Orange = postpone to v.2.

REST API

**Use cases**

The below paths are relative the base path “/rest”.

1. Get(”/”) or Get(“/resumes”) displays a list of resumes. As default all resumes shows up. In v.2 the results are pagable, showing only some results at a time.
2. Post(“resumes/search”) dispalys a list of resumes filtered by search criteria. As default all resumes shows up. In v.2 the results are pagable, showing only some results at a time. Redirects to “/resumes/” with results in the model.
3. Get(“/resumes/{id}”) shows details of one resume.
4. Post(“/resumes”) saves a new resume for a given user and returns the saved entity.
5. Put(“/resumes/{id}”) updates a specified resume. The user id of the resume cannot be changed. It returns the saved entity.
6. Get(“users/{id}/resumes”) gets a list of resumes for a given user.
7. GET(“/User”) gets all users and resumes. Or only user data?
8. Post(“/user/) saves a user. It returns the saved entity.
9. Put(“/user/{id}”) updates a user. It returns the saved entity.
10. Get(“/skills”) shows all availbale skills available to choose from
11. Post(“/skills”) saves a new skill to be available to choose from. It returns the saved entity.
12. Put(“/skills/{id}”) updates a skill. It returns the saved entity.
13. Post(“rest/checkCredentials”) checks credentials and returns status 401 or 200..
14. An external developer can show the REST api docs at “/rest-docs
15. HATEOAS implemented vs 2

**Scenarios**

1. GET(“/”, /resumes”) is mapped to ResumeRESTCtonroller.getResumes() which calls resumeService.findAll() and retunrs result.
2. Post(“resumes/search”) calls resumeService.search(searchCriteriaDTO). SearchCrietriaDTO (firstnameIn, lastnameIn, skillsIn, freeText) contains free text fields. .search() finds the resume where 1) person has firstNameIn in the name, 2) lastnameIn in the lastname, 3) freeText in either the resume.presentation.shordescription, .longDescrption or any of the skills. Returns result as list.
3. GET(“resumes/{id}”) is mapped to ResumesRESTController.getResume(Long id) which returns the resume with id == id. Gives 404 if id not found.
4. Post(“resumes)” is mapped to ResumesRESTController.createNewResume(ResumeDTO) which calls resumeService.createNew (Resume) after converting DTO to Resume. It returns the saved entity.
5. Put(“resumes/{id}”) is mapepdto ResumeRESTController.updateResume(resumeDTO) which calleds ResumeService.updateResume(Resume) after converting DTO to Resume. It returns the saved entity.
6. GET(“resumes/users/{id}”) is mapped to ResumeRESTController.getResumesbyUserId(Long userId) which calls ResumeService.getByUserId(userId).
7. Get(“users/{id}”) is mapped to UserRESTController.getUserById(Long id), returns a user or 404 not found.
8. POST(“users”) is mapped to USerRESTController.createNewUser(UserDTO), calls UserService.cretenewUSer(User) after converting from DTO. It returns the saved entity.
9. /rest-docs shold be mapped to Swagger docs as generated by swagger.

**Acceptance tests (REST api)**

**Test 2.01: Given 3 resumes in the service, access GET(rest/resumes) and expect status 200 OK and returned list contains is of size 3.**

**Test 2.03: Access rest/mumboJumbo and expect status 404 – not found.**

**Test 2.99: Access rest/resumes/search and expect status 200 OK**

**Common assumptions for search tests below**

(Called Search\_Test\_Conditions in the following)

Resume1 = {presentation = {shortdescription=“Very skilled developer”, longDescription=“I am a very very good developer”}, skills= {name=“java”,level=4}, anyAssignments, anyEducation }

Resume2 = {presentation = {shortdescription=“Very good developer”, longDescription=“I am a very very skilled developer”}, skills= {name=“java”,level=4}, anyAssignments, anyEducation }

Resume3 = {presentation = {shortdescription=“Very nice developer”, longDescription=“I am a very very nice developer”}, skills= {name=“java”,level=4}, anyAssignments, anyEducation }

Given Persons = {{firstName=”Pelle”, lastName=”Persson”, anyContact, {resume1, resume2},

{firstName=”Kalle”, lastName=”Karlsson”, anyContact, {resume3}}

**Test 3 – Search hit on firstname**

Given data Search\_Test\_Conditions abo ve and

SearchCriteria = {firstNameIn : “pel”, lastNameIn=””, freeText=””}

when resumeService.search(searchCriteria) is called, it should return 2 results (resume 1, resume2)

Test 4 – Search hit on lastname.

