CMPS 356 Enterprise Application Development - Spring 2018

**Lab 7 – Web APIs**

|  |  |
| --- | --- |
| **Student Name** |  |
| **Student Id** |  |
| **Email** |  |

**Grading Rubric - - In the *Functionality* column please specify either: *Working (completed x%)*, *Not Working (completed x%)* or *Not done*.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criteria | % | Functionality\* | Quality of the implementation | Score |
| Part A - Extend the Banking App to asynchronously read/write data from the accounts.json file and make the App functionality accessible via Web API | 30 |  |  |  |
| 1. Get /api/accounts/ Returns all accounts 2. Get /api/accounts/:id Returns an account by id 3. Post /api/accounts Adds an account 4. Put /api/accounts/:id Updates an account 5. Delete /api/accounts/:id Deletes an account by id |  |  |  |
| Part B - Extend the BookDonation App to asynchronously read/write data  from book-donation.json file and make the App functionality accessible via Web API | 70 |  |  |  |
| 1. Get /api/books?name=   Returns the book by name   1. Get /api/books?pageCount=   Returns the books with pages >= the pageCount parameters. E.g. Calling the function with pageCount=200 should return all the books with pages >= 200.   1. Get /api/books?author=   Returns all the books authored by that specific author.   1. Get /api/books?category=   Returns the books for a particular category.  E.g. Calling the function with category = Programming should return all the programming books.   1. Get /api/books/summary Returns a map that contains the author name and the number of books they have authored. E.g. 2. Post /api/books/ Adds a book 3. Put /api/books/:isbn Updates a book 4. Delete /api/books/:isbn Deletes a book |  |  |  |  |
| Total | 100 |  |  |  |
| Copying and/or plagiarism or not being able to explain or answer questions about the implementation | -100 |  |  |  |

**\* Possible grading for functionality**: ***Complete and*** ***Working*** (get 70% of the assigned grade), ***Complete and*** ***Not*** ***working*** (lose 40% of assigned grade) and ***Not done*** get 0. The remaining grade is assigned to the quality of the implementation. In case your implementation is not working then 40% of the grade will be lost and the remaining 60% will be determined based on of the code quality and how close your solution to the working implementation. Quality includes meaningful naming of identifiers, no redundant code, simple and efficient design, clean code without unnecessary files/code, use of comments where necessary, proper white space and indentation. **Marks will be reduced** forcode duplication, poor/inefficient coding practices, poor naming of identifiers and unnecessary complex/poor user interface design.

## Testing evidence