

## Criterion C – Development

### Used systems for development

OS: Microsoft Windows 10 Education

JDK version: 11

IDE: IntelliJ IDEA 2019.1

### Summary:

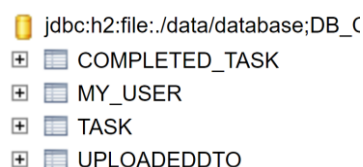
1. Use of H2 Database and JPA Repositories
2. Security
3. Lists
4. Quicksort
5. Input Validation
6. Use of Bootstrap
7. Use of Thymeleaf
8. Switch
9. LocalDate
10. Git

### Technique 1: Use of H2 Database and JPA Repositories

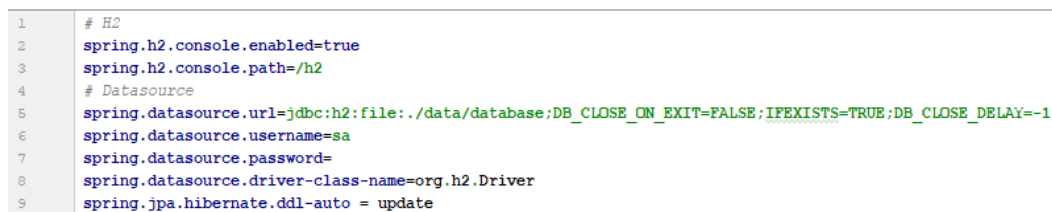
Since the purpose of the application is focused on processing data, and also one of the criteria of success is the keeping history of date of completion of tasks, a database with persistent memory was essential for the project.

The H2 database provides a browser based console application, which makes it easy to check whether the database stores what it is supposed to. H2 also allows for persistent storage after the closing of the application, which is needed in order to remember all the changes done by the client.

The application uses 4 tables in the database:



jdbc:h2:file:./data/database;DB\_C  
+ COMPLETED\_TASK  
+ MY\_USER  
+ TASK  
+ UPLOADEDDTO



```
1 # H2
2 spring.h2.console.enabled=true
3 spring.h2.console.path=/h2
4 # DataSource
5 spring.datasource.url=jdbc:h2:file:./data/database;DB_CLOSE_ON_EXIT=FALSE;IFEXISTS=TRUE;DB_CLOSE_DELAY=-1
6 spring.datasource.username=sa
7 spring.datasource.password=
8 spring.datasource.driver-class-name=org.h2.Driver
9 spring.jpa.hibernate.ddl-auto = update
```

Figure 1: H2 database set properties

For persisting objects into the database, accessing the objects, defining and executing queries, Spring JPA Repository was used.

```

package sk.silvia.projects.iassessment.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import sk.silvia.projects.iassessment.entity.MyUser;

public interface UserRepository extends JpaRepository<MyUser, String> {
    MyUser findUserByUsernameAndEncryptedPassword(String username, String password);
    MyUser findUserByUsername(String username);
}

```

Figure: Use of JPA Repository: UserRepository storing objects of type MyUser

```

public List<CompletedTask> listCompleted() {
    List<CompletedTask> completedTaskList = completedTasksRepository.findAll();
    return completedTaskList;
}

```

Figure 2: Working with repositories

## Technique 2: Security

Security of the data, so that not everyone with the link can actually access the data entered by the user is achieved by the requirement to Log in first before being able to access anything else than the “/home”, “/”, “/h2” and “/schedule/home”.

```

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests() .expressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/", "/home", "/schedule/home", "/h2").permitAll()
            .anyRequest().authenticated() .expressionUrlAuthorizationConfigurer<HttpSecurity>.Expression
            .and() .HttpSecurity
            .formLogin() .FormLoginConfigurer<HttpSecurity>
            .loginPage("/login") .FormLoginConfigurer<HttpSecurity>
            .permitAll() .FormLoginConfigurer<HttpSecurity>
            .and() .HttpSecurity
            .logout() .LogoutConfigurer<HttpSecurity>
            .permitAll();
    }
}

