Requirements

Use the supplied SQLite database to create and expose a Web API similar to the one you used for Assignment 05.

1. The database contains a list of questions found from TV's Jeopardy! game show.
2. The database contains three tables with columns defined as:
    a. Table: Users
        i. Columns
            1. UserID, text datatype, e.g.
            2. UserPassword, text datatype, stored as plain text and not encrypted
    b. Table: Questions
        i. Columns
            1. QuestionID, integer datatype, primary key, e.g. 50123
            2. CategoryCode, text datatype, foreign key into the Categories table, e.g. "HIS"
            3. AirDate, date datatype, stored with year, month, and date only (no time), e.g. 2004-12-31
            4. ShowNumber, integer datatype, e.g. 4680
            5. DollarValue, integer datatype, e.g. 200
            6. QuestionText, text datatype, e.g. "'In the title of an Aesop fable, this insect shared billing with a grasshopper'"
            7. AnswerText, text datatype, e.g. "the ant"
    c. Table: Categories
        i. Columns
            1. CategoryCode, text datatype, primary key, e.g. "HIS"
            2. CategoryTitle, text datatype, e.g. "HISTORY"
3. Implement your Web API using Node.js and SQLite as demonstrated in class.
    a. You can make use of the following npm packages: Express, body-parser, and sqlite3. If you'd like to use any additional packages, please check with one of the TAs first.
4. All endpoints should use JSON for sending and receiving data.
5. Your web API should expose the endpoints as described below.
6. Your server should run on the port specified in the PORT environment variable (process.env.PORT in node). If the environment variable is not set, default to port 8000,

Testing

For this assignment, you do not have to build and submit any sort of user interface since it is a back-end only assignment. Of course, you should test your Web API and can do so by creating your own testing UI, by using a web API testing app (e.g. Postman), or even by using a web browser for testing GET requests.

Special Note for future assignments

For Assignment 07 we will be adding major enhancements to the API for user authentication, and further CRUD operations.

Additionally, we will be adding an optional "bonus" assignment 7.5 that will allow you to earn up to 50 points to replace lost points from other homework assignments. Assignment 7.5 will require you to create a UI for your Assignment 07 web API *and* allow you to use the Bootstrap HTML/CSS/JavaScript templates and libraries when creating your UI (see https://www.w3schools.com/bootstrap/default.asp). So even though it is not required for Assignment 06, you may want to begin working on a web UI that can be used for testing your work on Assignment 06 so far.

Web API

Endpoint	Get/Post	Description
/auth/signin	POST	Validates the sent User ID and Password against the entries in the Users table. If a match is found, return a message of "success". Otherwise send an appropriate error message. Note that User ID should be a case insensitive compare but Password should be a case sensitive compare. **Request Format:** Accept requests using JSON-encoded data. The "userID" and "password" parameters should be included in the request body (not the url). An example request might look like this: `{ "userID": "myUsernameHere", "password": "myPasswordHere" }` **Response Format:** If the username is valid, return the following response with a 200 status code: `{ message: "success" }` If the username or password is missing, return the following response with a 400 status code: `{ message: "invalid_data" }` If the username or password are present, but not correct, return the following response with a

		401 status code: `{ message: “invalid_credentials” }`
/questions	GET	Queries the Questions table (joined with the Categories table) and returns a list of questions matching the search criteria. ● Pass search criteria as query string values. ● All search criteria is optional but your code should allow searches by a case insensitive exact matching Category Title, exact matching Dollar Value, and Question Text containing a character string. E.g. “I want all questions with $200 value that contain the string “grass” from category “HISTORY”. ● Entries should be returned sorted by Air Date in descending order (i.e. Most Recent First). ● Return at most 5000 matching questions. If the search would return more than 5000 questions, return an appropriate error message and do not return any matching questions. Example return JSON for a single entry: `{ "categoryTitle": "3-LETTER WORDS", "airDate": "2004-12-31", "questionText": "'In the title of an Aesop fable, this insect shared billing with a grasshopper'", "dollarValue": 200, "answerText": "the ant", "showNumber": 4680 }` **Request Format** Allow any of the following search criteria to be passed via url params: ● categoryTitle (a string) ● airDate (should follow the format YYYY-MM-DD) ● questionText (a string) ● dollarValue (a number) ● answerText (a string) ● showNumber (a number) **Response Format** For successful responses, please use the following format with a 200 status code: `[ {`

```
    "categoryTitle": "3-LETTER WORDS",
    "airDate": "2004-12-31",
    "questionText": "In the title of an Aesop fable, this insect
shared billing with a grasshopper",
    "dollarValue": 200,
    "answerText": "the ant",
    "showNumber": 4680
},
{
    "categoryTitle": "ANIMALS",
    "airDate": "2003-11-12",
    "questionText": "This animal is fluffy",
    "dollarValue": 500,
    "answerText": "The Panda",
    "showNumber": 1234
},
... etc... (in descending order)
]
```

If the user passes invalid data, then use the following format with a 400 status code:

```
{
    message: "invalid_data"
}
```

If the result set would be longer than 5000 results, then use the following format with a 400 status code:

```
{
    message: "too_many_results"
}
```

Due Date

Submit your assignment using Vocareum before **Thursday, November 2nd, 11:00 pm.** Please make sure to include a valid package.json file, and do not submit the node_modules folder.

Scoring

Max Points	**Requirement**
0	Correct file submission, coding style and readability, etc. *Note points may be deducted for poor file submission, coding style, readability, etc.*
5	Correct use of JSON, URL encoding, and error handling.
10	Correct implementation of the Sign In API
20	Correct implementation of the Questions GET API
35	**Total**