|  |
| --- |
| Deloitte |
| TODO App |
| Description |
|  |
| Version 1.0 |
|  |
| **Ion Iascerinschi** |
| **14-Aug-2019** |

|  |
| --- |
| Description for “TODO” application which was developed during 13 Aug 2019. |

Spring MVC was used for this project because it is easy to maintain this pattern (Model, View, Controller).

View (WEB) being the .jsp pages which can be dynamic because of Java, JSTL, JS. Models are mapped to in memory DB (HSQLDB) using Hibernate as ORM. And Controller module handles all HTPP requests (login, register, add-todo, remove-todo, update-todo).

In my opinion if the project would have a longer lifespan the best approach would be to use React or Angular because there are already a lot of WEB components like Angular Material which helps representing data in WEB format. For backend Spring Boot (RESTful or GraphQL) is the way to go. And in case if needed backend could be connected to mobile applications (IOS, Android, HarmonyOS) using Rest API calls with JSON objects.

For passwords, I used POST method on HTTP req. and for other stuff I used GET method.

On registration all paswords are hashed using BCRYPT.

Mocked users are - login: “test”, password: “pwd123” and login: “user”, password: “123pwd”.

In the project I added also Spring Security for user session.

http.authorizeRequests()  
 .antMatchers("/login").permitAll()  
 .antMatchers("/todo/show-todos").access("hasRole('ROLE\_USER')")  
 .and().formLogin().loginPage("/login")  
 .defaultSuccessUrl("/todo/show-todos")  
 .failureUrl("/error")  
 .usernameParameter("username").passwordParameter("password")  
 .and().csrf().disable();

For some inserts on application startup import.sql was used.

Log4j also is implemented for debug purpose (to see HQL).

**Design patterns:** MVC,

IoC using @Autowired,

@Autowired  
@Qualifier("userDetailsService")  
private UserDetailsService userDetailsService;

Factory Pattern (Application Context),

<**form:form** method="POST" action="${pageContext.servletContext.contextPath}/login"

Singleton.

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

**What can be improved:**

Of course UI, Unit Tests (JUnit + Mockito), Automation Tests (Selenium), a lot of code can be cleaned to be readable for future developers in the team.