

Exam 2: Problem Reference Booklet

This is the Exam Reference Booklet for the following Problems.

You will not get any credit for written work in this booklet.

- **Problems 2/3 (Client/Server JS) Books on a Budget**
 - Overview
 - Project Directory and `<course.txt>` File Details
 - Textbook API Documentation
- **Problem 4 (SQL): Who needs .txt-books anyways?**
 - **booksdb** database overview

Problem 2/3 Overview

With textbook prices on the rise, the team that brought you *Tricky Typing* has begun work on their next big project: a full-stack website designed to help you find the cheapest available textbooks for your courses.

In Problem 2, you will write the server-side **app.js** in Node/Express to implement the API following a provided specification. This API will then be used in Problem 3 (client-side `books.js`) to power the textbook search page (`books.html`). For both problems, you may assume the other is implemented correctly, and may choose to implement them in either order. On the rest of this page and the following page, we provide the Project Directory Structure as well as the reference API documentation. You will then find instructions for Problems 2/3.

Book Finder

A Dog's Guide to Web Programming

Available at University Book Store \$199.99

Select a course:

CSE142

CSE154

HCDE308

PSYCH200

Project Directory Structure

For both problems, you should assume the following directory structure, where the front-end views are served from `public/` similar to the other full-stack projects you've worked on in Modules 4/5.

Each course has exactly one **.txt** file in the **data** directory, named in the format **<course>.txt** with the first line being the full name of the required textbook and all following lines having the format **seller:price** to pair each seller known offering that textbook with their offered price (not assumed to be in any particular order). In this version of our API, we assume all courses in the **data** directly have only one textbook required.

<p>Directory structure</p> <pre>book-finder/ ├── app.js │ └── data/ │ ├── CSE142.txt │ ├── CSE154.txt │ ├── HCDE308.txt │ ├── PSYCH200.txt │ └── ... └── public/ ├── books.html ├── books.js └── styles.css</pre>	<p><course.txt> File Content Format</p> <pre>bookname seller1:price1 ... sellerN:priceN</pre> <p>Example CSE154.txt contents</p> <pre>A Dog's Guide to Web Programming University Book Store:199.99 PetCo:200.01</pre> <p>Example HCDE308.txt contents</p> <pre>Design of Everyday Things amazon:10.43</pre> <p>Example PSYCH200.txt contents</p> <pre>Psychology of Dogs PetCo:19.99 Real Penny Dealz:0.01</pre>
--	---

For each `<course>.txt` file, you should assume:

- The only occurrence of `:` is between the seller name and their price for the book. Use this delimiter to split each line string into an array of two strings.
- There are no duplicate sellers in the same file.
- All prices are formatted non-negative numbers with exactly 2 digits after the decimal. A price less than \$1.00 will always have a single 0 before the decimal.

Textbooks API Documentation

The textbooks API will be implemented in **app.js** and supports two **GET** requests - one returning plain text data, and one returning JSON data. All specified errors are sent as plain text, and any file- or directory- processing error should return a 500 error status with the message “Something went wrong on the server, please try again later.” Other specific details for each GET request are described below:

Endpoint 1: Get all Course Names

Request Format: **GET /courses**

Response Content Type: **plain text**

Description: Outputs a plain text response with each course’s name **on a new line**. The last line in the response **should not end** with the `\n` character (these are hidden characters, but are shown in bold in the example output for clarification):

Example Output: Below is the expected output according to a **data** directory that happens to have only 4 course txt files (but you should not assume these are always the same):

```
CSE142\n
CSE154\n
HCDE308\n
PSYCH200
```

Endpoint 2: Get the Cheapest Textbook for a Given Course

Request Format: **GET /books/:course**

Response Content Type: **JSON**

Description: Outputs a JSON response providing information on **the cheapest** available option for the course passed, where the **:course** value corresponds to a course name returned in Endpoint 1. The response should include the name of the course’s required textbook, the seller offering the cheapest option, and their offered price.

Example Request: **/books/CSE154**

Example Response:

```
{
  "name": "A Dog's Guide to Web Development",
  "seller": "University Book Store",
  "price": 199.99
}
```

Error-Handling: If a user attempts to lookup a course that does not correspond to a txt file in **data**, this endpoint should return a 400 plain text error with the message, “No data found for course.”

2. (Node.js) Books on a Budget

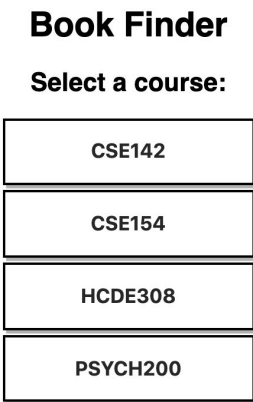
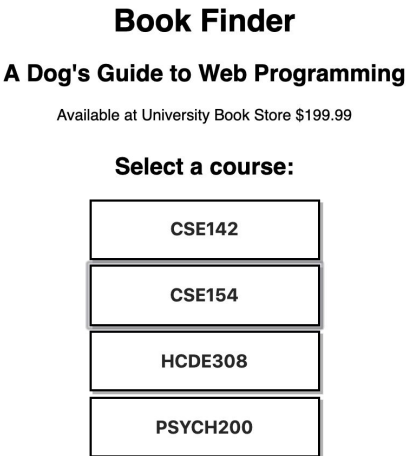
You will implement the two endpoints as Part A and Part B, completing the rest of the **app.js** provided in the Answer Booklet.