**Given data Search\_Test\_Conditions above and**

When SearchCriteria = {firstNameIn : “”, lastNameIn=”person”, freeText=””}

is called, it should return 2 results (resume 1, resume2)

**Test 4 – Search hit on free text shortdescrption**

Given data Search\_Test\_Conditions above and

When SearchCriteria = {firstNameIn : “”, lastNameIn=””, freeText=”skilled”}

is called, it should return 1 result (resume 1)

**Test 4 – Search hit on free text longdescrption**

Given data Search\_Test\_Conditions above and

When SearchCriteria = {firstNameIn : “”, lastNameIn=””, freeText=”good”}

is called, it should return 1 result (resume 2)

**Test 5 – Search hit on free text lastname of two different persons**

Given data Search\_Test\_Conditions above and

When SearchCriteria = {firstNameIn : “”, lastNameIn=”sson”, freeText=””}

is called, it should return 3 results (resume 1,2,3)

**Test 6 – Search hit on free text from resume short descriptions of two different persons**

Given data Search\_Test\_Conditions above and

When SearchCriteria = {firstNameIn : “”, lastNameIn=””, freeText=”developer”}

is called, it should return 3 results (resume 1,2,3)

**Test 10 – access rest/resumes/{id} with exting resume id**

**Test 11 – access /rest/resumes/{id} with non-existing id**

**Test 12 – Post to “/rest/resumes” with valid resumeDTO**

**Test 13 – Post to /rest/resumes with invalid resumeDTO**

**Test 14 – Put to rest/resumes/{id} with invalid id**

**Test 15 – Put to rest/resumes/{id} with valid id and valid data**

**Test 16 - Put to rest/resumes/{id} with valid id and invalid data (some invalid property)**

**Test 17 – Get rest/users/{id} with valid user id**

**Test 18 – Get rest/users/{id} with invalid user id**

**Test 19 – Get rest/users/{id}/resumes with valid user id**

**Test 20 – Post to rest/users with valid user data**

**Test 21 – Post to rest/users with invalid user data**

**Test 22 – Acess Swagger docs at /rest-docs**

**Workflow - REST**

Create data model (done)

Create DTOs (done)

Mappers (done)

UserService

ResumeService

Client

**Use cases**

1. An external user browses resumes at “/” or ”/resumes”. As default all resumes shows up. In v.2 the results are pagable, showing only some results at a time.
2. At ”/” or “/resumes”, the external user can filter by perform free text search for firstname, lastname and/or skills.
3. When user clicks a resume row, this shows details of that resume + person summary at “/resumes/{id}. The user can go back to “/resumes” by clicking “back”.
4. An external user can click a “login” link to log in. When entering credentials (user1, user1pass) the user is logged in and is directed to “/users/1 which shows editable details of user1 (name, address, picture and so on) and a list of his resumes. The user can edit the user details.
5. A logged in user can click a resume and the resume details are shows. All details can be edited and saved.
6. A logged in user can “log out” by click in the “log out” link that is always visible.
7. If a user tries to access a non-existing URL (404), an error page will be shows displaying “Oops, that page does not exist!” and provides a link to “/”.
8. If a user enters invalid data in search fields, a label above the field displays a validation error message.
9. If some other error occurs, a error page is displayed with message “Oops, an error occurred!” with a link back to “/”.

**Scenarios**

1a. GET(“/”,”/resumes”) are mapped to ResumeClientController.searchResumes(SearchCriteria) which calls the API GET(“/rest/resumes”) to get a list of all resumes. It returns the view name “searchResumes”. In v2. The results are paginated.

1b. searchResumes.html is a thymeleaf template showing a table of resume DTOs (resume title, person firstname, lastname, short list of skills)

2. POST(“/” or “/resumes”) are mapped to ResumeClientController.searchResumes(SearchCriteria) which calls the API POST(“/rest/resumes”) to get a filtered list of resume DTOs (including person name). It returns new viewname “searchResumes” with the model populated with the list of resumes.

3. Get(“resumes/{id}” is mapped to ResumesClientController.getResume(Long id) which calls the API at Get(“/rest/resumes/{id}” to get the details of a resume and returns the view “showResume.html” with the model attribute “resume” populated with the resume DTO (which including name and picture of person).

3b. showResume.html is a thymeleaf template showing a profile picture, firstname, lastname, list of assignments (title + description) and list of skills + skill grades. A link “back” is provided to URL “/”.

4. The “login” link redirects to view “login”. “

4b. “login.html” contains a name/password textfield and login button. The button performs a POST(“/login”). A link “back” redirects to “/”. A red label above the text fields shows the value of model property “loginmessage”.

4c. POST(“/login”) is mapped to LoginClientController.login(UserCredentials credientials) which performs POST(“/rest/checkCredentials”). If the http response code coming from the api is 401 – unauthorized, then a redicection to GET(“/login”) is performed with model attribute “loginmessage”=”Invalid credentials. Please try again. If the http response code from the api is 200 – OK then a rediction to “/users/{id}”, where id always has value 1 (for the sake of the demo).

5. GET(/users/{id}) is mapped to UserClientController.getUser(Long id) which gets a userDTO from GET(“/rest/users/{id}”), the users resumes from GET(“/rest/users/{id}/resumes) and returns the view “user”.

5b. “user.html” is a thymeleaf template showing user details (firstname, lastname, picture) and a list of resumes for that user.

Views

searchResumes.html

showResume.html

User.html

Login.html

404NotFound.html

401Unauthorized.html