02. Unit SEN

Test task for Front-end Developer

Author: Svetlin Saev

Version: 5
Date: 15-Dec-2016 16:04

Table of Contents

1 Contents

2 Overview

This document describes the standard test task that is used to evaluate the skills and framework knowledge of applicants for the Front-end Developer position.

3 Goal

The goal of the task is to evaluate a number of criteria that are important for a Front-end Developer's daily job performance:

- Ability to understand business requirements.
- Ability to design a solution based on business requirements.
- Understanding of and proficiency in the tools and technologies that are commonly used in Westernacher's solution.
- Coding style.
- Eye for detail, correctness, ease of use.

3.1 Task description

3.1.1 Business requirements

We're building a small single page web application that manages user accounts in a database via RESTful web services. The application has the following functionality:

- A list page that shows all user accounts in a sortable grid.
- Create new account dialog that adds new accounts to the database. The following properties should be supported
    - First name
    - Last name
    - Email address
    - Date of birth
- A Delete account action that removes accounts from the database.
- (optional). An Edit account functionality that allows to change account data of existing accounts.

Note that we don't accept any questions regarding these requirements. We'll leave the design of the solution up to you.

3.1.2 Architecture

The solution should use an architecture that's similar to the way applications are built at Westernacher. This means:

- A REST oriented architecture. Communication between the frontend and the backend services should only be done using RESTful HTTP services, described in Appendix A.
- The front end should use ECMAScript 6, ECMAScript 2016 or TypeScript.
- The front end should use a Javascript Framework, e.g. AngularJS 1 or 2, extJS, etc.
- The front end should be independent, deployable application.
- Feel free to use a CSS framework like Bootstrap to do the styling.
- Feel free to use a CSS pre-processor like SASS, LESS, etc.

3.1.3 Build

The front end should be independent, deployable application. You can use any package manager (f.e. npm), any task runner (f.e. Grunt) and/or any module bundler (f.e. webpack). Tests should be executed over any runner (f.e. Karma) using any testing framework (f.e. Jasmine).

Error handling or offline mode when back end is down will be highly appreciated.

3.1.4 System environment

The environment components should come from this list:

- ECMAScript 6, ECMAScript 2016 or TypeScript
- Windows 7/8/10, Mac OS X 10.9 or higher, Ubuntu LTS 12.04 or higher

3.2 Documentation

Please provide a short documentation that explains how we can build and deploy the app for testing, based on the source code. Make sure you list the system prerequisites for running the application.

We don't need end user or code documentation, we'll figure that out ourselves.

3.3 Final thoughts

Try to make sure that your application is an example of a state-of-the-art single page web application, using RESTful oriented architecture, with a good-looking, intuitive UI. These are the kinds of applications we're building for our customers so you should convince us that you're the right person to join our team.

Try to keep the codebase as compact as possible. Try programming defensively.

Be creative. If you think you found something nice you want in the application to show us your skills: go ahead

3.4 Reward

We don't want you to create this test application for free! If your work is approved by our reviewers and you pass your trial period at Westernacher, we'll pay you a **bonus of 500 LEV as a compensation for your efforts**.

4 Appendix A:

Please download the Back End Java App from: https://owncloud.westernacher.com/index.php/s/Ry9eR4oue44pMOt

To run the app you should have Java installed on your machine, locate yourself in the folder, where you downloaded the jar file and execute:

```
java -jar user-management-app-0.0.1-SNAPSHOT.jar
```

This will start a RESTful oriented Spring Boot application, which will serve you as an API.

The API documentation is as follows

HTTP Verb	**URL**	**Request body**	**Response**
GET	http://localhost:8080/users	Request parameters for paging: **page**: the page of elements you would like to display. Integer between 0 and Integer. MAX_NUMBER **size**: the size of the page you would like to display. Integer between 0 and Integer.MAX_NUMBER **sort**: the sort parameter to be used, f.e. firstName comma separated with the sort direction, f.e. desc An example call to get you the first page of 10 accounts sorted by first name descending would be: http://localhost:8080/users?page=0&size=10&sort=firstName,desc	200 OK with a JSON array where each element in this array is an user object to be displayed and paging information **Example**:

HTTP Verb	URL	Request body	Response
			`{ "content": [{ "id": 1, "firstName": "Birgit", "lastName": "Prete", "email": "birgit.prete@mail.com", "dateOfBirth": "1987-12-30" }, { "id": 2, "firstName": "Valorie", "lastName": "Griffie", "email": "valorie.griffie@gmail.com", "dateOfBirth": "1987-12-30" }, ...], "last": true, "totalElements": 15, "totalPages": 1, "size": 20, "number": 0, "sort": null, "first": true, "numberOfElements": 15 }`

HTTP Verb	**URL**	**Request body**	**Response**
			or empty array if there are no resources saved

POST	http://localhost:8080/users	`{`
		`    "firstName": "first`
		`Name",`
		`    "lastName": "lastNa`
		`me",`
		`    "email": "email@gma`
		`il.com",`
		`    "dateOfBirth": "198`
		`7-12-30"`
		`}`
			201 Created with Location header of the newly created resource and JSON payload of the created user
			Example:
			JSON payload:
			`{`
			`  "id": 16,`
			`  "firstName":`
			`"firstName",`
			`  "lastName": "`
			`lastName",`
			`  "email": "ema`
			`il@gmail.com",`
			`  "dateOfBirth"`
			`: "1987-12-30"`
			`}`
			Location: http://localhost:8080/users/16/

HTTP Verb	**URL**	**Request body**	**Response**

GET	http://localhost:8080/users/16		200 OK with the JSON payload for the selected user
			Example:
			JSON payload:
			`{`
			`  "id": 16,`
			`  "firstName": "firstName",`
			`  "lastName": "lastName",`
			`  "email": "email@gmail.com",`
			`  "dateOfBirth": "1987-12-30"`
			`}`
			404 Not Found if the requested id is not found in the db.

HTTP Verb	URL	Request body	Response
PUT	http://localhost:8080/users	`{ "id": 16, "firstName": "newFirstName", "lastName": "newLastName", "email": "newEmail@gmail.com", "dateOfBirth": "1987-12-30" }`	200 OK with the updated resource and its JSON payload **Example**: JSON payload: `{ "id": 16, "firstName": "newFirstName", "lastName": "newLastName", "email": "newEmail@gmail.com", "dateOfBirth": "1987-12-30" }` 400 Bad Request if the id is not presented.
DELETE	http://localhost:8080/users/16		200 OK for the deleted resource for the given id