [x] Update ./src/index.css\
      - [x] Trashcan it all
    - [x] Update ./src/index.js
    - [x] From default file, delete:
      - [x] Lines 14-17
      - [x] Lines 5
  - [x] Add additional dependencies
    - [x] `npm i react-router-dom`

- [x] Create components (indents below indicate parent-child relations)
  - [x] Nav Component
  - [x] Login Component
  - [x] Home Component - welcoming and showing all pub todos
    - [x] Welcome Component - nested inside Home
    - [x] Todos (container) Component
      - [x] Todo Component
      - [x] Delete Component
  - [x] AddTodo Component
- [x] Add react-router to our project
  - [x] At the top of index.js
    - [x] `import { BrowserRouter } from 'react-router-dom';`
    - [x] Change `<React.StrictMode>` to `<BrowserRouter>`
    - [x] Change `</React.StrictMode>` to `</BrowserRouter>`
- [x] Build out base Home component
  - [x] Functional component, don't forget to export!
- [x] Build out base Welcome component
  - [x] Functional component, don't forget to export!
- [x] Add `<Home />` to App.js
  - [x] `import Home from "./Home";`
  - [x] Add flavor-text to ground ourselves
- [x] Add `<Welcome />` to Home.js
  - [x] `import Welcome from "./Welcome";`
  - [x] Add flavor-text to ground ourselves
- [x] Add `<Nav />` to App.js
  - [x] `import Nav from "./Nav";`
  - [x] Add flavor-text to ground ourselves
- [x] Begin implementing Routes in App.js
  - [x] `import { Routes, Route } from 'react-router-dom';`
  - [x] `<Routes>`
    - [x] `<Route path="/" element={<Home />} />`
    - [x] `// ^^ Home Page Route, at base dot-com URL`
  - [x] `</Routes>`
- [x] Begin implementing Links in Nav.js
  - [x] `import { Link } from 'react-router-dom';`
  - [x] `<Link to="/">Home</Link>`
- [x] Add `<Todos />` component to Home.js
  - [x] `import Todos from "./Todos";`
- [ ] In Todos.js...
  - [x] Create State to store public todos
    - [x] `import { useState } from 'react';`
      - [x] `const [pubTodos, setPubTodos] = useState([]);`
  - [x] Implement `useEffect()` hook for setting state on fetch
    - [x] `import { useState, useEffect } from 'react';`
    - [x] `useEffect(() => {`

- [x] Use Fetch API to retrieve our public todos
  - [x] Verify CORS is open in your TodoController (Java)
    - [x] `@CrossOrigin(origins = {"http://localhost:3000"})`
  - [x] `fetch("http://localhost:8080/api/todo/public")`
  - [x] `.then(response => {`
    - [x] `if (response.status === 200) {`
      - [x] `return response.json()`
    - [x] `} else {`
      - [x] `alert("Something went wrong when fetching")`
    - [x] `}`
  - [x] `})`
  - [x] `.then(todosData => setPubTodos(todosData))`
  - [x] `.catch(rejection => alert("Failure: " + rejection.status))`
- [x] `}, [])`
- [x] `import Todo from './Todo'`
- [x] Implement a `<Todo />` factory function
  - [x] `function todoFactory() {`
    - [x] `return pubTodos.map(todo => <Todo key={todo.todoId} todoObj={todo} />);`
  - [x] `}`
  - [x] Call function inside of the return for `<Todos />`
    - [x] `return (`
      - [x] `<>`
        - [x] `{todoFactory()}`
      - [x] `</>`
    - [x] `)`
- [x] Build out the base `<Todo />` component
  - [x] Functional component, don't forget to export
- [x] Use `props.todoObj` to access the todo and display in Todo.js
  - [x] Destructure the properties of my todoObj into variables
    - [x] const { text, userId, createDate } = props.todoObj;
    - [x] Build HTML/JSX structure to display data to return
      - [x] `<div className="todo-item">`
        - [x] `<h3>User Id: {userId}</h3>`
        - [x] `<p>Created: {createDate}</p>`
        - [x] `<p>Text: {text}</p>`
      - [x] `</div>`
- [x] Update index.css with Dev-CSS to help visualize
  - [x] `.todo-item {`
    - [x] `border: 1px black solid;`
    - [x] `padding: 20px;`
    - [x] `margin-bottom: 30px;`
  - [x] `}`

- [x] Implement useContext hook
  - [x] Create AuthContext.js
    - [x] import { createContext } from 'react';

- [x] const AuthContext = createContext();
- [x] export default AuthContext

- [x] In App.js, implement Context
  - [x] `import { useState } from 'react';`
  - [x] `import AuthContext from "./AuthContext";`
  - [x] `const [user, setUser] = useState(null);`
  - [x] Inside of the return
    - [x] Before rendering any other components, encapusulate with:
      - [x] `<AuthContext.Provider value={[user, setUser]}>`
        - [x] `(everything else you already had here)`
      - [x] `</AuthContext.Provider>`

- [ ] Build out Login component
  - [ ] In terminal: `npm i jwt-decode`
  - [ ] `import { useState, useContext } from "react";`
  - [ ] `import { useNavigate } from "react-router-dom";`
  - [ ] `import jwtDecode from "jwt-decode";`
  - [ ] `import AuthContext from "./AuthContext";`
  - [ ] const [username, setUsername] = useState("");
  - [ ] const [password, setPassword] = useState("");
  - [ ] const [user, setUser] = useContext(AuthContext);
  - [ ] const navigate = useNavigate();
  - [ ] Build out a form for logging in
    - [ ] `<form onSubmit={submitHandler}>`
      - [ ] `<label>Username:</label><br />`
      - [ ] `<input onChange={event => setUsername(event.target.value)}></input><br /><br />`
      - [ ] `<label>Password:</label><br />`
      - [ ] `<input type="password" onChange={event => setPassword(event.target.value)}></input><br /><br />`
      - [ ] `<button>Submit</button>`
    - [ ] `</form>`
  - Create submit handler for form
    - [ ] `function submitHandler(event) {`
      - [ ] `event.preventDefault()`
      - [ ] `fetch("http://localhost:8080/api/security/login", {`
        - [ ] `method: "POST",`
        - [ ] `headers: {`
          - [ ] `"Content-Type": "application/json"`
        - [ ] `},`
        - [ ] `body: JSON.stringify({`
          - [ ] `username, password`
        - [ ] `})`
      - [ ] `})`
      - [ ] `.then(response => {`
        - [ ] `if (response.status === 200) {`
          - [ ] `const { jwt_token } = await response.json()`

```
            localStorage.setItem("token", jwt_token)
            setUser({user: jwtDecode(jwt_token)})
            navigate("/")
        } else {
            alert("Something bad");
        }
    })
    .catch(rejection => alert(rejection))
}
```