32.0 hours

Project 11

Build a Course Rating API With Express

Why can’t I submit this project yet?

You must first complete all prior Techdegree content before you can submit this project. Additionally, you cannot submit more than one project at a time.

Submit for Peer Review

Your current activity is HTML Tables.

- Instructions
- How you'll be graded

In this project, you’ll create a REST API using Express. The API will provide a way for users to review educational courses: users can see a list of courses in a database; add courses to the database; and add reviews for a specific course.

To complete this project, you’ll use your knowledge of REST API design, Node.js, and Express to create API routes, along with Mongoose and MongoDB for data modeling, validation, and persistence.

Before you start

To prepare for this project you'll need to make sure you complete and understand these steps.

5 steps

- **Download the project files. We've supplied the following files for you to use:**
    - The `src` folder is where you'll be building your Node.js REST API. We’ve provided some files to get you started.
    - The `data/data.json` file contains seed data for your database that can be used in conjunction with the `mongoose-seeder` npm package.
    - The `index.js` file configures Express to serve the static files in the `public` folder. We've also configured the `morgan` npm package to log HTTP requests/responses to the console.
        - We’ve included a `.gitignore` file to ensure that the `node_modules` folder doesn't get pushed to your GitHub repo.
        - The `CourseAPI.postman_collection.json` file is a collection of Postman requests that you can use to test and explore your REST API.
        - The `nodemon.js` file configures the nodemon Node.js module, which we are using to run your REST API.
        - The `package.json` file is the project’s npm configuration, which includes the project’s dependencies.

- **Ensure that you have MongoDB installed.**
  - Open a `Command Prompt` (on Windows) or `Terminal` (on Mac OS X) instance and run the command `mongod` (or `sudo mongod`) to start the MongoDB daemon.
  - If that command failed then you'll need to install MongoDB.
  - How to Install MongoDB on Windows
  - How to Install MongoDB on a Mac
- **We've provided you with a basic project to get started. When developing your REST API, just add your code to the `src` folder. You'll need to run the provided Node.js Express application locally and use Postman to test your REST API.**
  - Open a `Command Prompt` (on Windows) or `Terminal` (on Mac OS X and Linux) instance and browse to the root project folder.
  - Run the command `npm install` to install the required dependencies.
  - Run the command `npm start` to run the Node.js Express application using `nodemon`. Using `nodemon` gives you the ability to leave your application running while you edit your application's code (`nodemon` will restart your Node.js application when it detects that you've made changes to your code).
  - As you build your routes, you can see what information it sends back with Postman using the preconfigured requests in the collections file included in the project.
  - You can press `Ctrl-C` to stop the Node.js REST API.
- **Postman is a great Chrome App that you will use to explore and test REST APIs. To help you jumpstart the exploration process, we've provided you with a collection of Postman requests as part of the project files. Here's how to load the provided collection into Postman:**
  - If you haven't already, install Postman. Links and instructions are available on their website at https://www.getpostman.com/.
  - Once you have Postman installed and open, click on the "Import" button in the top left hand corner of the application's window.
  - In the opened dialog, click the "Choose Files" button and browse to the folder that contains your project files.
  - Select the `CourseAPI.postman_collection.json` file.
  - You should now see the `Course API` collection in the left hand pane of the main Postman window.
  - Click on one of the available requests to load it into a tab. Click on the `Request` button to issue the request to the local server.
  - Be sure that your REST API is currently running (see the previous project step for details).
- **As you build out your REST API, you'll naturally encounter errors and unexpected behavior. Here are some reminders and suggestions on how to debug your REST API.**

- If Node.js crashed as a result of the error, you can look in the `Command Prompt` (on Windows) or `Terminal` (on Mac OS X and Linux) window and see the exception information.
- Sometimes errors don't result in exceptions, but instead are returned as `400` or `500` HTTP status codes. Errors returned from your REST API will be logged to your browser's console, so you can open your browser's web developer tools and look at the error information in the console.
- For a deeper, more detailed analysis of the state of your application, you can use Google Chrome to debug your Node.js application. Watch the Debugging Node Applications With Google Chrome workshop for more information.

Project Instructions

To complete this project, follow the instructions below. If you get stuck, ask a question in the community.

9 steps

- **Set up a database connection.**
  - Use `npm` to install Mongoose.
  - Using Mongoose, create a connection to your MongoDB database.
  - Write a message to the console if there's an error connecting to the database.
  - Write a message to the console once the connection has been successfully opened.
- **Create your Mongoose schema and models. Your database schema should match the following requirements:**
  - User
    - _id (`ObjectId`, auto-generated)
    - fullName (`String`, required)
    - emailAddress (`String`, required, must be unique and in correct format)
    - password (`String`, required)
    - Course
    - _id (`ObjectId`, auto-generated)
    - user (_id from the `users` collection)
    - title (`String`, required)
    - description (`String`, required)
    - estimatedTime (`String`)
    - materialsNeeded (`String`)
    - steps (`Array` of objects that include stepNumber (`Number`), title (`String`, required) and description (`String`, required) properties)
    - reviews (`Array` of `ObjectId` values, _id values from the `reviews` collection)
    - Review
    - _id (`ObjectId`, auto-generated)
    - user (_id from the `users` collection)
    - postedOn (`Date`, defaults to "now")
    - rating (`Number`, required, must fall between "1" and "5")
    - review (`String`)
  - Mongoose validation gives you a rich set of tools to validate user data. See http://mongoosejs.com/docs/validation.html for more information.

