**Assignment 4 – Composer API**

**Setup**

* ~~Create a new database named~~ **~~web420DB~~** ~~and a collection named~~ **~~composers~~**~~.~~
* ~~Create a new database user and name it~~ **~~web420\_user~~**~~.~~
* ~~Create a custom role named~~ **~~web420Role~~** ~~and assign it to the user~~ **~~web420\_user~~**~~.~~
* ~~Add a new directory under the root of your GitHub repository and name it~~ **~~models~~**~~.~~
* ~~Add a new directory under the root of your GitHub repository and name it~~ **~~routes~~**~~.~~
* ~~Add a new file under the~~ **~~model’s~~** ~~directory and name it~~ **~~<yourLastName>-composer.js~~**~~.~~
* ~~Add a new file under the~~ **~~route’s~~** ~~directory and name it~~ **~~<yourLastName>-composer-routes.js~~**~~.~~

**<yourLastName>-composer.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 composerSchema with the following fields:~~

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

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

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

* ~~Add the appropriate requirement statements (express, router, Composer)~~
* ~~Create a variable named router and assign it the express.Router() function.~~
* Create three operations: findAllComposers, findComposerById, createComposer

# **findAllComposers**

**Operation: GET**

**Path: /api/composers**

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

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

**Additional Programming Requirements**

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

# **findComposerById**

**Operation: GET**

**Path: /api/composers/:id**

|  |  |  |
| --- | --- | --- |
| **Param Type** | **Field** | **Data Type** |
| RequestParams | id | String |

|  |  |
| --- | --- |
| **Response Code** | **Message** |
| 200 | Composer document |
| 500 | Server Exception |
| 501 | MongoDB Exception |

**Additional Programming Requirements**

* + ~~Wrap the code in a try/catch block.~~
  + ~~Query the composers collection using the findOne() function and the RequestParms id on the Composer model.~~
  + ~~Either return the found composer document or the appropriate message depending on the status code.~~
  + ~~Describe the operation using the OpenAPI Specification above the request as developer comments.~~

# **createComposer**

**Operation: POST**

**Path: /api/composers**

|  |  |  |
| --- | --- | --- |
| **Param Type** | **Field** | **Data Type** |
| RequestBody | firstName | String |
| RequestBody | lastName | String |

|  |  |
| --- | --- |
| **Response Code** | **Message** |
| 200 | Composer document |
| 500 | Server Exception |
| 501 | MongoDB Exception |

**Additional Programming Requirements**

* + ~~Wrap the code in a try/catch block.~~
  + ~~Create an object literal named composer and map the fields from the RequestBody to its properties.~~
  + ~~Call the create() function on the Composer model.~~
  + ~~Either return the newly added composer document or the appropriate message depending on the status code.~~
  + ~~Describe the operation using the OpenAPI Specification above the request as developer comments.~~
* 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. The documentation for installing and testing an API with SoapUI is located under the SoapUI Guide. 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.