```

Security of the data is increased by the storing of the password in the hashed form. The hashing is done using sha1Hex, which is a cryptographic hash function, more secure than encryption offered by the Spring boot security.

```

public void registerUser(String username, String password) {
    String cryptePassword = DigestUtils.sha1Hex(password);

    MyUser user = new MyUser();
    user.setUsername(username);
    user.setPassword(cryptePassword);

    userRepository.save(user);
}

```

Figure 3: Use of hashing function sha1Hex

### **Technique 3: Lists**

The code features numerous uses of Lists. They were chosen as the most appropriate for storing multiple objects of the same type, since in the cases of their use in this project, the number of objects to be stored is not known and changes throughout the time. Also, the deletion and insertion are more easily carried out when working with Lists rather than with arrays.

Examples of functions using Lists:

- public List<Task> getSelectedTaskList()
- public List<Task> loadSelected()
- public void setTasksForUploadFalse()

```
public List<Task> getSelectedTaskList() {
    List<Task> selectedTasksList = new LinkedList<>();
    List<Task> allTasks = taskRepository.findAll();
    for(int i = 0; i < allTasks.size(); i++) {
        if (allTasks.get(i).isSelected() == true)
            selectedTasksList.add(allTasks.get(i));
    }
    return selectedTasksList;
}
```

Figure 4: Example of the use of Lists in the function getSelectedTaskList()

### **Technique 4: Quicksort**

When generating a schedule, it is needed to sort Tasks according to their duration in an ascending order. For this purpose, Quicksort, with its average complexity  $O(n \log n)$  and best case scenario complexity  $O(n \log n)$  was used (Quicksort, 2017), which is appropriate for the application.

```
public static int partition(Task arr[], int low, int high) {
    int pivot = arr[high].getDuration();
    int i = (low-1);
    for (int j=low; j<high; j++) {
        if (arr[j].getDuration() <= pivot) {
            i++;
            Task temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    Task temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;
    return i+1;
}

public static void sort(Task arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}
```

Figure 5: Implementation of quick sort sorting algorithm (code implemented based on (Quicksort, 2017))

## Technique 5: Validating inputs

### a) Try Catch in Java – Exception handling

```
public boolean validateInputInt(String input) {
    if(input.isEmpty()) return false;
    try {
        Integer.parseInt(input);
    } catch (NumberFormatException e) {
        return false;
    }
    return true;
}
```

Figure 6: Code of function boolean validateInputInt(String input) from class Validation, package: service

### b) possible exception scenarios

```
if (user == null)
    return false;
if (!(validation.validateString(newUsername) || validation.validateString(newPassword)))
    return false;
```

Figure: Code of function boolean changeLoginCredentials(String oldUsername, String newUsername, String oldPassword, String newPassword) from class UserService, package: service

### c) User chooses from an already defined set of possibilities

The image shows a snippet of HTML code defining five radio button options for task categories. Below the code is a visual representation of the form with five radio buttons labeled 'School', 'Work', 'Household', 'Free time', and 'Other'. The 'Other' radio button is selected, indicated by a filled circle.

```
<div class="d-flex flex-row justify-content-center align-content-center">
<div class = "p-2"><input id = "School" type="radio" value="School" th:field="*(categorySelected)" /><label for="School">School</label></div>
<div class = "p-2"><input id = "Work" type="radio" value="Work" th:field="*(categorySelected)" /><label for="Work">Work</label></div>
<div class = "p-2"><input id = "Household" type="radio" value="Household" th:field="*(categorySelected)" /><label for="Household">Household</label></div>
<div class = "p-2"><input id = "Free" type="radio" value="Free time" th:field="*(categorySelected)" /><label for="Free">Free time</label></div>
<div class = "p-2"><input id = "Other" type="radio" value="Other" th:field="*(categorySelected)" /><label for="Other">Other</label></div>
</div>
```

4      Category of tasks

School   Work   Household   Free time   **Other**

Figure 7: Code from generateSchedule.html

Multiple methods of ensuring that the input data from the user is in the correct form were incorporated in the project, such as Try Catch blocks, exception handling, assuring that the situations with a possibility for an error are accounted for, or even instead of letting the user to write the input, make him choose from a defined set of answers.