Finish app.js as specified in the Problem 2 section of the Exam 2 Answer Book.

3. (Client-side JS with AJAX)

What's an API without an interface? In this Problem, you will implement the client-side books.js to fetch from the textbooks API from Problem 2.

Provided Screenshots:

 <p>Book Finder Select a course:</p> <p>CSE142</p> <p>CSE154</p> <p>HCDE308</p> <p>PSYCH200</p> <p>Figure 1: When page is loaded, after course buttons are populated on the page (retrieved from <code>/courses</code> endpoint)</p>	 <p>Book Finder A Dog's Guide to Web Programming Available at University Book Store \$199.99</p> <p>Select a course:</p> <p>CSE142</p> <p>CSE154</p> <p>HCDE308</p> <p>PSYCH200</p> <p>Figure 2: After clicking the CSE154 button and populating the book info area.</p>
--	--

Initial Page Behavior

When the page loads, you should make a GET request to the `/courses` endpoint. Recall that this endpoint will return a list of all available courses as newline-separated text. Use this response create and append a button element for each course to `#courses` with `textContent` being the course name.

Viewing the Cheapest Textbook for a Course

When a course button is clicked, a fetch call should be made to `/books/:course`, with the respective course name passed as the `:course` path parameter. Recall that this endpoint will return the cheapest book as a JSON object, shown below. Use this information to populate `#book-name`, `#seller`, and `#price`. The `.hidden` class should be removed from `#book-info` on success.

Example response from `/books/CSE154`:

```
{
  "name": "A Dog's Guide to Web Development",
  "seller": "University Bookstore",
  "price": 199.99
}
```

If an error is thrown during any fetch request to the API, **#book-info** and **#courses** should be hidden and **#error-info** should be made visible. Remember that you may assume **checkStatus** is provided with the rest of the **id**, **qs**, **qsa**, and **gen** helper functions. These are included in the cheatsheet for reference if needed.

Provided **books.html**:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="books.js"></script>
    <link href="styles.css" rel="stylesheet" />
  </head>
  <body>
    <h1>Book Finder</h1>

    <p id="error-info" class="hidden">An error has occurred. Please refresh the page.</p>

    <div id="book-info" class="hidden">
      <h2 id="book-name"></h2>
      <p>
        Available at <span id="seller"></span>
        for $<span id="price"></span>
      </p>
    </div>
    <h2>Select a course:</h2>
    <div id="courses">
      <!-- A button for each course retrieved from the API should be added here -->
    </div>
  </body>
</html>
```

Finish books.js as specified in the Problem 3 section of the Exam 2 Answer Book.

4. (SQL) Who needs .txt-books anyways?

With the growth of their project, the Textbook Search team has decided to convert their .txt files into a MySQL database called **booksdb**. With this transition the team has expanded the capabilities of their service, enabling users to find cheap textbooks like never before.

With this improved SQL-powered service, we can expand the scope of our textbook search - this table breakdown is similar to the directory and file structure you used in Problems 2/3, but now supports courses with *multiple required books*.

The new textbook database consists of three tables (not all data shown):

books			offers				requirements		
id	name	author	offer_id	seller	price	book_id	req_id	course	book_id
1	Java for Duckies	Anonymous	1	University Book Store	199.99	1	1	CSE142	1
2	A Dog's Guide to Web Development	Dubs	2	BOOKS 4 CHEEP	0.05	1	2	CSE154	2
3	Design of Everyday Things	Don Norman	3	University Book Store	199.99	2	3	HCDE308	3
4	Psychology of Dogs	Caesar Milan	4	PetCo	200.01	2	4	PSYCH200	4
5	A Crash Course in Theoretical JS	Albert Einstein	5	AMAZON	10.43	3	5	CSE154	5
6	Where Wizards Stay Up Late	Katie Hofman	6	PetCo	19.99	4	6	CSE154	6
...				

books

The **books** table contains information for all textbooks on the service, including the name and author. The id column is a primary key.

offers

The **offers** table associates textbooks with their sellers, as well as the price offered by that seller. The offer_id column is a primary key, and the book_id column references the books table PRIMARY KEY.

requirements

The **requirements** table associates course names with their required textbooks. Remember that unlike the constraint in our **<course>.txt** files in Problems 2/3, our table now allows each course to have multiple **book_ids** (e.g. CSE154 has 4 books in the **requirements** table). The book_id column references the **books** table PRIMARY KEY.