- **Seed your database with data.**
    - We've provided you with seed data in JSON format (see the `src/data/data.json` file) to work with the `mongoose-seeder` npm package.
    - See https://github.com/SamVerschueren/mongoose-seeder for documentation on how to use `mongoose-seeder`.
    - Important: `mongoose-seeder` requires an open connection to the database to be available, so be sure to not call your database seed code until Mongoose has successfully opened a connection to the database.
- **Create the user routes**
    - Set up the following routes (listed in the format `HTTP VERB` Route `HTTP Status Code`):
        - `GET /api/users 200` - Returns the currently authenticated user
        - `POST /api/users 201` - Creates a user, sets the `Location` header to "/", and returns no content
- **Create the course routes**
    - Set up the following routes (listed in the format `HTTP VERB` Route `HTTP Status Code`):
        - `GET /api/courses 200` - Returns the Course "_id" and "title" properties
        - `GET /api/course/:courseId 200` - Returns all Course properties and related documents for the provided course ID
        - When returning a single course for the `GET /api/courses/:courseId` route, use Mongoose population to load the related `user` and `reviews` documents.
        - `POST /api/courses 201` - Creates a course, sets the `Location` header, and returns no content
        - `PUT /api/courses/:courseId 204` - Updates a course and returns no content
        - `POST /api/courses/:courseId/reviews 201` - Creates a review for the specified course ID, sets the `Location` header to the related course, and returns no content
- **Update any `POST` and `PUT` routes to return Mongoose validation errors.**
    - Use the `next` function in each route to pass any Mongoose validation errors to Express's global error handler
    - Send the Mongoose validation error with a`400` status code to the user
- **Update the `User` model to store the user's password as a hashed value.**
    - For security reasons, we don't want to store the password property in the database as clear text.
    - Create a pre save hook on the user schema that uses the `bcrypt` npm package to hash the user's password.
    - See https://github.com/ncb000gt/node.bcrypt.js/ for more information.
- **Create an authentication method on the user model to return the user document based on their credentials**
    - Create a static method on the user schema that takes an email, password, and callback
    - The method should attempt to get the user from the database that matches the email address given.
    - If a user was found for the provided email address, then check that user's password against the password given using bcrypt.
    - If they match, then return the user document that matched the email address

  - If they don't match or a user with the email given isn't found, then pass an error object to the callback

- **Set up permissions to require users to be signed in**

  - Postman will set an `Authorization` header with each request when a user is signed in.
  - Add a middleware function that attempts to get the user credentials from Authorization header set on the request.
  - You can use the `basic-auth` npm package to parse the `Authorization' header into the user's credentials.
  - Use the authenticate static method you built on the user schema to check the credentials against the database
  - If the authenticate method returns the user, then set the user document on the request so that each following middleware function has access to it.
  - If the authenticate method returns an error, then pass it to the `next` function
  - Use this middleware in the following routes:
    - `POST /api/courses`
    - `PUT /api/courses/:courseId`
    - `GET /api/users`
    - `POST /api/courses/:courseId/reviews`

Extra Credit

To get an "exceeds" rating, you can expand on the project in the following ways:

3 steps

- **Review model**

  - Validation added to prevent a user from reviewing their own course

- **User routes**

  - Tests have been written for the following user stories:
    - When I make a request to the GET `/api/users` route with the correct credentials, the corresponding user document is returned
    - When I make a request to the GET `/api/users` route with the invalid credentials, a 401 status error is returned

- **Course routes**

  - When returning a single course for the `GET /api/courses/:courseId` route, use Mongoose deep population to return only the fullName of the related `user` on the course model and each review returned with the course model.
  - Example user object returned: `{ "_id": "wiubfh3eiu23rh89hcwib", "fullName": "Sam Smith" }` *See the Project Resources section for more information about deep population.

NOTE:

  - To get an "Exceeds Expectations" grade for this project, you'll need to complete **each** of the items in this section. See the rubric in the "**How You'll Be Graded**" tab above for details on how you'll be

graded.
  - If you’re shooting for the "Exceeds Expectations" grade, it is recommended that you mention so in your submission notes.
  - Passing grades are final. If you try for the "Exceeds Expectations" grade, but miss an item and receive a “Meets Expectations” grade, you won’t get a second chance. Exceptions can be made for items that have been misgraded in review.

Download files

Zip file

Project Resources

Course

Mongo Basics

Course

REST API Basics

Course

Build a REST API with Express

Course

User Authentication with Express and Mongo

External Link

Mongoose Documentation

External Link

Email Address Regular Expression That 99.99% Works

Need Help?

Have questions about this project? Start a discussion with the community and Treehouse staff.

Get Help