

Final Report Document

for Project Staffr

Prepared by   
Kryštof Sýkora, Marek Szeles  
ČVUT FEL SIT,   
Enterprise architectures

Version 0.4  
2. 1. 2018

# Summary

## Introduction

With Staffr, we aimed to produce a Java EE based Maven compilable program that allows for staff administration to a company representative with appropriate rights.

In effect, most of the program is heavily inspired by the provided “reporting tool” and “ear-setup” repositories, with other functions added to that base according to our assignment.

## Development report

In the

# Used Technologies

For most of the program, we were inspired by the “reporting tool” concept, so we built our program on a similar base wireframe.

## Backend

### Functionality

The implemented functionality is based on the Staffr Software Requirements Specification version 1.2 from November 26th, 2017.

### Persistence layer

The persistence layer is based on the functionality described in the Staffr Software Requirements Specification.

#### Cascade persist

The main cascade persist used is implemented when persisting a user, and it allows to persist all the skills associated with it. This is done using a prePersist implementation.

In AbstractRepositoryService.java:

@Transactional  
@Override  
public void persist(T instance) {  
 Objects.*requireNonNull*(instance);  
 prePersist(instance);  
 getPrimaryDao().persist(instance);  
}

And in the UserService.java:

@Override  
void prePersist(User instance) {  
 persistSkills(instance);  
}

private void persistSkills(User instance) {  
 Optional.*ofNullable*(instance.getSkills())  
 .ifPresent(  
 skills -> skills.stream()  
 .filter(t -> t.getUser() == null)  
 .forEach(  
 t -> {  
 t.setUser(instance);  
 }  
 )  
 );  
}

This cascade is also tested in UserServiceTest.java:

@Test  
public void persistCascadesForSkills() {  
 final UserDao ud = new UserDao();  
 final UserService us=new UserService(ud);  
  
 final User user = new User("Marek","Szeles",1996,"a@b.com","",3, Status.*ACTIVE*);  
 Set<Skill> skills=new HashSet<>();  
 Skill skl\_excel\_3=new SoftSkill();  
 skl\_excel\_3.setName("Excel");  
 skl\_excel\_3.setProfficiency(SkillProfficiency.*ADVANCED*);  
 skills.add(skl\_excel\_3);  
 Skill skl\_word\_3=new SoftSkill();  
 skl\_word\_3.setName("Word");  
 skl\_word\_3.setProfficiency(SkillProfficiency.*ADVANCED*);  
 skills.add(skl\_word\_3);  
  
 user.setSkills(skills);  
 us.persist(user);  
  
 final User result = us.find(user.getId());  
 Assert.*assertEquals*(skills.size(), result.getSkills().size());  
}

#### Ordering

Ordering is used mainly when retrieving collections of users.

It is also tested by UserDaoTest.java:

@Test  
public void findAllReturnsUsersOrderedByNameDescending() {  
  
 final User Peter\_Smith = new User();  
 Peter\_Smith.setFirstName("Peter");  
 Peter\_Smith.setLastName("Smith");  
 Peter\_Smith.setEmail("P.SmithYo@Yahoo.com");  
   
 final User Charlotte\_Guido = new User();  
 Charlotte\_Guido.setFirstName("Charlotte");  
 Charlotte\_Guido.setLastName("Guido");  
 Charlotte\_Guido.setEmail("CG-see-gay@seznam.cz");  
  
 final User Ivan\_Terrible = new User();  
 Ivan\_Terrible.setFirstName("Ivan");  
 Ivan\_Terrible.setLastName("Terrible");  
 Ivan\_Terrible.setEmail("Impala@google.com");  
  
 List<User> users = new LinkedList<User>();  
 users.add(Ivan\_Terrible);  
 users.add(Charlotte\_Guido);  
 users.add(Peter\_Smith);  
  
 Collections.*shuffle*(users);  
 dao.persist(users);  
  
 final List<User> result = dao.findAll();  
 Assert.*assertEquals*(users.size(), result.size());  
 assertNameDescendingOrder(result);  
}  
  
private void assertNameDescendingOrder(List<User> users) {  
 if (users.size() == 0) {  
 return;  
 }  
  
 User previous = users.get(0);  
 for (int i = 1; i < users.size(); i++) {  
 final User current = users.get(i);  
 *assertTrue*(current.getLastName().compareTo(previous.getLastName())<=0);  
 previous = current;  
 }  
}

#### Named Querries

There are many named querries used in the DAO layer, for example, in the UserDao:

public User findByUsername(String username) {  
 try {  
 return em.createNamedQuery("User.findByUsername", User.class).setParameter("username", username)  
 .getSingleResult();  
 } catch (NoResultException e) {  
 return null;  
 }  
}

### CRUD layer

All parts of the CRUD layer are implemented, spread across the different roles:

* All users
  + Show user overview
    - C**R**UD read operation.
  + Edit own personal information
    - CR**U**D update operation.
* Project Leader only
  + Search for user according to criteria
    - C**R**UD read operation.
  + Create project pages
    - **C**RUD create operation.
  + Delete project pages
    - CRU**D** delete operation.
* Admin only
  + Register new user
    - **C**RUD create operation.
  + Remove user
    - CRU**D** delete operation.

### Transactionality

The transactionality is used in the service layer, most notably in the AbstractRepositoryService.java masterclass, for example:

@Transactional(readOnly = true)  
@Override  
public List<T> findAll() {  
 final List<T> result = getPrimaryDao().findAll();  
 result.forEach(this::postLoad);  
 return result;  
}

## Frontend

### ReactJS

The whole frontend is done in the ReactJS library, communicating with the backend using a REST interface.

## Failed technologies

### Javascript emulation

To save time, we have tried to use the DotVVM (<https://www.dotvvm.com/>) project as a framework to generate javascript, however implementations of this have failed to compile and be compatible with the main project, and so we abandoned the idea.

# Project outputs

The outputs are a Java EE Maven compilable program with pre-defined basic functionality, user manuals for it and multiple documents reporting on the development and purpose of the program.

## Further documentation, source code¨

Further documentation can be

<https://gitlab.fel.cvut.cz/B171_B6B33EAR/sykorkry>

# Installation and deployment

## Development Environment Setup

The following software needs to be installed on the system for development:

* JDK 8
* NodeJS v6 or later
* ReactJS
* Maven
* Apache Tomcat (or any other application server)

To start developing, first go to src\main\webapp and run npm install. This will download the necessary Node dependencies (they are used by the UI written in ReactJS). You can check that everything is working by running npm test.

## Storage Setup

The application uses a standard relation database. It is preconfigured to a PostgreSQL server named “Staffr\_db”, running at “localhost:5432”, and credentials “ear” / “ear”.

## Running the Application

To run the application locally, start JS compile watcher by running npm start from app/root/src/main/webapp. The watcher will recompile JS whenever a change is made to the UI code.

Running the application is simple, just build it with maven and deploy the artifact into you application server.

# Prepared Sample User Walkthrough

## Booting the application

## Login

Credentials:

This user is a project leader

## Show own user page

## Edit personal information

## Search for user according to criteria

## Create project page

## Delete project page