# Digipog Exam Prep Project

1. Create a JSON file with two properties
   1. ‘students’ starts as an empty array
   2. ‘sequence’ starts at 0
2. Install and import ‘fs’, ‘expressJS’, and ‘EJS’ modules
3. Use ‘fs’ to read the JSON file above, and parse it to a JS object called “studentData”
4. Create a new class called ‘Student’
   1. The constructor takes the arguments ‘name’ and ‘seqeunce’
   2. The constructor sets this object’s ‘name’ property to ‘name’
   3. The constructor sets this object’s ‘id’ property to ‘sequence’
   4. The constructor sets this object’s ‘balance’ property to 0
5. Create an ExpressJS application object
6. Configure the ExpressJS application to use EJS as a view engine
7. Configure the ExpressJS application to use URL encoding
8. Create an ExpressJS HTTP listen server
9. Create a GET endpoint called “/view”
   1. If the query parameter “studentid” was not present, respond with an EJS template containing an unordered list of links to view each student by sending the ‘studentid’ query parameter for each student
   2. If the query parameter ‘studentid’ is present, use ‘find()’ to find the student with the ‘id’ property equal in value to the ‘studentid’ query parameter
      1. If find() returns a valid student object, respond with an EJS template that contains the student’s name, their ‘balance’, a link back to ‘/view’, and an HTML form with the following fields:
         1. ‘id’: hidden input with the student’s ‘id’ property as the value
         2. ‘change’: text input that only takes numbers
         3. ‘action’: select input whose options’ values are ‘add’ and ‘subtract’
         4. A “submit” button that POSTs to ‘/edit’
      2. If find() does not return a student object, respond with an EJS template that tells the user “Student not valid”, with a link back to “/view”
10. Create a POST endpoint called “/edit”
    1. If the body parameters ‘id’, ‘change’ and ‘action’ are present:
       1. Use find() to find the student with the ‘id’ property equal to the ‘id’ body parameter
          1. If find() returns a valid student object
             1. If ‘action’ had the value ‘add’, add the ‘change’ value to the student’s ‘balance’ property
             2. If the ‘action’ had the value ‘subtract’ subtract the ‘change’ value to the student’s ‘balance’ property
             3. Stringify the ‘studentData’ object and use ‘fs’ to write to the JSON file above
          2. If find() does not return a valid student object
             1. Respond with the EJS template you made for “/view”, which tells the user “Student not valid” with a link back to “/view”
11. Create a GET endpoint called “/new”
    1. Respond with an EJS template with an HTML form containing the following:
       1. A text input named “studentName”
       2. A submit button that POSTS to “/new”
12. Create a POST endpoint called “/new”
    1. If the ‘studentName’ body parameter was present, create a new “Student” object and pass it the arguments ‘studentName’ and studentData’s ‘sequence’ property incremented by 1 ( ***studentData.sequence++*** ).
       1. Push this new Student object to the studentData’s ‘students’ array
       2. Stringify the studentData and use ‘fs’ to write to the JSON file above
    2. If there is no ‘studentName’, respond with the EJS template you made for “/view”, which tells the user “Student not valid” with a link back to “/view”
13. Print your application’s main JS file (for example, ‘app.js’ or ‘index.js’)