### **Technique 6: Use of Bootstrap**

Bootstrap is a popular front-end framework. In the project Bootstrap design was used for layout, tables, form, buttons and more, allowing for a more elaborate, but still simple appearance of components on the page, achieving so in little code.

```
<div class="d-flex flex-row justify-content-center align-content-center">
  <div class = "p-2"><input id = "School" type="radio" value="School" th:field="*{taskCategory}" /><label for="School">School</label></div>
  <div class = "p-2"><input id = "Work" type="radio" value="Work" th:field="*{taskCategory}" /><label for="Work">Work</label></div>
  <div class = "p-2"><input id = "Household" type="radio" value="Household" th:field="*{taskCategory}" /><label for="Household">Household</label></div>
  <div class = "p-2"><input id = "Free" type="radio" value="Free time" th:field="*{taskCategory}" /><label for="Free">Free time</label></div>
  <div class = "p-2"><input id = "Other" type="radio" value="Other" th:field="*{taskCategory}" /><label for="Other">Other</label></div>
</div>

<button type="submit" name="action" value="delete" class="btn btn-dark"><i class="far fa-trash-alt buttons"></i></button> </div>
```

Figure 8: Example of code for flex layout and buttons, for which Bootstrap was used

### **Technique 7: Use of Thymeleaf**

Thymeleaf is a Java library and a server-side template system, which applies a set of transformations to template files in order to display data produced by applications (Thymeleaf, 2017).

Thymeleaf, with its `th:` notation, was often used for iterating through a list, using conditionals statements, or post http requests.

- a) `<td th:text="${valid} ? ' ' : 'Wrong inputs. Try again.'" />`
- b) `<div class="tasksFont" th:each="task : ${taskList}"> ... </div>`
- c) `<form th:action="@{/login}" method="post"> ...`

Figure 9: Uses of Thymeleaf in the html files

### **Technique 8: Use of switch**

A switch was used numerous times throughout the project to get a clear, easy to follow, piece of code.

```

public void editingTask(Long id, String function) {
    switch(function) {
        case "delete":
            Task task = taskRepository.getOne(id);
            taskRepository.delete(task);
            break;
        case "completed":
            Task task1 = taskRepository.getOne(id);
            CompletedTask completedTask = new CompletedTask(task1);
            completedTask.setTodayAsDate();
            completedTasksRepository.save(completedTask);
            taskRepository.delete(task1);
            break;
        case "selected":
            Task task2 = taskRepository.getOne(id);
            if (task2.isSelected())
                task2.setSelected(false);
            else
                task2.setSelected(true);
            taskRepository.save(task2);
            break;
        default:
            break;
    }
}

```

Figure 10: Use of switch in a method

### **Technique 9: Local Date**

In the class CompletedTask, two of the instance variables concern the date (private LocalDate localDate and private String localDateString).

```

public void setTodayAsDate() {
    localDate = LocalDate.now();
    localDateString = localDate.format(DateTimeFormatter.ofPattern("dd/MM/yyyy"));
}

```

Figure 11: Use of Local Date in a method

### **Technique 10: Use of Git to access the file in an online form**

The project is available at <https://github.com/silviaGoldasova/CS-IA.git>. The online version was used to make development easier (possible on both my notebook and PC due to sharing possibility), but can be also used in future for accessing the project both by me or someone else, allowing for more development.

## Bibliography

QuickSort (2017). Retrieved from <https://www.geeksforgeeks.org/quick-sort/>

Tutorial: Using Thymeleaf. (2017, November 5) Retrieved from <https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>