**Assignment 5 – Person’s API**

**Setup**

* Create a new collection and name it people.
* Add a new file under the model’s directory and name it **<yourLastName>-person.js**.
* Add a new file under the route’s directory and name it **<yourLastName>-person-routes.js**.

**<yourLastName>-person.js**

* Add a require statement for mongoose and assign it to a variable named **mongoose**.
* Add a new variable named Schema and assign it the mongoose.Schema object.
* Create a schema named roleSchema with the following fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Schema** |
| text | String | n/a |

* Create a schema named dependentSchema with the following fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Schema** |
| firstName | String | n/a |
| lastName | String | n/a |

* Create a schema named personSchema with the following fields:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Schema** |
| firstName | String | n/a |
| lastName | String | n/a |
| roles | Array | roleSchema |
| dependents | Array | dependentSchema |
| birthDate | String | n/a |

***Note: for an example, refer to student.js and node-tech-routes.js in the course’s repo.***

* Name the model “Person” and export it using module.exports

**<yourLastName>-person-routes.js**

* Add the appropriate requirement statements (express, router, and Person).
* Create a variable named router and assign it the express.Router() function.
* Create two operations: findAllPersons and createPerson

# **findAllPersons**

**Operation: GET**

**Path: /api/persons**

|  |  |  |
| --- | --- | --- |
| **Param Type** | **Field** | **Data Type** |
| n/a | n/a | n/a |

|  |  |
| --- | --- |
| **Response Code** | **Message** |
| 200 | Array of person documents |
| 500 | Server Exception |
| 501 | MongoDB Exception |

**Additional Programming Requirements**

* + Wrap the code in a try/catch block.
  + Query the people collection using the find() function on the Person model.
  + Either return an array of person documents or the appropriate message depending on the status code.
  + Describe the operation using the OpenAPI Specification above the request as developer comments.

# **createPerson**

**Operation: POST**

**Path: /api/persons**

|  |  |  |
| --- | --- | --- |
| **Param Type** | **Field** | **Data Type** |
| RequestBody | firstName | String |
| RequestBody | lastName | String |
| RequestBody | roles | Array of Role objects |
| RequestBody | dependents | Array of Dependent objects |
| RequestBody | birthDate | String |

|  |  |
| --- | --- |
| **Response Code** | **Message** |
| 200 | Array of person documents |
| 500 | Server Exception |
| 501 | MongoDB Exception |

**Additional Programming Requirements**

* + Wrap the code in a try/catch block.
  + Create a new object literal named newPerson and map the RequesBody fields to its properties.
  + Call the create() function on the Person model and add the document to MongoDB.
  + Either return the new person documents or the appropriate message depending on the status code.
  + Describe the operation using the OpenAPI Specification above the request as developer comments.
* Export the router using module.exports.
* Test one of the API’s using the generated Swagger documentation at <https://localhost:3000/api-docs> and test the second two API’s using SoapUI. Take screenshots of the testing you did in SoapUI and from the Swagger documentation link.
* The screenshots you take of SoapUI and Swagger must be added to your personal portfolio website under the “API Unit Tests Page.” This is a gradable item.
* If you run into issues, refer to the courses GitHub repository and the work you completed in WEB 340.