INF653 Back End Web Development - Final Project Requirements

1) PROJECT: You will build a Node.js REST API for US States data using both Express and MongoDB.

2) DATA Requirement: A states.json file will be provided in Blackboard.
    a) This file will provide MOST of the states data. There will be no need to store this data in MongoDB. You can access this data directly from the file by storing it in your project.

3) DATA Requirement: You will need to create a MongoDB collection.
    a) The collection should be represented with a States.js model file in your project.
    b) As you learned, the model file should contain a Mongoose Schema for the model.
    c) The Schema will have a stateCode property which is:
        i) a string
        ii) required
        iii) Unique
    d) The Schema will also a funfacts property which is
        i) an array that contains string data
    e) In the collection, the stateCode property will contain state abbreviation values.
    f) In the collection, the funfacts array will contain “fun facts” about the state.
    g) You are not expected to provide fun facts for all 50 states. Instead, provide at **minimum of 3** fun facts for each of the following 5 states: (this site might be handy: 50states.com)
        i) Kansas
        ii) Missouri
        iii) Oklahoma
        iv) Nebraska
        v) Colorado
    h) Look at your state.json file. The fun facts should not repeat what is in the file.
    i) Please do not add fun facts to the following 5 states (until grading is complete):
        i) New Hampshire
        ii) Rhode Island
        iii) Georgia
        iv) Arizona
        v) Montana

4) DEPLOYMENT Requirement: You will host your project with a free glitch.com account
    a) This allows for EASY deployment from Github
    b) Glitch supports environment variables.
        i) Do NOT include your .env file in your Github repository
        ii) After deploying to Glitch, you can edit your project at Glitch and add the environment variable(s) needed

5) ROOT URLs:
    a) Your root project URL should follow this pattern:
    https:/your-project-name.glitch.me/ - this will be a public HTML page

    b) The REST API root URL should start the same, but end in /states/:
    https:/your-project-name.glitch.me/states/
    Additional API endpoints should be added to the /states/ route.

    c) A catch all should be provided to serve a 404 status if the route does not exist.
        i) If the client accepts “text/html”, the response should be an HTML page.
        ii) If the client accepts “application/json”, the response { "error": "404 Not Found" }

6) Your REST API will provide responses to the following **GET** requests:

Request:	**Response:**
/states/	All state data returned
/states/?contig=true	All state data for contiguous states (Not AK or HI)
/states/?contig=false	All state data for non-contiguous states (AK, HI)
/states/:state	All data for the state URL parameter
/states/:state/funfact	A random fun fact for the state URL parameter
/states/:state/capital	{ 'state': stateName, 'capital': capitalName }
/states/:state/nickname	{ 'state': stateName, 'nickname': nickname }
/states/:state/population	{ 'state': stateName, 'population': population }
/states/:state/admission	{ 'state': stateName, 'admitted': admissionDate }

NOTES on GET routes:

1) If you have a catch all for routes that do not exist in your server, you will not need to check if URL parameters exist. If they are entered wrong, the response will be a 404.
2) The :state URL parameter above represents state codes like KS, NE, TX, NY, etc. Entering in full state names should result in a 404.
3) Check the example application to verify the exact responses expected.
4) Also *check the example application for expected messages when required parameters are not received or no fun facts are found for a requested state*.
5) Notice **contig** above is a query parameter where **:state** is a URL parameter.
6) "All state data" means all state data from states.json merged with the fun facts stored in MongoDB.

7) Your REST API will provide responses to the following **POST** request:

Request:	**Response:**
/states/:state/funfact	The result received from MongoDB

Notes:

1) The body of this POST request should contain a "**funfacts**" property providing an array providing one or more fun facts about the state. It should be possible to post all of your fun facts about a state with one POST request.
2) If the state already has some fun facts saved, submitting a POST request should add to those fun facts and not delete the pre-existing data.
3) We are not indexing fun facts or trying to determine if they already exist. Duplicate entries in the funfacts array should be avoided but are possible.

8) Your REST API will provide responses to the following **PATCH** request:

Request:	**Response (fields):**
/states/:state/funfact	The result received from MongoDB

Note:

1) The body of the PATCH submission MUST contain the index of the funfacts array element to replace and the new fun fact. Required request body properties: **index** and **funfact**
2) The index parameter value should not be zero-based. This will allow you to check if the index is sent: if (!index) etc. …afterwards, you should subtract 1 to adjust for the data array which is zero-based.

9) Your REST API will provide responses to the following **DELETE** request:

Request:	**Response (fields):**
/states/:state/funfact	The result received from MongoDB

Note:

1) The body of the DELETE submission MUST contain the index of the funfacts array element to remove. Required request body property: index
2) The index parameter value should not be zero-based. This will allow you to check if the index is sent: if (!index) etc. …afterwards, you should subtract 1 to adjust for the data array which is zero-based.

10) Your project should have a GitHub repository. I recommend the easy deployment of Glitch, but it is manual last I checked. If you prefer a pipeline, you can use Heroku or another host of your choice that provides one.

11) **Submit the following:**
    a) A link to your GitHub code repository (no code updates after the due date accepted)
    b) A link to your deployed project
    c) A one page PDF document discussing what challenges you faced while building your project.
    d) Automated testing - remember to not only run the tests but also submit your score.

12) As with the midterm, the **automated tests** for this final project are the bulk of your grade. You can test as often as you want before submitting, **BUT** remember to submit your project on the automated testing page. You will need the id you generated when submitting your midterm. Contact me if you have forgotten / lost this necessary id - asap please. (An emergency email on the night this is due will not be read in time.)

✅ Check your requests and responses with Postman: https://www.postman.com/downloads/

🚀 **For students who want an extra challenge (not required):**
Much more is possible with this API concept if you want to implement it. The states.json file provides population data that could be ranked with responses that ascend or descend. Maybe allow users to request a list of states that were only admitted after or before a specific year. Many other ideas could be generated - go for it and share with the class when you complete your project!